RÈGLES DE CODAGE

CHRISTIAN BONHOMME

23/09/2012

INTRODUCTION

Ce document présente les différentes règles de codage pour les applications Intranet ou WEB.
Une application WEB se compose de 5 types de programmation :

* **HTML** (**H**ypertext **M**arkup **L**anguage**)** pour la mise en forme du contenu des pages html lues par le navigateur (IE, Firefox, …)
* **CSS** (**C**ascading **S**tyle **S**heet) pour la présentation et la charte graphique des pages html
* **Javascript** pour les programmes du côté navigateur (client) qui utilise le **DOM** (**D**ocument **O**bject **M**odel) pour la modification du document WEB.
* **PHP** (**P**ersonal **H**ypertext **P**reprocessor) pour les programmes du côté serveur
* **SQL** (**S**tructured **Q**uery **L**anguage) pour l'interrogation et la manipulation des données d'une base de données.

Ce document décrit les règles de codage pour chaque type de programmation en respectant les règles édictées par le **W3C** (**W**ord **W**ide **W**eb **C**onsortium) organisme rédigeant les recommandations applicables au **HTML**, au **CSS** et au **DOM**.

RÈGLES GÉNÉRALES

Le nom des constantes, des variables, des paramètres, des fonctions et des classes, doit être écrit , si possible, en anglais et être explicite.
Les programmes doivent être le plus possibles documentés.
On adoptera pour les commentaires la syntaxe **PHPDoc** afin d'obtenir une documentation automatique.
Les règles de codages suivantes s'appliquent au **PHP** et au **Javascript**.

PRÉSENTATION DU CODE

Les fonctions et les méthodes doivent être écrites sans blanc entre le nom et la première parenthèse.

```
function modify()
```

Les fonctions, les méthodes, les classes, les boucles et les structures conditionnelles doivent être écrite de la façon suivante :

* l'accolade **{** apparaît au début de la ligne suivante,
* l'accolade **}** apparaît seule sur la dernière ligne ou suivie d'un commentaire.

```
if ($title != 'Titre')
{
  contenu du if
}
```

L'indentation entre chaque bloc doit être de 2 caractères.
En cas de comparaison entre une variable et une constante, la variable doit être mise à droite :

```
if ('Titre' == $title)
```

Ce qui évite des erreurs du type `if ($title = 'Titre')` qui est une affectation et ne génère pas d'erreur et renvoie toujours vraie.

Les boucles **`for`** doivent être écrites de la façon suivante :

```
for ($i = 0; $i < $nb; ++$i)
```

Un blanc après chaque point-virgule, un blanc avant et après l'opérateur de comparaison, utilisé la pré-incrémentation plutôt que la post-incrémentation.

Les boucles **`for`**, les structures conditionnelles **`if`** et **`else`** étant suivies d'une seule ligne doivent être encadrées par des accolades.

```
for ($i = 0; $i < $nb; ++$i)
{
  une ligne;
}

if ($i)
{
  une ligne;
}
else
{
  une autre ligne;
}
```

Remarque :
*Il est impératif de respecter cette règle pour le javascript (**risque d'erreurs avec IE**).*

Les opérateurs ternaires doivent être écrits de la façon suivante :

```
$a = (operation de comparaison) ? $val1 : $val2;
```

Des parenthèses doivent encadrées l'opération de comparaison.

Remarque :
*L'opérateur ternaire doit être utilisé plutôt que le **`if -- else --`** sauf cas particulier.*

INITIALISATION

Initialiser les variables là où elles sont déclarées.
Ne pas initialiser plusieurs variables à une même valeur dans un seul énoncé :
Ne pas écrire :

```
$a = $b = 4;
```

Mais plutôt :

```
$a = 4;
$b = 4;
```

Éviter les initialisations imbriquées :

```
$a = ($b = $c + $d) + $e;
```

et remplacer par :

```
$b = $c + $d;
$a = $b + $e;
```

Éviter de mettre des nombres dans le code excepté 0 et 1. Définir plutôt des constantes.
Ne pas écrire :

```
for ($i = 0; $i < 1000 ; ++$i)
```

mais plutôt :

```
define('LONG_MAX', 1000);
for ($i = 0; $i < LONG_MAX ; ++$i)
```

Pour éviter les problèmes de précédence des opérateurs, il est recommandé d'utiliser les parenthèses dans des expressions impliquant des opérateurs mixtes. Il n'est pas évident que le programmeur connaisse les règles de précédence aussi bien que le développeur original.

Ne pas écrire :

```
if ($a == $b && $c ==$ d)
```

mais plutôt :

```
if (($a == $b) && ($c == $d))
```

Pour tester la différence, il vaut mieux utiliser le != que le <>
Ne pas écrire :

```
if ($a <> 'val')
```

mais plutôt :

```
if ($a != 'val')
```

Si possible, il est préférable d'utiliser la pré-incrémentation à la post-incrémentation.

RÈGLES HTML

Les balises doivent être conformes à la norme **XHTML5** du **W3C**.

Les balises doivent être écrites en minuscules et les valeurs des attributs doivent être entre des guillemets double (").

Le nom des paramètres intervenant dans les URL ou passer dans les formulaires doit être en majuscule.

```
url?EXEC=home&amp;TITLE=Titre
```

```
<input name="TITLE" value="" size="40"/>
```

Les pages html générées doivent être validées avec le validateur **W3C** (http://validator.w3.org/).
Le codage **HTML** doit s'attacher au contenu et au sens plutôt qu'à la présentation (règle du balisage sémantique).
Par exemple les formulaires ne doivent plus être écrits avec des tableaux, mais de la façon suivante :

```
<form action="url_form" method="post">
  <legend>Titre</legend>
  <fieldset>
    <p>
      <label for="name">Nom :</label>
        <input id="name" name="NAME" value="" size="20" />
    </p>
    <p><input type="submit" name="SUBMIT" value="Ok" /></p>
  </fieldset>
</form>
```

De même les menus ne doivent plus être écrits avec des tableaux, mais sous forme de listes ordonnées :

```
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
</ol>
```

L'utilisation des microformats est recommandée (règle du web sémantique) :

```
<div class="vcard">
  <p class="fn">Christian Bonhomme</p>
  <p class="org">IUT</p>
  <p class="tel">5260</p>
       <a class="mail" href="mailto:cb@iut.fr">cb@iut.fr</a>
</div>
```

Le nom formaté (*fn*), l'organisation (*org*), le numéro de téléphone (*tel*) et l'adresse email (*mail*) ont été identifiés en utilisant des noms de classes spécifiques ; et l'ensemble est contenu dans une **`class="vcard"`** (Visit Card), le tout forme un microformat appelé **hCard** (raccourci de "HTML vCard").
Les navigateurs de nouvelle génération pourront extraire l'information de ces microformats et la transférer vers d'autres applications, comme pour notre exemple un carnet d'adresses
.
Dans les formulaires les balises **input** doivent être précédées par la balise **<label>** avec l'attribut **for** afin de respecter les règles d'accessibilité pour les mal voyants.

```
<label for="name">Nom :</label>
<input id="name" name="NAME" value="" size="20" />
```

Dans les formulaires les boutons **Submit** de type image doivent utiliser la syntaxe recommandée suivante :

```
<input type="image" src="save.png" alt="Save" title="Save"
value="Save" />
```

RÈGLES CSS

Le codage **CSS** s'écrit de la façon suivante :

```
balise1
{
propriété1: valeur1;
propriété2: valeur2;
}

balise2
{
propriété1: valeur1;
propriété2: valeur2;
}
```

Règle :

* Après une balise, on change de ligne et on ouvre une accolade **{** .
* Après l'accolade on met la propriété (ou les propriétés séparées par une virgule), sans blanc ou tabulation.
* La propriété est suivie d'un **:** (sans blanc).
* Le **:** est suivi d'un blanc et de la valeur de la propriété (ou des valeurs séparées par un blanc, ou une vigule et un blanc suivant les cas).
* La dernière valeur se termine par **;** (sans blanc).
* Les lignes suivantes respectent les mêmes règles.
* A la fin de la dernière propriété, on passe à la ligne et on ferme l'accolade **}**.
* La balise suivante est écrite en passant une ligne.

Les pages CSS doivent être validées avec le validateur **W3C** (**http://jigsaw.w3.org/css-validator/**).

RÈGLES JAVASCRIPT

Les variables locales (variables implicites) - dans un bloc d'accolades - doivent être déclarées à l'aide du mot-clé var, même les variables de boucles, sinon ces variables deviennent des variables globales (variables explicites) ce qui peut à terme générer des erreurs difficiles à détecter :

```
var somme;
var produit = 0;
for (var i = 0; i < nb; ++i)
```

Les scripts javascripts doivent être le plus possibles appelés par des fichiers externes.

```
<script src="script.js"></script>
```

Si un script doit être local il doit être entouré par le commentaire html `<!--//  //-->` :

```
<script type= "text/javascript ">
<!--//
script javascript
//-->
</script>
```

Règles PHP

Convention d'écritures

Constantes

Le nom des constantes doit être écrit en majuscule.

```
define('LOGIN', 'login');
```

S'il est la juxtaposition de plusieurs mots, tous les mots à partir du second doivent être séparés par un underscore '_'.

```
define('CSS_PAGE', '../Css/page.css');
```

Variables

Le nom des variables doit être écrit en minuscule.

```
$title = 'Titre';
```

S'il est la juxtaposition de plusieurs mots, tous les mots à partir du second doivent être séparés par un underscore '_'.

```
$list_product = new array();
```

Si la variable est assignée à un paramètre son nom doit être écrit en majuscule, ce nom doit être identique à la clef du tableau associatif.

```
$TITLE = $_REQUEST['TITLE'];
```

Chaînes

Pour l'assignation des variables de type chaînes de caractères ne contenant pas de variables, la chaîne doit être encadrée par un guillemet simple (') plutôt qu'un guillemet double (").
Ne pas écrire :

```
$title = "Titre";
```

mais plutôt :

```
$title = 'Titre';
```

Tableaux

Les clefs dans les tableaux associatifs doivent être encadrées par des guillemets simples (')

```
$tableau['key']
```

pseudo fonction echo

Les règles de codage pour la pseudo fonction **`echo`** sont :

Simple : pas de guillemet

```
$a = 'Bonjour';
echo $a; // affiche Bonjour
```

Plusieurs variables : guillemet simple pour les chaînes et virgule entre les chaines et les variables

```
$a = 'Christian';
$b = 'Bonhomme';
echo 'Nom : ', $a, ' – Prénom : ', $b; // affiche Nom :
Bonhomme – Prénom : Christian
```

HTML avec une simple variable : pas de guillemet et utilisation des short tags (<?= $val ?>)

```
<h1><?= $a ?></h1>
```

HTML avec plusieurs variables : utilisation du "here document" (`echo <<<HERE---HERE;`)

```
echo <<<HERE
  <div>
    <p>Nom : $a</p>
    <p>Prénom : $b</p>
    <p>Tableau : {$tableau['key']}</p>
    Texte HTML avec d'autres variables
    ...
   </div>

HERE;
```

règle :

* ***`HERE;`*** *commence en première colonne*
* *après* ***`echo <<< HERE`*** *et* ***`HERE;`*** *pas de caractère, ni de blanc*
* *avant* ***`HERE;`*** *une ligne blanche (retour chariot)*
* *si une variable est un tableau associatif ou scalaire multiple, encadrer la variable avec des accolades (forçage de valuation :* ***Tableau : {$tableau['key']}*** *).*

Fonctions

Le nom des fonctions doit être écrit en minuscule.

```
function modify()
```

S'il est la juxtaposition de plusieurs mots, tous les mots à partir du second doivent être séparés par un underscore '_'.

```
function form_insert()
```

L'accolade de fin de fonction doit être suivie d'un commentaire redonnant le nom de la fonction. L'accolade de fin de bloc doit être précédée par une ligne blanche.

```
function modify()
{
  contenu de la fonction

} // modify()
```

Les fonctions doivent être précédées par un commentaire de type **Doxygen []** explicitant le rôle de la fonction et de son action, et décrivant les arguments et la variable de retour.

```
/**
* \fn modify ($type)
* \brief Fonction de modification de la base de
données
*
* \param $type : Type de la modification (insert,
update, delete)
*/
```

Classes

Le nom des classes doit être écrit en minuscule sauf les deux premières lettres qui doivent être en majuscule.

Si elles sont la juxtaposition de plusieurs mots, tous les mots à partir du second doivent commencer par une majuscule.

Si la classe est une classe en dehors de la méthode **MVC** (**M**odel **V**iew **C**ontroler), c'est à dire de type fonctionnel, la première lettre de la classe sera un **C**.

```
class CPolygone

class CElementFini
```

Si la classe appartient à la méthode **MVC**, le nom explicite sera précédé par :

* un **V** pour les classes de types **View** : `class VHtml`
* un **M** pour les classes de types **Model** : `class MImpuretes`

De plus si on utilise la technique de l'**autoload** le nom des fichiers de classes seront identiques au nom de la classe suivi de l'extension correspondante et suivi de **php** :

* **class** pour les classes fonctionelles : **`class CPolygone.class.php`**
* **view** pour les classes de types **View** : **`class VHtml.view.php`**
* **mod** pour les classes de types **Model** : **`class MImpuretes.mod.php`**

Le nom des méthodes de classe doit être écrit en minuscule avec la première lettre en majuscule.

```
public function Select()
```

S'il est la juxtaposition de plusieurs mots, tous les mots à partir du second doivent être séparés par un underscore '_'.

```
public function Select_all()
```

Les arguments des méthodes de classes seront précédés par un underscore '_'.

```
public function Set_value($_value)
```

Les arguments des méthodes doivent respecter les règles sur les variables.

L'accolade de fin de méthode doit être suivie d'un commentaire redonnant le nom de la méthode.
L'accolade de fin de bloc doit être précédée par une ligne blanche.

```
public function Set_value($_value)
{
  contenu de la fonction

} // Set_value()
```

L'accolade de fin de classe doit être suivie d'un commentaire redonnant le nom de la classe.
L'accolade de fin de bloc doit être précédée par une ligne blanche.

```
class VHtml
{
  contenu de la class

} // VHtml
```

Les classes doivent être précédées par un commentaire de type **Doxygen []** explicitant le rôle de la classe et de son action.
Les membres doivent être décrits.
Les méthodes de classe doivent être précédées par un commentaire explicitant le rôle de la méthode, de son action, et décrivant les arguments et la variable de retour.

```
/*! \class VHtml
* \brief classe affichant un fichier Html
*
*/
class $VHtml
{
  public function __construct(){}

  public function __destruct(){}

  /*!
  * \brief affichage du fichier $_html
  *
  * \param : $_html fichier html a afficher
  */
  public function View_html($_html)
  {
    (file_exists ($_html)) ? require($_html) ? require("../Html/$_html");

  } // view_html()

}; // VHtml
```

RÈGLES SQL

Les attributs et les tables doivent être écrits en majuscule.
Les requêtes doivent être indentées de la façon suivante :

select

```
select ATTRIBUT1,
            ATTRIBUT2
from NOM_TABLE1,
          NOM_TABLE2
where ATTRIBUT3 = $ATTRIBUT3
and ATTRIBUT4 = $ATTRIBUT4
```

règle : *alignement des attributs et des tables*

insert

```
insert into NOM_TABLE
(ATTRIBUT1,
 ATTRIBUT2
)
values
($ATTRIBUT1,
 $ATTRIBUT2
)
```

règle : *alignement des attributs et des valeurs d'attributs*

update

```
update NOM_TABLE
set ATTRIBUT1 = $ATTRIBUT1,
       ATTRIBUT2 = $ATTRIBUT2
where ATTRIBUT3 = $ATTRIBUT3
and ATTRIBUT4 = $ATTRIBUT4
```

règle : *alignement des attributs*

delete

```
delete from NOM_TABLE
where ATTRIBUT3 = $ATTRIBUT3
and ATTRIBUT = $ATTRIBUT4
```

Les requêtes SQL doivent être explicites :
Ne pas écrire :

```
select *
from NOM_TABLE
```

mais plutôt :

```
select ATTRIBUT1,
            ATTRIBUT2
from NOM_TABLE
```

De même, ne pas écrire :

```
insert into NOM_TABLE
values
('val1',
 'val2'
)
```

mais :

```
insert into NOM_TABLE
(ATTRIBUT1,
 ATTRIBUT2
)
values
($val1,
 $val2
)
```