Norme de développement

Table des matières

Norme de développement

Il est important d'être cohérent, surtout lorsque l'on écrit du code open-source, car ce code appartient en définitive à des millions de personnes, et que la résolution des bugs repose en grande partie sur ces personnes pour repérer les problèmes et comprendre comment les résoudre.

C'est pourquoi, lorsque vous écrivez du code pour PrestaShop, que ce soit pour un thème, un module ou une modification du coeur du logiciel, vous devez faire en sorte de suivre les indications qui suivent. Elles sont déjà suivies par les développeurs de PrestaShop, et les suivre également est la manière la plus sûre de voir votre code s'intégrer dans PrestaShop de manière élégante.

Pour résumer, la cohérence du code permet de s'assurer que celui-ci est lisible et facile à maintenir.

Si vous utilisez un éditeur, vous pouvez utiliser le validateur CodeSniffer pour vous aider à mieux écrire votre code.

PHP

Noms des variables

Tout comme le nom des classes, des méthodes et des fonctions, le nom des variables doit toujours être écrit en anglais, afin d'être compréhensible par le plus grand nombre.

Utilisez des caractères minuscules, et séparez les mots par des caractères soulignés. N'utilisez jamais le format camelCase.

1. Pour les données en provenance de la base de données : `$my_var`.
2. Pour les algorithmes : `$my_var`.
3. La visibilité d'une variable membre n'affecte pas son nom : `private $my_var`.

Assignations

1. Il doit y avoir une espace entre une variable et l'opérateur d'assignation :

```
$my_var = 17;
$a = $b;
```

Operateurs

1. "+", "-", "*", "/", "=" et toute combinaison de ces opérateurs (ex. : "/=") doivent avoir une espace avant et après.

   ```
   $a + 17;
   $result = $b / 2;
   $i += 34;
   ```

2. "`.`" doit ne pas avoir d'espace avant ou après.

   ```
   echo $a.$b;
   $c = $d.$this->foo();
   ```

   > ⚠️ **Recommandation**
   >
   > Pour des raisons de performance, n'abusez pas des concaténations.

3. "`.=`" doit avoir une espace avant et après.

   ```
   $a .= 'Debug';
   ```

4. Lorsque vous testez une variable booléenne (true/false), n'utilisez pas un opérateur de comparaison, mais testez directement la valeur elle-même, ou la valeur préfixe d'un point d'exclamation :

   ```
   // ne faites pas ceci
   if ($var == true)
   // ...ni ceci
   if ($var == false)

   // faites ceci
   if ($var)
   // ...ou ceci
   if (!$var)
   ```

Déclarations

1. `if`, `elseif`, `while`, `for`: doivent recevoir une espace entre le mot-clé et les parenthèses :

   ```
   if (<condition>)

   while (<condition>)
   ```

2. Lorsque vous utilisez une combinaison de `if` et de `else`, et que chacun renvoie une valeur, le e `lse` peut être omis :

```
if (<condition>)
        return false;
return true;
```

> ⚠ **Recommandation**
>
> Nous vous recommandons de n'utiliser qu'un `return` par méthode/fonction.

3. Lorsqu'une méthode ou une fonction renvoie un booléen (`true`/`false`) et que méthode/fonction en cours dépend de sa valeur, la déclaration if peut être omise :

```
public aFirstMethod()
{
        return $this->aSecondMethod();
}
```

4. Les tests doivent être groupés par entité :

```
if ($price AND !empty($price))
        ...
if (!Validate::$myObject OR $myObject->id === NULL)
        ...
```

Visibilité

1. La visibilité doit être définie à chaque fois, même quand il s'agit d'une méthode publique.
2. L'ordre des propriétés de la méthode doit être ainsi : `visibility static function functionName()`.

```
private static function foo()
```

Noms de méthodes et fonctions

1. Le nom des méthodes et fonctions doivent utiliser la méthode CamelCase : commencer par un caractère en minuscule, chaque mot suivant doit commencer par un caractère en majuscule :

```
public function myExampleMethodWithALotOfWordsInItsName()
```

2. Les accolades qui ouvrent le code d'une méthode doit être précédées d'un retour à la ligne :

```
public function myMethod($arg1, $arg2)
{
        ...
}
```

3. Le nom des méthodes et fonctions doit être explicite ; les noms tels que `b()` ou `ef()` sont donc à proscrire :

> **Exceptions**
>
> Les seules exceptions admises sont la fonction de traduction (nommée `l()`) et les fonctions de débogage (nommées `p()` et `d()`).

Énumération

Les virgules doivent être suivies (et non précédées) d'un espace :

```
protected function myProtectedMethod($arg1, $arg2, $arg3 = null)
```

Objets / Classes

1. Le nom d'un objet doit être au singulier :

   ```
   class Customer
   ```

2. Le nom d'une classe doit suivre la norme CamelCase, sauf pour la première lettre qui doit être en majuscule :

   ```
   class MyBeautifulClass
   ```

Constantes

1. Le nom des constantes doit être écrit en majuscule, sauf pour `true`, `false` et `null` qui doivent être en minuscule : `ENT_NOQUOTE`, `true`.
2. Le nom des constantes doit être préfixé avec "PS_" dans un module ou le coeur de PrestaShop :

   ```
   define('PS_DEBUG', 1);
   define('PS_MODULE_NAME_DEBUG', 1);
   ```

3. Le nom des constantes ne doit utiliser que des caractères alphabétiques, ainsi que le signe "_".

Mots-clés

Tous les mots-clés doivent être en minuscule : `as, case, if, echo, null`.

Variables de configuration

Les variables de configuration doivent suivre les mêmes règles que celles définies ci-dessus.

Chaînes

Une chaîne doit être entourée de guillemets droits simples (`'`), et non de guillemets droits doubles (`"`) :

```
echo 'Debug';
$myObj->name = 'Hello '.$name;
```

Commentaires

1. À l'intérieur d'une fonction et d'une méthode, seul le commentaire de type "`//`" est autorisé.
2. Il doit y avoir un espace à l'ouverture d'un signe de commentaire "`//`" :

```
// My great comment
```

3. Le marqueur de commentaire "`//`" est toléré à la fin d'une ligne de code :

```
$a = 17 + 23; // A comment inside my example function
```

4. En dehors des fonctions et méthodes, seuls les marqueurs "`/*`" and "`*/`" sont autorisés :

```
/* This method is required for compatibility issues */
public function foo()
{
        // Some code explanation right here
        ...
}
```

5. Un commentaire phpDoc est requis en ouverture de méthode :

```
/**
 * Return field value if possible (both classical and multilingual fields)
 *
 * Case 1: Return value if present in $_POST / $_GET
 * Case 2: Return object value
 *
 * @param object $obj Object
 * @param string $key Field name
 * @param integer $id_lang Language id (optional)
 * @return string
 */
protected function getFieldValue($obj, $key, $id_lang = NULL)
```

> **À propos de phpDoc**
>
> Pour plus d'information sur la syntaxe phpDoc : http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_tags.pkg.html.

Valeurs retournées

1. La déclaration `return` ne nécessite pas de parenthèses, sauf dans le cas d'une expression composée :

```
return $result;
return ($a + $b);
return (a() - b());
return true;
```

2. La déclaration `return` peut être utilisée pour sortir d'une méthode :

```
return;
```

Appels

Il est interdit d'appeler une fonction précédée d'un "@", mais faites attention aux appels de fonctions /méthodes qui ont un identifiant/mot de passe ou un chemin dans leurs arguments :

```
myfunction();

// Dans l'exemple suivant, nous utilisons @ pour des raisons de sécurité
@mysql_connect(...);
```

Balises

1. Il doit y avoir une ligne vide après la balise d'ouverture de PHP :

```
<?php

require_once('my_file.inc.php');
```

2. Il est interdit d'utiliser la balise de fermeture de PHP en fin de fichier.

Indentation

1. "`\t`" est le seul caractère d'indentation autorisé.
2. Chaque niveau d'indentation doit utiliser un seul caractère d'indentation :

```
function foo($a)
{
        if ($a == null)
                return false;
        ...
}
```

Tableaux

1. Le mot-clé `array` ne doit pas être suivi d'un espace :

```
array(17, 23, 42);
```

2. Lorsqu'il y a trop de données dans un tableau, l'indentation doit être comme suit :

```
$a = array(
        36 => $b,
        $c => 'foo',
        $d => array(17, 23, 42),
        $e => array(
                0 => 'zero',
                1 => $one
        )
);
```

Blocs

Les accolades sont à éviter lorsqu'elles n'encadrent qu'une seule instruction ou une combinaison de déclarations :

```
if (!$result)
    return false;

for ($i = 0; $i < 17; $i++)
    if ($myArray[$i] == $value)
    {
        $result[] = $myArray[$i];
        return $result;
    }
    else
        $failed++;
```

Sécurité

1. Toutes les données en provenance de l'utilisateur doivent être typées :

```
$data = Tools::getValue('name');

$myObject->street_number = (int)Tools::getValue('street_number');
```

2. Tous les paramètres de méthodes/fonctions doivent être typés (avec `Array` ou `Object`) dès récupération :

```
public myMethod(Array $var1, $var2, Object $var3)
```

3. Tous les autres paramètres doivent être typés à chaque utilisation, sauf quand ils sont envoyés à d'autres méthodes/fonctions :

```
protected myProtectedMethod($id, $text, $price)
{
        $this->id = (int)$id;
        $this->price = (float)$price;
        $this->callMethod($id, $price);
}
```

Limitations

1. Chaque ligne du code source doit s'arrêter à 150 caractères.
2. Les lignes de fonctions/méthodes doivent s'arrêter à 80 caractères. Une fonction doit avoir une bonne raison d'avoir un long nom : ne gardez que l'essentiel !

Autres

1. Il est interdit d'utiliser un opérateur ternaire dans un autre opérateur ternaire, comme `echo ((true ? 'true' : false) ? 't' : 'f');`.
2. Nous vous recommandons de préférer `&&` et `||` dans vos conditions : `echo ('X' == 0 && 'X' == true)`.
3. Évitez d'utiliser des paramètres avec référence, comme ceci :

```
function is_ref_to(&$a, &$b) { ... }
```

SQL

Noms de table

1. Les noms de table doivent commencer avec le préfixe de PrestaShop, à l'aide de "`_DB_PREFIX_`" :

```
... FROM `'. _DB_PREFIX_.'customer` ...
```

2. Les noms de table doivent avoir le même nom que l'objet qu'elles représentent : "`ps_cart`".
3. Les noms de table doivent être au singulier : "`ps_order`".
4. Les données de langue doivent être stockées dans une table nommée exactement de la même manière que la table de l'objet, avec le suffixe "`_lang`" : "`ps_product_lang`".

Requêtes SQL

1. Les mots-clés doivent être écrits en majuscule :

```
SELECT `firstname`
FROM `'._DB_PREFIX_.'customer`
```

2. Le caractère accent grave ("`") doit encadrer les noms des champs SQL et des tables SQL :

```
SELECT p.`foo`, c.`bar`
FROM `'._DB_PREFIX_.'product` p, `'._DB_PREFIX_.'customer` c
```

3. Les alias de table doivent être nommés en prenant la première lettre de chaque mot, le tout en minuscule :

```
SELECT p.`id_product`, pl.`name`
FROM `'._DB_PREFIX_.'product` p
NATURAL JOIN `'._DB_PREFIX_.'product_lang` pl
```

4. Lorsque d'un conflit survient entre deux alias de tables, le second caractère doit également être ajouté dans le nom :

```
SELECT ca.`id_product`, cu.`firstname`
FROM `'._DB_PREFIX_.'cart` ca, `'._DB_PREFIX_.'customer` cu
```

5. L'indentation doit être faite pour chaque clause :

```
$query = 'SELECT pl.`name`
FROM `'._DB_PREFIX_.'product_lang` pl
WHERE pl.`id_product` = 17';
```

6. Il est interdit de faire un `JOIN` dans une clause `WHERE`.

Installation du validateur de code (PHP CodeSniffer)

Voici un bref tutoriel pour installer la moulinette de norme sur son PC et l'utiliser pour valider ses fichiers. La moulinette de norme passe par PHP CodeSniffer qui est un package de PEAR (http://pear.php.net/package/PHP_CodeSniffer/). La norme PrestasShop a été créée pour l'occasion, constituée de nombreuses règles reprises des normes déjà existantes, plus un certain nombre de règles personnalisées pour coller davantage au projet.

La norme PrestaShop est disponible via Git ici : https://github.com/PrestaShop/PrestaShop-norm-validator (cette étape est obligatoire pour la suite).

Afin qu'elle soit reconnue en tant que norme de base, il faut la placer dans le dossier `/Standards` de CodeSniffer.

Intégration à PhpStorm

Suivez ces étapes si vous utilisez PhpStorm (http://www.jetbrains.com/phpstorm/) :

1. Aller dans Settings -> Inspection -> PHP -> PHP Code Sniffer ;
2. Choisir le chemin vers l'exécutable `phpcs` ;
3. Choisir le coding standard "PrestaShop" (disponible uniquement si vous l'avez bien placé dans le dossier `/Standards` de CodeSniffer).

Intégration à vim

Plusieurs plugins existent sur le net. Par exemple, vous pouvez utiliser celui-ci : https://github.com/bpearson/vim-phpcs/blob/master/plugin/phpcs.vim
Placez-le dans votre dossier `~/.vim/plugin` .

Vous pouvez rajouter deux raccourcis (par exemple, F9 pour tout afficher et Ctrl+F9 pour masquer les warnings) dans votre fichier `.vimrc` en mode normal et insertion:

```
nmap <C-F9>:CodeSniffErrorOnly<CR>
imap <C-F9> <Esc>:CodeSniffErrorOnly<CR>
nmap <F9>                       :CodeSniff<CR>
imap <F9> <Esc>:CodeSniff<CR>a
```

Ligne de commande (Linux)

Vous n'êtes pas obligé d'utiliser PhpStorm pour profiter de la norme. Vous pouvez également installer PHP CodeSniffer afin de l'appeler en ligne de commande :

1. Installez PEAR : http://pear.php.net/
   `$> apt-get install php-pear`
2. Installez PHP CodeSniffer dans PEAR : http://pear.php.net/package/PHP_CodeSniffer
   `$> pear install PHP_CodeSniffer`
3. Ajoutez la norme PrestaShop que vous avez téléchargez du SVN, et placez-la dans le dossier `/Standards` de CodeSniffer
   `$> svn co http://svn.prestashop.com/branches/norm/ /usr/share/php/PHP/CodeSniffer/Standards/Prestashop`
4. Configurez Prestashop comme étant la norme de base
   `$> phpcs --config-set default_standard Prestashop`

Les différentes options de la commande sont disponibles et bien expliquées dans la doc, voici cependant une façon simple de le lancer :

```
$> phpcs --standard=/chemin/vers/norme/Prestashop /folder/or/fileToCheck
```

Afin de n'afficher que les erreurs et non pas les warnings :

```
$> phpcs --standard=/chemin/vers/norme/Prestashop --warning-severity=99 /chemin/ou/fichierAVerifier
```

Si vous avez installé PHP CodeSniffer "à la main", l'exécutable se trouve dans le dossier `/scripts` de PEAR.

⚠ Pour les utilisateurs Windows, un fichier `phpcs.bat` est disponible dans le dossier `/scripts` de PEAR, il est possible qu'il faille le modifier pour qu'il marche bien, voici ce qu'il doit contenir (remplacer les chemins par les vôtres) :

```
chemin/vers/php.exe -d auto_apprend_file="" -d auto_prepend_file -d include_path="chemin/vers/PEAR/"
chemin/vers/pear/scripts/phpcs
```