Vilsagame

THE FUTURE IS
IN OPEN SOURCE

>>> Normes de codage

Vilsafur

14 septembre 2016

Table des matières

Résumé

Les normes de codages sont plus des codes de bonnes conduites que des règles stricts et figées. Il faut bien comprendre qu'elles ne sont pas là pour handicaper le développeur mais bien au contraire à l'avantager lors de ces différentes phases de relecture, que ce soit pour déboguer un code ou encore améliorer ce dernier.
De plus, elles permettent au futurs contributeurs de mieux appréhender le code et de mieux le comprendre.

Un code propre et bien indenté est toujours plus facile à lire et à comprendre.
Ces différentes normes seront donc à appliquer dans l'ensemble des projets mis en place au nom de Vilsagame.

1 Règles communes

Cette partie présentera les différentes normes présentes dans la plupart des langages utilisés. Elle comportera notamment les normes concernant les classes, les variables, etc...

1.1 Organisation du code

1.1.1 Taille des fichiers

Un fichier ne devrait pas dépasser 1000 lignes de codes. Effectivement, un fichier supérieur à cette taille pourrait être difficile à relire.
Outre cela, on pourrait conclure qu'un tel fichier est l'expression d'un mauvais découpage des traitement, d'une mauvaise compréhension de la conception/développemnt orienté objet. Ils traduisent généralement la présence de classe "gestionnaire" ayant accès à toutes les données possibles et imaginables. Or chaque classe devrait être responsable des traitement qui lui sont propres.

1.1.2 Taille des méthodes

Les méthodes ne devraient pas dépasser les méthodes de plus de 100 lignes. Certains traitements peuvent néanmoins justifier un dépassement de ce nombre de lignes, mais des méthodes trop longues traduisent un mauvais découpage d'algorithme et conduisent généralement à l'obtention d'un code incompréhensible.

1.2 Nommage

1.2.1 Classes

- Être en anglais,
- Première lettre en majuscule,
- Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule,
- Donner des noms simples et descriptifs,
- Éviter les acronymes hormis les communs (Xml, Url, Html),
- N'utiliser que des lettres [a-z][A-Z] et [0-9],
- Ne jamais être un verbe ou une action.

1.2.2 Fonctions

- Être en anglais,
- Comporter que des lettres [a-z][A-Z] et [0-9],
- Mélanger des minuscules et des majuscules avec la première lettre de chaque mot en majuscule ormis pour le premier mot,
- Donner des noms simples et descriptifs.

1.2.3 Variables

- Être en anglais,
- Comporter que des lettres [a-z][A-Z] et [0-9],
- Commencer par une minuscule,
- Mélanger des minuscules et des majuscules avec la première lettre de chaque mot en majuscule ormis pour le premier mots,
- Donner des noms simples et descriptifs,

- Variable d'une seule lettre à éviter au maximum sauf dans des cas précis et locaux (tour de boucle).

1.2.4 Constantes

- Tout en majuscule,
- Séparer les mots par des underscore,
- Donner des noms simples et descriptifs,
- N'utiliser que des lettres [a-z][A-Z] et [0-9].

1.3 Lisibilité

1.3.1 Indentation du code

Avoir une bonne indentation du code, pour certain langage (notamment Python), est plus que nécessaire, elle est indispensable. Nous allons donc définir ici les normes à respecter concernant l'indentation du code, et ce, quelque soit le projet ou le langage utilisé.

- Les tabulations sont remplacés par des espaces,
- Une tabulation représente 2 caractères.

1.4 Commentaire

Les commentaires occupent une grande place dans le code, quelque soit le langage utilisé. Ils permettent de mieux appréhender un code existant. Bien mis en place, nous pouvons même utiliser des outils permettant de générer automatiquement des fichiers de documentations.
Nous allons donc voir comment les utiliser afin de pouvoir utiliser Doxygen pour la génération de ces fameux fichiers.

1.4.1 Documentation des fichiers

- \file afin de définir un bloc de documentation pour un fichier,
- \version pour indiquer la version du fichier,
- \author, une balise par auteur,
- Description du fichier.

1.4.2 Documentation des namespace

- \namespace afin de définir un bloc de documentation pour un namespace,
- Description du namespace.

1.4.3 Documentation des classes

- \class afin de définir un bloc de documentation pour une classe,
- Description de la classe,

1.4.4 Documentation des fonctions

- \fn afin de définir un bloc de documentation pour une fonction,
- \param pour chaque paramètre de la fonction,
- \return pour définir le type de retour,
- Description du fichier.

2 Règles spécifique aux langages HTML5 et CSS3

2.1 CSS

Nous utiliserons un détournement de la méthode BEM décrite dans l'article de tarh sur developpez.com : http://tarh.developpez.com/articles/2014/bonnes-pratiques-en-css-bem-et-oocss/.
En voici les principaux axes :

- Nommage des classes :
  - ComponentName
  - ComponentName.modifierName
  - ComponentName-descendantName
  - ComponentName-descendantName.modifierName
  - ComponentName.isStateOfComponent
- Utilisation de classes "transversales".
- Un composant ne contient aucune information de positionnement.
- Ne pas utiliser une classe CSS pour une ancre JavaScript, préférer une classe nommée ".js-".

2.2 SASS

Afin de faciliter l'écriture des fichiers CSS, nous utiliserons le préprocésseur SASS avec le language SCSS.
De plus, nous utiliserons les normes suivantes :

- Les variables seront sous la forme : element-propriété
- Les mixins et les fonctions seront nommés comme vue précédemment (cf 1.2.2)

3 Règles spécifique au langage PHP

Afin de respecter les standards utilisés par la majorité des développeurs, les règles cités ci-dessous seront, dans la majorité, issues des normes PSR.

3.1 autoloading

Comme le souligne le PSR-0, pour qui l'objectif est de standardiser le chargement des classes et ainsi éviter les "include" et "require". Il se base sur la bonne utilisation des namespaces. En voici les règles :

- Les classes et les espaces de noms entièrement qualifiés doivent disposer de la structure suivante : \\<Nom du Vendor>\\(<Espace de noms>\\)*<Nom de la Classe>.
- Chaque espace de noms doit avoir un espace de noms racine : ("Nom du Vendor").
- Chaque espace de noms peut avoir autant de sous-espaces de noms qu'il le souhaite.
- Chaque séparateur d'un espace de noms est converti en DIRECTORY_SEPARATOR lors du chargement à partir du système de fichiers.
- Chaque "_" dans le nom d'une CLASSE est converti en DIRECTORY_SEPARATOR. Le caractère "_" n'a pas de signification particulière dans un espace de noms.
- Les classes et espaces de noms complètement qualifiés sont suffixés avec ".php" lors du chargement à partir du système de fichiers.
- Les caractères alphabétiques dans les noms de vendors, espaces de noms et noms de classes peuvent contenir n'importe quelle combinaison de minuscules et de majuscules.

L'utilisation de Composer, qui respecte cette norme, pour la gestion des fichiers d'autoload sera privilégié.

3.2 Les normes de codage de base

Le PSR-1 décrit décrit les éléments standards de codage nécessaires pour assurer un niveau élevé d'interopérabilité technique pour le partage du code PHP. Cette norme concerne : les fichiers, les espaces de nom et noms des classes ainsi que les constantes de classe, propriétés et méthodes. Nous en retiendrons les règles suivantes :

- Les fichiers DOIVENT utiliser seulement les tags < ?php et < ?=.
- Les fichiers de code PHP DOIVENT être encodés uniquement en UTF-8 sans BOM.
- Les fichiers DEVRAIENT soit déclarer des symboles (classes, fonctions, constantes, etc.) soit causer des effets secondaires (par exemple, générer des sorties, modifier paramètres .ini), mais NE DOIVENT PAS faire les deux.
- Les espaces de noms et les classes DOIVENT suivre le point précédent.
- Les noms des classes DOIVENT être déclarés comme StudlyCaps [1].
- Les constantes de classe DOIVENT être déclarées en majuscules avec un sous-tiret en séparateurs.
- Les noms des méthodes DOIVENT être déclarés comme camelCase [2].

1. Chaque mot commence par une majuscule et sont collés les uns aux autres
2. Chaque mot, hormis le premier, commence par une majuscule et sont collés les uns aux autres

3.3 Style d'écriture de code

La norme PSR-2 complète la norme PSR-1 et décrit les bonnes pratiques en matière de programmation. Nous retiendrons les points suivant :

- Le code DOIT suivre les points précédents.
- Le code DOIT utiliser 2 espaces pour l'indentation et aucune tabulation.
- La limite acceptable d'une ligne est de 120 caractères
- Il DOIT y avoir une ligne vide après la déclaration de l'espace de noms, et il DOIT y avoir une ligne vide après le bloc de déclarations use.
- L'ouverture des accolades pour les classes DOIT figurer sur la ligne suivante, les accolades de fermeture DOIVENT figurer sur la ligne suivante après le corps de la classe.
- L'ouverture des accolades pour les méthodes DOIT figurer sur la ligne suivante, les accolades de fermeture DOIVENT figurer sur la ligne suivante après le corps de la méthode.
- La visibilité DOIT être déclarée sur toutes les propriétés et méthodes ; abstract et final doivent être déclarés avant la visibilité, static DOIT être déclaré après la visibilité
- La structure des mots-clés de contrôle DOIT avoir un espace après eux, les méthodes et les appels de fonction NE DOIVENT PAS en avoir.
- L'ouverture des accolades pour les structures de contrôle DOIT figurer sur la même ligne, et la fermeture des accolades DOIT figurer sur la ligne suivante après le corps.
- L'ouverture des parenthèses pour les structures de contrôle NE DOIT PAS contenir d'espace après eux, la fermeture de parenthèses pour les structures de contrôle NE DOIT PAS contenir d'espace avant.

3.4 Interface pour les logs

La norme PSR-3 définit comment seront définit les logs. En voici les concepts de base :

— L'interface LoggerInterface expose huit méthodes pour écrire les logs pour les huit RFC 5424levels (debug, info, notice, warning, error, critical, alert, emergency).

— Une neuvième méthode, log, accepte un niveau de journalisation en tant que premier argument. L'appel de cette méthode avec l'une des constantes du niveau de journalisation DOIT avoir le même résultat que l'appel de la méthode de niveau spécifique. L'appel de cette méthode avec un niveau non défini par cette spécification DOIT lancer un Psr\\Log\\InvalidArgumentException si l'implémentation ne reconnaît pas le niveau. Les utilisateurs NE DEVRAIENT PAS utiliser de niveau personnalisé sans savoir avec certitude si l'implémentation le supporte.

Afin de respecter au plus près cette norme, l'utilisation de monolog est privilégié.

3.5 autoloading avancé

Le PRS-4 complète le PSR-0 en fixant les règles de l'arborescence des fichiers.