

|  |
| --- |
| SQLI  **WebSolution** |
|  |
| Standard de codage PHP |
| Version 1.0 – 15/10/2018 |

Table des matières

[I. Introduction 4](#_Toc529702931)

[II. Recommandation d’outl de dèveloppement : 5](#_Toc529702932)

[III. Convention de codage 6](#_Toc529702933)

[3.1. Conventions de nommage 6](#_Toc529702934)

[3.1.1. Les fonctions et les variables – (LowerCamelCase) 6](#_Toc529702935)

[3.1.2. Les Constants – ALL\_CAPS 6](#_Toc529702936)

[3.1.3. Les classes et les interface – UpperCamelCase 6](#_Toc529702937)

[3.2. Utilisation des espaces 7](#_Toc529702938)

[3.2.1. Les lignes vides 7](#_Toc529702939)

[3.2.2. Les espaces 7](#_Toc529702940)

[3.2.3. L’indentation 8](#_Toc529702941)

[3.2.4. Les lignes de continuation 8](#_Toc529702942)

[3.3. Les Commentaires 9](#_Toc529702943)

[3.4. Les instructions 10](#_Toc529702944)

[3.4.1. Les Instructions simples 10](#_Toc529702945)

[3.4.2. Affectations et expressions 10](#_Toc529702946)

[3.4.3. L’instruction « if » 10](#_Toc529702947)

[3.4.4. L’instruction « switch/case » 11](#_Toc529702948)

[3.4.5. L’instruction « for » : 11](#_Toc529702949)

[3.4.6. L’instruction « foreach » : 11](#_Toc529702950)

[3.5. Les Guillemets 12](#_Toc529702951)

[3.6. Balises courtes d'ouverture 12](#_Toc529702952)

[3.7. E\_ALL 12](#_Toc529702953)

Historique des révisions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Vers. | Date | Modifications | Auteur | Date de validation | Approbateur |
| 1.0 | 15/10/2018 |  | A. VATIN |  |  |
|  |  |  |  |  |  |

# Introduction

Ce document contient l’ensemble des standards de codage et les bons pratiques à appliquer lors du développement des projets PHP au sein de l’équipe Web solution.

Toutes scripte qui ne respecte pas ces normes est susceptible d’être refusé par l’expert technique même s’il répond au besoin demandée.

Ces règles sont mises en place afin de garantir :

* La qualité et la lisibilité de code.
* L’efficacité et l’optimisation du code.
* l’organisation et la documentation des développements.
* Faciliter la maintenance et l’évolution des applications.
* Faciliter le transfert/reprise entre développeurs.

# Recommandation d’outl de dèveloppement :

Lors de développement d’une application ou d’un nouveau site il faut s’assurer de travailler avec les outils suivants :

**Version PHP :** PHP 7 ou supérieur

**Serveur apache :** Apache 2.2.4 ou supérieur

**Outil de versioning :** GIT

**Outil de développement :** PHP Storm avec le plugin « file Editor » installer

**Serveur emailing :** Exim4

**Gestion de dépendance :** Composer 1.6 est plus

**Client FTP et SSH :** WinScp et putty en tant que client SSH

**NB : Les bases de données et les éditeurs doivent être configurer en UTF8**

# Convention de codage

## Conventions de nommage

### Les fonctions et les variables – (LowerCamelCase)

Le style retenu pour le nommage des fonctions et variables est: « Camel Case »

Le « Camel Case » est la pratique qui vise à utiliser une majuscule comme séparateur de mots. Le 1er mot commence toujours par une minuscule.

Eviter les abréviations au maximum, qui nuisent à la rapidité de compréhension du code.

// Correcte  
getListeUtilisateur();  
updateVehicule();  
creerCommande();  
  
// Incorrecte  
InsererEnr(); // Démarre avec une majuscule, nom pas explicite.  
ajouter\_ligne\_commande(); // Utilisation des under-score.  
  
// Correcte  
$unUtilisateur;  
$booleen;  
  
// Incorrecte  
$fab; // Mot pas explicite, éviter les abréviations.  
$codecommande; // 2ème mot ne commence pas par une majuscule.

### Les Constants – ALL\_CAPS

Les constantes sont toujours exclusivement en majuscule même si celles-ci sont composées de plusieurs mots. Les under-score peuvent être utilisés séparateur de mot si besoin est. Les valeurs booléennes« false » et « true » sont toujours en minuscule.

define("CURRENT\_YEAR", date('Y'));

### Les classes et les interfaces – UpperCamelCase

Les noms de classe doivent être écrits de la même façon que les fonctions (en camelCase) mais en commençant par une majuscule.

// Interface  
**interface** Transport  
{  
 **public function** getDescription();  
}  
  
// Class  
**class** ModelsCar **implements** Transport  
{  
 // Instructions  
}

Il faut se limiter à une classe par fichier.

## Utilisation des espaces

### Les lignes vides

Les lignes vides peuvent améliorer la lisibilité par le regroupement des sections du code qui ont des liens logiques. Une ligne vide doit également être utilisée dans les emplacements suivants :

* + Après le commentaire du copyright (début de fichier)
  + Entre des déclarations de classes.
  + Entre des déclarations de fonctions.

### Les espaces

Un espace ne doit pas être utilisé :

* 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 après la fermeture d’une accolade carré (« \* » ou « ] »).
* Avant chaque « , ».

Un espace doit être utilisé :

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

// Correcte  
**function** getListeUtilisateur($idUtilisateur, $typeUtilisateur)  
$type = $resultat[*‘type’*];  
**if** ($admin == *ADMINISTRATEUR* || $verifier != **false**);  
// Incorrecte  
**function** getListeUtilisateur ( $idUtilisateur , $typeUtilisateur )  
$type=$resultat[ *‘type’* ] ;  
**if**($admin==*ADMINISTRATEUR*||$verifier!=**false**) ;

### L’indentation

4 espaces doivent être utilisés pour tout niveau d’indentation : le premier niveau est indenté de 4 espaces, le deuxième de 8 espaces, le troisième de 12 espaces, etc. Afin de pouvoir utiliser la touche tabulation il faut définir dans l’IDE utilisé qu’une tabulation correspond à 4 espaces.

### Les lignes de continuation

Les lignes doivent être limitées à 85 caractères (espaces compris). Les lignes d’une longueur supérieure à 85 caractères doivent être découpées selon le besoin en une ou plusieurs lignes de continuation. Toutes les lignes de continuation doivent être alignées et indentées de 4 espaces à partir de la première ligne de l’expression.

// Correcte  
**function** creerUtilisateur(  
 $nom, $prenom, $age, $secu, $cb, $ville, $cp, $adresse,  
 $tel, $fax, $email  
)  
// Incorrecte  
**function** creerUtilisateur($nom, $prenom, $age, $secu,  
 $cb, $ville, $cp, $adresse, $tel, $fax, $email  
)

Dans une fonction ou méthode, le découpage de la 1ère ligne doit se faire juste après la parenthèse ouvrante.

// Correcte  
**if** ($x == $y  
 || $x == $z  
 || ($x == $a && $a == $b)  
) {  
 // Instructions  
}  
// Incorrecte  
**if** ($x == $y || $x  
 == $z || ($x == $a && $a == $b)  
) {  
 // Instructions  
}

Dans une condition, le découpage de la 1ère ligne doit se faire avant un opérateur logique (||, &&) et non un opérateur de comparaison (==, !=).

## Les Commentaires

Le langage PHP permet deux types de commentaires : des commentaires de bloc ( /\* \*/ ), et des commentaires de ligne ( //). De manière générale il faut placer des commentaires de type bloc : - en début de fichier pour expliquer leur but.

* Avant les déclarations de classes.
* Avant les déclarations de fonctions.

Eviter de faire des commentaires « évidents » du genre :

$i = $i + 1; // Incrémenter i

Ne pas oublier que les commentaires flous, trompeurs et/ou inexacts sont pires que l’absence de commentaire, lors de la création d’un commentaire il faut :

* Eviter de mettre dans les commentaires des informations qui risquent de devenir obsolètes.
* Eviter d’entourer les commentaires dans les boîtes dessinées aves des caractères spécial. Les commentaires de type Perl/shell « # » sont à éviter.

Commentaire d’en-tête de fichier :

*/\*\*  
 \* Valide les commentaires saisis sur les terminaux et les  
 \* dates de livraisons rapatriées  
 \*  
 \* PHP Version 5.3  
 \*  
 \** ***@category*** *Void  
 \** ***@package*** *Void  
 \** ***@author*** *EB <****emmanuel@e-baudoin.fr****>  
 \** ***@license*** *E.Baudoin  
 \** ***@link http://www.e-baudoin.fr*** *\*  
 \*/*

Commentaire d’en-tête de fonction :

*/\*\*  
 \* Retourne si oui ou non un fabricant est rattaché à un transporteur.  
 \*  
 \** ***@param*** *Connection $db  
 \** ***@param*** *int $idTransporteur  
 \** ***@param*** *int $idFabricant  
 \*  
 \** ***@return*** *Boolean  
 \*/***function** isTransporteur($idTransporteur)  
{  
 // Instructions  
}

## Les instructions

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

### Les Instructions simples

Une instruction simple contient un seul point-virgule. Si vous devez écrire l’instruction sur plusieurs lignes, la deuxième ligne doit avoir une indentation de 4 espaces par rapport à la première ligne.

$resultatSoumission = '<span class="messageConfirmation">Les données ont  
 bien été envoyées</span>';

### Affectations et expressions

Chaque ligne doit contenir au maximum une instruction

$numeroLigne = 0; $cpt = 0; // Incorrecte  
$numeroLigne = 0; // Correcte

### L’instruction « if »

Les instructions « if » doivent toujours apparaître sur au moins deux lignes. Si le « if » ne comporte qu’une seule instruction, la présence des accolades est malgré tout nécessaire. Leur présence augmente la lisibilité du code et réduit le risque d'erreur logique lors de l'ajout de nouvelles lignes de code.

// Incorrecte  
**if** (!is\_dir($unRepertoire)) mkdir($unRepertoire, 0777);  
// Correcte  
**if** (!is\_dir($unRepertoire)) {  
 mkdir($unRepertoire, 0777);  
}

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

// Incorrecte  
**if** ($idEntite == *‘’*)  
{  
 $idEntite = 1;  
}  
**else**{  
 $idEntite++;  
}  
// Correcte  
**if** ($idEntite == *‘’*) {  
 $idEntite = 1;  
}  
**else** {  
 $idEntite++;  
}

### L’instruction « switch/case »

Ce type d’instruction est à éviter car elle est plus coûteuse en termes de ressources qu’un « elseif ».

### L’instruction « for » :

Eviter les répétitions de calculs inutiles et coûteux du genre :

**for** ($i = 0; $i < count($tableau); $i++) {  
 //instructions  
}

car count($tableau) sera calculé n fois. Passer par une variable intermédiaire et faire ce calcul en dehors de la boucle.

// Incorrecte  
**for** ($i=0;$i<$max;$i++) {  
 //instructions  
}  
// Correcte  
**for** ($i = 0; $i < $max; $i++) {  
 // Instructions  
}

### L’instruction « foreach » :

Une boucle « foreach » est vivement conseillée à un « while » dès lors qu’un tableau associatif est utilisé dans la boucle, Les gains de performances sont conséquents. Par contre avec un simple tableau il est fortement déconseillé d’utiliser un « foreach ». Il faut préférer un « while » ou un « for » pour des performances 10 fois supérieures.

**foreach** ($tableau **as** $i => $uneLigne) {  
// Instructions   
}

## Les Guillemets

L'utilisation des guillemets simples est à préférer aux guillemets doubles dans les scripts PHP et JavaScript. Les guillemets doubles doivent être réservés pour les requêtes SQL et le code HTML. Le but est d’optimiser l’interprétation du code par le serveur.

## Balises courtes d'ouverture

L'utilisation des balises courtes d'ouverture **suivant est fortement** déconseillée.

<% %>

<%= %>

<script language="php"></script?>

Par contre les seules balises d’ouverture autorisées sont :

<?php ?>

<?= ?>

## E\_ALL

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