Coding Standard

De Boreal-IS.

Sommaire

1 Objectifs

Le présent document vise à définir des standards sur les normes de documentation logicielle à respecter dans le cadre du développement des applications de Boreal-IS. Ce document devra être respecté par tous les membres. Il est de la responsabilité de tous les membres de s'assurer de la qualité de son contenu et de sa conformité avec les objectifs de l'équipe. Il est aussi possible que des standards différents soient imposés lors du développement. Ce document devra refléter ces nouveaux standards et sera révisé en conséquence.

2 Convention pour la langue de références

La langue de références qui doit être utilisée pour les noms de répertoires, fichiers, classes, méthodes, variables, base de données, table et pour les commentaires dans le code est l'anglais. Pour les documents écrits tels que celui-ci, la langue à utiliser sera le français.

3 Convention pour le code HTML

- Les balises HTML doivent être écrites en minuscules;
- Les balises HTML doivent être écrites en minuscules;
- Les différents niveaux doivent être indentés par une tabulation;
- Les tabulations doivent être de 4 caractères;
- Une ligne de code HTML doit être d'environ 80 caractères et ne jamais être plus longue que la largeur d'un écran.

Voici un exemple:

```
<html>

    <title>
        ...
    </title>
    <body>
        ...
    </body>
</html>
```

4 Convention pour le code Java et PHP

4.1 Nom des répertoires et fichiers

Les noms de répertoire doivent toujours être écrits en minuscules. Dans le cas d'un nom de répertoire de plusieurs mots, le caractère « _ » doit être utilisé pour séparer les mots. Ne jamais inclure de signe de ponctuation et d'espace. Utiliser les abréviations seulement lorsque nécessaire. Utiliser autant que possible qu'un seul mot pour décrire un nom de répertoire. Commencer un nom de répertoires avec un chiffre et toujours s'assurer que le nom est significatif pour des personnes autres que le créateur.

Pour les noms de fichier, la première lettre de chacun des mots doit être une majuscule. Éviter l'utilisation de signe de ponctuation, de caractères tels que « - » ou « _ » et tout autre caractère particulier. Ne jamais inclure d'espace dans un nom de fichier. Le nom de fichier doit décrire le module qu'il contient. Un fichier ne doit contenir qu'un seul module afin d'améliorer la clarté de l'organisation. Une seule classe par fichier et le nom de la classe doit être exactement le même que le nom du fichier.

Voici un exemple valide :

4.2 Nommage des classes et des interfaces

Le nom d'une classe doit toujours débuter avec une lettre majuscule. Par la suite, chacun des mots constituant le nom de la classe doit débuter par une lettre majuscule. Le nom du fichier contenant la classe doit porter le même nom que la classe (voir la section Nommage des répertoires et fichiers).

Pour les interfaces, la façon d'assigner les noms est la même que pour les noms avec l'ajout de la lettre « I » au début du nom.

Pour une plus grande homogénéité du code, les indicateurs de visibilité devront toujours précéder le nom d'une variable ou d'une méthode.

Débuter le nom des variables membres d'une classe avec les caractères « m_ ». Le résultat devrait ressembler à ceci : « m_NameOfMyVariable » (voir les conventions de nommage des variables pour le nom à donner à une variable). Selon les standards de code de PHP, le caractère « _ » devrait toujours précéder le nom d'une variable dont la visibilité est « private ». Cependant comme les variables membres d'une classe devraient toujours être accédées par les méthodes « getNameOfMyVariable() » || « isNameOfMyVariable » et « setNameOfMyVariable() », cette façon de faire ne devra pas être utilisée.

```
class MySubClass extends MyClass implements IMyInterface
{
    private $m_iMyAge;

    public function getMyAge()
    {
        return $this->m_iMyAge;
    }
    ....
}

interface IMyInterface
{
    public function getMyAge();
    ...
}
```

4.3 Nommage des fonctions

Le premier mot du nom d'une fonction est toujours en minuscules. Les autres mots qui forment le nom de la fonction ont toujours une majuscule comme première lettre. Le nom est descriptif de son action. Voici un exemple :

```
public  function buildMyReport($strMyInformation)
{
    ...
}
```

4.4 Nommage des variable

Même dans les langages non typés, il est important de savoir quels types de données l'on manipule. Il peut être ardu, particulièrement dans les langages non typés, de retrouver le type de valeur contenue dans une variable.

Une variable débute toujours avec un ou des caractères définissant le type de valeur qu'elle contient. Ces caractères doivent toujours être en minuscule. Vient ensuite le nom de la variable. La première lettre de chaque mot désignant le nom d'une variable doit être une lettre majuscule. Un nom ne doit pas contenir de caractère tel que:« _ », « - », de signe de ponctuation et autre caractère non standard. Ne jamais débuter un nom de variable avec un chiffre et toujours s'assurer que le nom est significatif pour des personnes autres que le créateur. Il ne devrait jamais avoir de variable avec des noms comme « i », « y », « x », ou tout autre nom non significatif. Voici la liste des caractères à utiliser en fonction des types de variable à utiliser :

- Boolean → b
- Int → i
- Long → l
- Float → f
- Short → s
- Double → d
- Char → c
- String → str
- Byte → byte
- Type complexe comme un « Array » → objArray
- Autre object → obj « NameOfObject »

- objBDD... -> object BorealisDomDoc
- objNodes... -> object Résultat Xpath
- objNode... -> un élément de objNodes
- objArray... -> un object array

Pour ce qui est des types complexes comme les objets, ils doivent toujours débuter avec l'abréviation pour le type objet « obj » suivi du type. Dans le cas d'un « Array » de nom d'animaux par exemple, le nom pourrait être « objArrayAnimalsName ». S'il s'agit d'une instance d'une classe développée par le programmeur, le nom devrait ressembler à « objNameOfMyClass » pour une classe portant le nom « NameOfMyClasse ».

Une attention doit être également portée aux variables globales et membre d'une classe. En PHP, il est possible de déclarer des variables globales comme en langage C. L'utilisation de ce type de variable peut apporter certains problèmes même s'ils sont parfois nécessaires. Dans le cas des variables membres d'une classe, il est important de pouvoir rapidement savoir si l'on accède à une variable globale d'une classe ou locale à une fonction de celle-ci. Pour ces deux cas, le code d'identification suivant sera utilisé :

- Variable glogale → g_NameOfMyVariable
- Variable membre → m_NameOfMyVariable
- Variable membre static → s_NameOfMyVariable
- Variable param → pNameOfMyVariable

Les constantes doivent être écrite toutes en majuscule et les mots séparés par le caractère « _ ».

Il faut éviter les déclarations de variable sur une seule ligne. Cette façon de faire peut porter à confusion.

Voici des exemples de nom de variables valides :

- private var $m_bIsAvailable;
- $iMyInt;
- g_bMyStatus;
- objMyClass;
- MY_CONSTANT;
- etc...

5 Exemple de documentation d'un fichier de code

5.1 Exemple PHPDoc for PHP Type Hints

http://www.nusphere.com/products/phpdoc.htm
http://manual.phpdoc.org/HTMLframesConverter/default/

5.2 Contenu

L'exemple suivant présente la documentation qui devrait normalement être présent dans un fichier de code PHP. Cette façon de faire est portable en langage Java en ne modifiant que quelques petits détails spécifiques aux façons de faire de ce langage.

5.3 Exemple

```
/**
 * Courte description du fichier
 *
 * Description plus détaillé du fichier (si besoin en est)...
 *
 * PHP versions 5
 *
 * LICENSE:
 * Contenu à définir...
 *
 * @package    Nom du paquetage
 * @author     Auteur original <auteur@example.com>
 * @author     Un autre author <autre@example.com>
 * @copyright  Boreal-IS
 * @link       http://www.boreal-is.com
 * @see        Si nécessaire
 * @deprecated Si nécessaire
 */

/**
 * Ceci est un commentaire "Docblock Comment" aussi connu sous le nom de
 * "docblock.
 * La classe docblock, plus bas, vous montre un exemple complet
 * d'utilisation.
 */
require_once 'PEAR.php';

// constants

/**
 * Valeur de retour des méthodes si elle réussissent.
 */
define('NET_SAMPLE_OK', 1);

/**
 * Description courte de la classe
 *
 * Description plus détaillé de la classe (si besoin en est)...
 *
 * @author     Auteur original <auteur@example.com>
 * @author     Un autre author <autre@example.com>
 * @version    Release: @package_version@
 * @see        NetOther, NetSample::NetSample()
 */
class NetSample
{
    // properties

    /**
     * The status of foo's universe Potential values
     * are 'good', 'fair', 'poor' and 'unknown'.
     *
     * @var string
     */
    private $m_strFoo = 'unknown';

    /**
     * The status of life
     *
     * Note that names of private properties or methods must be
     * preceeded by an underscore.
     *
     * @var boolean
     */
    private $m_bGood = true;

    /**
     * Une phrase ou paragraphe descriptif de l'utilisation.
     *
     * @param Argument1 Description de l'argument 1
     * @param Argument2 Description de l'argument 2
     *
     * @return Description de la valeur de retour
     *
     * @note Note importante s'il y a lieu
     */
    public function setFoo($strFoo)
    {
        /*
         * This is a "Block Comment."  The format is the same as
         * Docblock Comments except there is only one asterisk at the
         * top.  phpDocumentor doesn't parse these.
         */

        //This is a simple line comment
```

6 Autres conventions

- Les tabulations doivent être de quatre espaces.
- La longueur des lignes doit être d'environ 80 caractères, soit la largeur d'une page. La longueur d'une ligne ne doit jamais être plus longue que la largeur de l'écran.
- Les TODO dans le code sont sous la forme suivante :

```
// TODO: Description de la tâche à effectuer.
```

- Les bugs dans le code sont écris de la façon suivante:

```
// BUG: Description du bug.
```

- Le formatage général du code:

```
define(MY_COUNTER_LIMIT, 5);

public function myFirstFunction($iMyFirstParam, $strMySecondParam, ...)
{
    ...
    for($iMyCounter; $iMyCounter < 5; $iMyCounter++)
    {
        ...
    }
}
```

ou

```
public function mySecondFunction($bMyFirstParam,
                                 $lMySecondParam,
                                 ...)
{
    ...
    if(MyCondition)
    {
        ...
    }
}
```

ou

```
public function myThirdFunction($bMyFirstParam,
    $lMySecondParam, &strMyThirdParam, ...)
{
    ...
    if(MyCondition)
    {
        ...
    }
}
```

7 Convention pour le code SQL

Le nom des tables et colonnes de l'application doit être écrit en minuscule et les mots séparés par des « _ ».

Le code SQL de l'application doit également être écrit en minuscules. Une attention particulière doit être

apportée à l'indentation de celui-ci afin que celui-ci soit le plus lisible possible. Si la requête est effectuée dans une seule table il n'est pas nécessaire décrire le nom de la table devant chacune des colonnes référencées dans la requête. Dans le cas contraire, il faut écrire le nom de la table devant chacun des noms de colonne pour faciliter la lecture de la requête. Il est fortement conseillé d'utiliser les alias si le nom des tables est trop long. Voici un exemple invalide :

```
SELECT p.firstname, p.lastname, p.favorite_animals, a.animal_properties
    FROM player_information AS p, animals AS a
    WHERE p.favorite_animals IS dog
        AND p.favorite_animals IN (select animals_name from animals)
        AND p.age BETWEEN 5 AND 12
        AND city IN (SELECT city_name FROM available_city
            WHERE country IS canada)
    GROUP BY p.firstname;
```

8 Procédure pour la rédaction d'un bug

Il est important lorsque l'on entre un bug, ou que l'on en corrige un, d'entrer un minimum d'information afin de pouvoir faire un suivi. Voici un template pour la création d'un nouveau bug:

- **Titre:** Une courte phrase qui décrit le bug. Celui-ci ne doit jamais être changé. (Il s'agit du texte qui apparaît dans la colonne suject dans « Sugar ».
- **Description du problème :** Ici l'on d'écrit en détail le problème et ces impacts.
- **Scénario :** Ici l'on décrit précisément le scénario pour reproduire le problème.

Information à ajouter (Date courante) : Toutes information ajouté dans la description d'un bug après sa création doit-être faites dans cette section.

L'évaluation de l'impact pour le client et de la gravité d'un bug doit permettre d'assigner une priorité. Voici maintenant les informations qui doivent être contenu dans la résolution d'un problème:

- **Description de la solution :** Une description détaillé de ce qui causait le problème et de qui a été fait pour le résoudre.
- **Impact du fixe :** Description des impacts du fixe sur l'application et description de ce qui devra être vérifié avant de fermer le problème.

Ne jamais retirer de l'information d'un bug. Toujours ajouter une nouvelle section avec la date et les nouvelles précision à ajouter. Voici un exemple de ce que l'on devrait retrouvé dans la description d'un bug.

```
****** Bug description - Start (2006-07-20) **************
Description : A problem occur when I try to enter this bug description sample.

Scenario : 1- first blabla
           2- after this blabla
           3- ...

Other information to add (2006-07-21) : Each other information added in the problem description must be
****** Bug description - End *************************
```

```
****** Solution description - Start (2006-07-21) **************
Solution description : Description of the solution...

Fixe impact : Description of the impacts and the tests necessary before closing the bug.
****** Solution description - End *************************
```

Toutes les notes de travaille doivent être ajouté dans les log de travail avec la date ou la note a été entrer. Voici un exemple:

```
EA (2006-07-20): This is my working log for this sample.
```


- Dernière modification de cette page le 4 février 2009 à 20:41.