Convention de codage

Diapazen

ISEN | école d'ingénieurs

v1.0 –11/06/2013

Sommaire

Ce document est une convention de codage.

Elle a pour but de :

- Garantir l'efficacité et l'optimisation du code.
- Faciliter la maintenance et l'évolution des applications :
    - Lisibilité et cohérence du code.
    - Facilité de transfert/reprise entre développeurs.

I- Généralités

D'une manière générale les variables, noms de fonctions, noms de classes, noms de fichiers, noms de constantes etc. devront :

- Être rédigés en anglais.
- Avoir un nom explicite.
- Avoir un nom le plus court possible.

De plus, le code sera commenté à différents niveaux :

- De manière globale afin de permettre la compréhension d'un bloc (fonctionnelle).
- De manière précise (à l'intérieur des blocs) afin de permettre une compréhension en détail du code.

On utilisera toujours le tag **< ?php ?>** et non sa version abrégée pour délimiter du code PHP.

Tout code doit être compatible « E_ALL ». Il ne doit produire aucune erreur, ni « warning », ni « notice ».

II- Encodage et nom des fichiers

A- Encodage :

Les fichiers seront encodés en **UTF-8**, la base de données utilisera pour moteur InnoDB. Les requêtes SQL seront en UTF-8. (SET NAMES UTF8). La base de données utilisera de même l'encodage : utf8_general_ci.

B- Nom des fichiers

Les fichiers auront un nom clair, précis et fonctionnel. En effet le nom du fichier devra indiquer directement la fonctionnalité de celui-ci. Ils seront basés sur le modèle suivant :

fonctionnalite.extension

Pour les fichiers d'inclusion (c'est-à-dire ne contenant que des constantes etc.) on rajoutera le mot-clef **inc** entre le nom et l'extension. La base devient alors pour ces fichiers :

fonctionnalite.inc.extension

Pour les fichiers contenant une classe (on rajoutera le mot clef **class** entre le nom et l'extension. La base devient alors pour ces fichiers :

NomDeLaClasse.class.extension

III- Portée des variables

De manière générale, la portée d'une variable doit être la plus restreinte possible en fonction de son utilité.

De ce fait l'utilisation des variables de type $GLOBAL est à proscrire. Il faut préférer à cela les variables de session.

Il faut aussi éviter de récupérer des variable par $_REQUEST pour permettre de voir directement l'origine d'une variable. On utilisera plutôt $_GET ou $_POST.

Il est recommandé de ne laisser perdurer une variable que le temps nécessaire mais aussi de ne pas affecter de valeur à une variable avant le moment de son utilisation. Il faut en outre factoriser le code et augmenter la visibilité que si cela est nécessaire.

IV- Nommage

A- Nommage des méthodes, fonctions, attributs et variables

Le style d'écriture retenu est le **Lower Camel Case**[1] . On évitera les abréviations pour rendre la compréhension plus rapide. Notons que les normes énoncées dans la section *généralités* restent applicables.

Tous les attributs d'une classe seront préfixés par le caractère « m ». Exemple : mUsername

B- Nommage des constantes

Les constantes sont écrites **entièrement en majuscule**, l'underscore peut servir de séparateur entre différents mots. Bien entendu, les normes énoncées dans la section *généralités* restent applicable.

Les valeurs booléennes « false » et « true » doivent être écrites en minuscules.

C- Nommage des classes et interface

Le style d'écriture retenu est l'**Upper Camel Case**[2] . On évitera les abréviations pour rendre la compréhension plus rapide. Notons que les normes énoncées dans la section *généralités* restent applicables. L'accolade ouvrante est sur une nouvelle ligne.

Rappel : le nom du fichier et le nom de la classe seront le même (à l'exception du suffixe et de l'extension)

---

[1] Lower Camel Case : Les mots sont séparés par des majuscules. Le premier mot commence par une minuscule.
Exemple : unExempleDeVariable

[2] Upper Camel Case : Les mots sont séparés par des majuscules. Le premier mot commence par une majuscule.
Exemple : UneClasse

V- Utilisation des espaces

A- Les lignes vides

On peut utiliser des lignes vides afin d'améliorer la lisibilité du code, en effet elles peuvent permettre de regrouper des portions de code liées.

On utilise aussi des lignes vides dans les cas suivants :

- Après le cartouche (début de fichier).
- Entre les déclarations de fonctions.

B- Les espaces

Les espaces ne sont pas utilisés dans les cas suivants :

- Entre un nom de fonction et l'ouverture de ses parenthèses.
- Avant ou après le caractère de fin d'instruction « ; ».
- Après l'ouverture d'une parenthèse ou avant la fermeture d'une parenthèse.
- Avant l'ouverture ou la fermeture d'un crochet (« [» ou «] »).
- Avant chaque virgule.

Un espace doit être utilisé dans les cas suivants :

- Entre un opérateur unaire et son opérande.
- Après un « if », « elseif » , « else », « while » , « for », « foreach ».
- Après chaque virgule.
- Avant et après chaque égal d'affectation.

C- L'indentation

L'indentation sera de 4 espaces. Il faudra donc configurer l'IDE en conséquence.

D- Longueur des lignes

Les lignes doivent être limitées à 85 caractères (espaces compris). Si elles ont une longueur supérieure à la limite, il convient de les découper. Les portions remises à la ligne devront être décalées d'une indentation à partir du niveau de la première ligne de l'expression.

Dans une fonction ou méthode, le découpage de la 1ère ligne se fait au niveau de la parenthèse ouvrante.

Dans une condition, le découpage doit se faire au niveau d'un opérateur logique et non d'un opérateur de comparaison.

VI- Commentaires

On utilisera plutôt des commentaires de type bloc :

- En début de fichier : explication du but.
- Avant les déclarations de classe.
- Avant les déclarations de fonction.

Évitez les commentaires flous, évidents, inexacts, qui vont devenir obsolètes. De même évitez d'entourer les commentaires dans des boîtes dessinées avec un caractère spécial (ex. *).

On utilisera le système de documentation PHPDocs.

Chaque fichier commencera par un commentaire d'entête dont le format est spécifié ci-dessous :

```
/**
 *
 * Fichier de configuration de Diapazen
 *
 * @package     Diapazen
 * @copyright   Copyright (c) 2013, ISEN-Toulon
 * @license     http://www.gnu.org/licenses/gpl.html GNU GPL v3
 *
 * This file is part of Diapazen.
 *
 * Diapazen is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License 3 as published by
 * the Free Software Foundation.
 *
 * Diapazen is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Diapazen.  If not, see <http://www.gnu.org/licenses/>.
 *
 */
```

Chaque fonction commencera par un commentaire d'entête dont le format est spécifié ci-dessous :

```
/**
 * Short description
 *
 * Long description
 *
 * @param     type  name    short description
 * @return    type      short description
 */
```

VII- Les instructions

Une instruction est une ou plusieurs lignes de code suivies d'un point-virgule.

A- Généralités

Voici quelques règles s'appliquant à l'ensemble des instructions :

- Il n'y a qu'une seule instruction par ligne.
- Lorsqu'une instruction est écrite sur plusieurs lignes on devra bien respecter les règles d'indentation.

B- Instructions conditionnelle if/ switch

Lorsque la lisibilité n'est pas mise en cause on utilisera des structure avec des elseif plutôt qu'un switch afin d'économiser les ressources.

Les instructions if apparaitrons toujours sur plusieurs lignes, on utilisera les accolades dans tous les cas afin d'augmenter la lisibilité du code et réduire le risque d'erreur lors de l'ajout de nouvelles lignes de code.

Dans les instructions « if » composées, placer chaque accolade qui sépare un « else » ou « elseif » sur la ligne suivant.

C- Instructions de bouclage for/while/foreach

On évitera les calculs répétitifs donnant toujours le même résultat (on utilisera à la place une variable intermédiaire).

On choisira la structure la plus optimisée dans chaque cas.

D- Imbrication de fonctions

On limitera le nombre d'imbrication de fonction, c'est-à-dire l'utilisation de fonction comme paramètre d'une autre fonction, à 3 imbrications maximales.

VIII- Guillemets, inclusion et Magic numbers

Afin d'optimiser l'interprétation du code par le serveur, on préférera les guillemets simples. Les guillemets doubles seront réservés aux requêtes SQL et aux codes HTML.

On utilisera des constantes plutôt que la valeur directe lors de l'utilisation de Magic numbers[3].

Lors des inclusions, afin d'éviter d'inclure plusieurs fois les mêmes fichiers (sauf si cela est utile) on utilisera **include_once** ou **require_once.**

IX- Règles concernant la Base de Donnée

A- Nom des tables :

Toutes les tables de la base de données seront nommées de la manière la plus synthétique en décrivant leur contenu. Leur nom commencera par le préfixe « ***dpz_*** **»** .

B- Requête SQL

On utilisera au maximum les vues plutôt que les instructions directes.

Les mots clefs contenus dans les requêtes SQL devront être écrits en majuscules, permettant une meilleure compréhension du code écrit.

[3] Magic Numbers : valeur numérique utilisé pour un état précis
Ex : -1 pour une erreur lors d'une fonction sensée retourner un nombre positif

X- HTML / CSS

Les codes HTML et CSS seront totalement séparés, ceci afin de garder une clarté optimale.

A- Généralité

On utilisera les blocs nécessaires et optimaux dès que possible. L'aspect graphique des pages ne doit pas être gérer à l'aide du HTML. Par exemple, nous n'utiliserons pas de tableau pour structurer un affichage ! L'aspect sera géré par les CSS.

B- Utilisation des ID et classes

Le nom des classes et id doit se reposer sur l'aspect sémantique du bloc et non son aspect. Ainsi, la modification de l'aspect ou position liée à une classe ou un id ne causera pas de paradoxe avec son nom.

Les classes et id ayant un nom composé utiliseront le format suivant : **nom_de_classe**.

C- Optimisation du temps d'accès

Afin de permettre un plus grand affichage possible, les pages doivent être pensées pour être rapidement affiché. Pour cela quelques optimisations sont demandées :

- Usage maximal des éléments du langage plutôt que créer de nouvelles classes.
- Regrouper les règles communes a plusieurs classes, balises etc. afin d'éviter de répéter des lignes de style.
- Utiliser au maximum les effets donnés par le CSS3 plutôt que l'utilisation d'images et de code JS.
- Minimiser les sources du code JQuery (on prendra la version sans commentaires etc.).
- Utiliser des images légères et adaptées voire si possible les sprites CSS[4].

[4] sprites CSS : plusieurs images regroupé en une seule. L'affichage se fait à l'aide du CSS et ne demande plus de télécharger de nouvelles images lors d'événement comme le survol etc.

D- Indentation CSS

Les différentes propriétés CSS devront être présentées sous la forme suivante :