Documentation
XI-PHP

Cahier des charges

Date d'ouverture : 2007-10-09

Modification : 2007-12-19 [21:16:15]

Auteur & Chef de projet : Patrick Guay

Historique des modifications

Version	Date (de fin)	Information
1	2007-10-26	Première version fonctionnelle.
2	2007-12-19	Révision de hiérarchie des classes.

Introduction

XI-PHP est le diminutif pour eXtended Interface PHP.

Le projet est sous licence GPL version 3. : http://www.yaugsoft.com/logiciels/GPLv3.TXT et une traduction française non officielle : http://www.yaugsoft.com/logiciels/GPLv3fr.TXT

Ce projet a pour but de créer une interface entre le XHTML et PHP pour la création d'application web. Et ce, à l'aide d'une collection de classes, clef en main. De plus, il est à noter que la version 5 de PHP sera utilisée.

Cette présente documentation a pour but de décrire les procédures pour le développement du projet. Et de détailler tous les composants et la structure du projet.

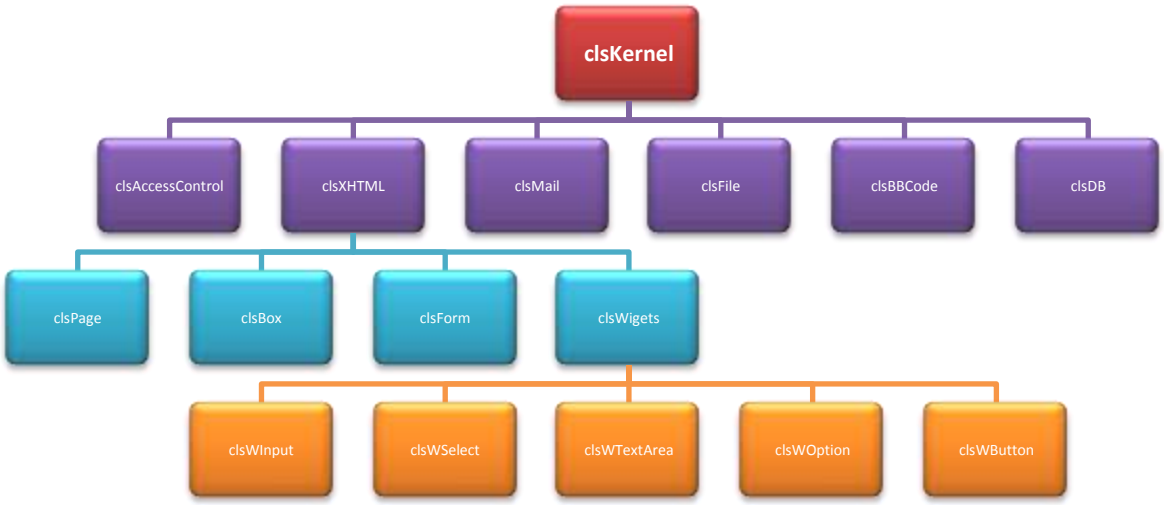
Objectifs

Le but premier du projet est de créer une interface PHP pour avoir le moins possible à coder du XHTML, certes il ne l'éliminera pas au complet, mais il tentera de le réduire au minimum. De plus, il permettra de faciliter la gestion des sessions, et l'accès aux bases de données (principalement MySQL). Et finalement, il tentera d'aider à la gestion générale de l'application web, avec la gestion des langues, la gestion de thèmes (CSS), et la configuration globale de l'application.

Autre point important, tout le projet devra principalement user des classes et de la programmation-objet (POO).

Schéma global des classes

Ce schéma représente la hiérarchie des classes de XI-PHP.



Description des classes

Voici la description générale de ce à quoi chaque classe est destinée.

clsKernel

La classe Kernel est la base de toute la hiérarchie de l'interface. C'est elle qui contient les éléments pour la manipulation des fichiers de configuration et des fichiers de langages. De plus, elle contient les informations de version de XI-PHP, soit une série de constantes qui englobent toutes les informations de l'Interface. Voici une liste de ces constantes :

XIPHP_Description	Description de l'interface
XIPHP_Version	No de version de l'interface
XIPHP_VersionDesc	Description textuelle de la version
XIPHP_Copyright	Copyright
XIPHP_Licence	Type de licence et lien
XIPHP_URLSite	Adresse du site du projet
XIPHP_URLRepos	Adresse de "repository" du projet
XIPHP_Contact	Courriel de contact du projet.

Enfin, toutes les classes de l'interface devraient, normalement, hériter de cette classe à la base.

clsAccessControl

Cette classe a pour principale mission de gérer les sessions et les "cookies", de plus c'est elle qui contrôlera le système de grade, et de groupe pour les accès de privilège aux composants de l'interface.

clsXHTML

C'est cette classe qui devrait être à la base de tout ce qui est visuel dans l'interface, et tout ce qui touche au XHTML, au CSS et l'introduction de script tel que JavaScript. C'est l'une des classes majeures de l'interface, c'est grâce à elle et tout c'est héritier, qui va permettre de réduire au minimum le code XHTML dans le projet XI-PHP lui-même ou les dérivés basés sur celle-ci.

clsPage

Cette classe fait partie des classes, dites visuelles, qui héritent de clsXHTML. Elle a pour but de contrôler tout ce qui touche les balises d'entête et de corps d'une page web XHTML. Prendre note qu'elle devra se servir de la classe clsAccessControl pour les accès aux objets de cette classe.

clsBox

Cette classe fait partie des classes, dites visuelles, qui héritent de clsXHTML. Elle permet la gestion et la conception de tout ce qui est division et section d'une page web. Prendre note qu'elle devra se servir de la classe clsAccessControl pour les accès aux objets de cette classe.

clsForm

Cette classe fait partie des classes, dites visuelles, qui héritent de clsXHTML. Comme son nom le laisse présager, c'est elle qui contrôle tous les formulaires et les wigets qui les composent.

clsWigets

Cette classe fait partie des classes, dites visuelles, qui héritent de clsXHTML. Comme son nom le laisse présager, c'est elle qui est à la base de tous les contrôles (Wigets) d'un formulaire. Chaque Widget possède sa classe, en voici la liste : clsWInput, clsWSelect, clsWTextArea, clsWOption, lsWButton.

clsMail

Pour cette classe, son but est la gestion de tout ce qui est envois de courriels.

clsFile

Cette classe est là pour la manipulation de tout ce qui est fichier et dossier. La gestion des chemins (Path) pour le fonctionnement de l'interface se fait au niveau de clsKernel.

clsBBCode

Cette classe permettra la gestion et l'introduction du système de BBCode, qui est souvent utilisé sur le web. Nous pensons qu'un système reconnu de tag sera plus bénéfique, plutôt que d'en créer un nouveau.

clsDB

C'est cette classe qui a pour mission de gérer les accès aux bases de données (MySQL seulement) et leur manipulation.

Conventions de codage

Les conventions de codage permettent de conserver une cohérence et une consistance au code, pour ainsi le rendre plus lisible et au final être maintenu plus facilement par les développeurs.

Prendre note que cette présente convention a beaucoup été inspirée du projet en PHP, Pear. D'ailleurs, certains paragraphes sont copiés intégralement du manuel de Pear. De plus, la plupart des exemples sont également tirés du PHP de ce manuel.

Indentation et longueur des lignes

L'indentation devra être de 4 espaces. Ne pas utiliser le caractère de tabulation pour l'indentation si possible, habituellement cette fonction est configurable dans les logiciels d'édition.

À moins de raison valable, les lignes ne devraient pas dépasser les 80 à 85 caractères.

Structures de contrôles

Les structures de contrôles incluent les : "if", "for", "try-catch", "foreach", etc. Il se fait avec le style K&R.

Les instructions de contrôle doivent avoir un espace entre le mot clé de l'instruction et la parenthèse ouvrante, afin de les distinguer des appels de fonctions.

Il est vivement recommandé de toujours utiliser des accolades, même dans les situations où elles sont techniquement optionnelles. Leur présence augmente la lisibilité du code et réduit le risque d'erreur logique lors de l'ajout de nouvelles lignes de code. Voici quelques exemples :

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
```

```
switch (condition) {
case 1:
    action1;
    break;

case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}
```

Appels de Fonctions

Les fonctions doivent être appelées sans espace entre le nom de la fonction, la parenthèse ouvrante, et le premier paramètre ; avec un espace entre la virgule et chaque paramètre et aucun espace entre le dernier paramètre, la parenthèse fermante et le point virgule. Voici un exemple :

```
$var = foo($bar, $baz, $quux);
```

Comme montré ci-dessus, il doit y avoir un espace de chaque côté du signe égal utilisé pour affecter la valeur de retour de la fonction à une variable. Dans le cas d'un bloc d'instructions similaires, des espaces supplémentaires peuvent être ajoutés pour améliorer la lisibilité :

```
$short         = foo($bar);
$long_variable = foo($baz);
```

Définitions des fonctions

La déclaration des fonctions respecte l'indentation du style << BSD/Allman >> :

```
function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}
```

Les arguments possédant des valeurs par défaut vont à la fin de la liste des arguments.

Contrairement au contrôle, l'accolage d'ouverture ce situe sur la ligne suivante et non à la suite (style K&R) de la déclaration de la fonction.

Commentaire (PHP)

Tous les commentaires de fin de ligne ou qui se trouvent à l'intérieur d'un bloc se font avec les commentaires standard C++ : "//".

```
$iMaVar = 12;  //Ceci est un commentaire
```

Les commentaires du type C : "/* */" serve pour les entêtes de fonction, de classe, procédure, routine, etc. Et doivent être inscrite selon les standards compatibles pour PHPDocumentor. Voici un exemple avec les tags (@) les plus courants :

```
/**
 * The short description
 *
 * As many lines of extendend description as you want {@link element}
 * links to an element
 * {@link http://www.example.com Example hyperlink inline link} links to
 * a website. The inline
 * source tag displays function source code in the description:
 * {@source }
 *
 * In addition, in version 1.2+ one can link to extended documentation like this
 * documentation using {@tutorial phpDocumentor/phpDocumentor.howto.pkg}
 * In a method/class var, {@inheritdoc may be used to copy documentation from}
 * the parent method
 * {@internal
 * This paragraph explains very detailed information that will only
 * be of use to advanced developers, and can contain
 * {@link http://www.example.com Other inline links!} as well as text}}}
 *
 * Here are the tags:
 *
 * @abstract
 * @access       public or private
 * @author       author name <author@email>
 * @copyright    name date
 * @deprecated   description
 * @deprec       alias for deprecated
 * @example      /path/to/example
 * @exception    Javadoc-compatible, use as needed
 * @global       type $globalvarname
   or
 * @global       type description of global variable usage in a function
 * @ignore
 * @internal     private information for advanced developers only
 * @param        type [$varname] description
 * @return       type description
 * @link         URL
 * @name         procpagealias
   or
 * @name         $globalvaralias
 * @magic        phpdoc.de compatibility
 * @package      package name
 * @see          name of another element that can be documented,
 *                produces a link to it in the documentation
 * @since        a version or a date
 * @static
 * @staticvar    type description of static variable usage in a function
 * @subpackage    sub package name, groupings inside of a project
 * @throws       Javadoc-compatible, use as needed
 * @todo         phpdoc.de compatibility
 * @var        type    a data type for a class variable
 * @version    version
 */
function if_there_is_an_inline_source_tag_this_must_be_a_function()
{
// ...
}
```

Pour plus d'info : http://www.phpdoc.org/

Convention des noms

Les noms donnés aux variables, fonctions et autres doivent toujours suivre un format standard, les normes établies ici suivant en partie le principe de la technique hongroise.

Règles générales : Le caractère "_" ne doit jamais servir, autre que pour déterminer une catégorie ou section. (Ex. : sLang_TextIntro ou clsForm_Cadre ou clsForm_Contenue…) et pour les prépréfixes de variable. Autres endroits où le "_" peut être utilisé, et ce doit être le premier caractère, c'est pour nommer une variable globale (ex. : _MaVarGlobale ou _Config_NbrDeLigne). Et enfin, pour les constantes (voir plus bas).

Les classes & Interface

Les classes doivent toujours débuter avec le préfixe "cls" en minuscule suivi du nom de la classe débutant avec une majuscule. Pour les interfaces c'est "int"

```
clsFormulaire
clsKernel
intMonInterface
```

Fonctions, procédure et Méthodes

Les fonctions et Cie doivent débuter par une majuscule et doivent être composées d'un verbe décrivant l'action de la fonction. Si le nom est composé de plusieurs mots, les autres mots débutent avec une majuscule également. Prendre note que la notion de catégorie ou section ne devrait pas être utilisée pour les fonctions et Cie, utilisée de préférence les classe pour les regrouper.

```
AfficheCadre()
Supprimer()
FermetureBdExterne()
```

Constantes

Les constantes sont toujours en majuscule. Si le nom est composé de plusieurs mots, ils sont séparés par des "_" . (Ex. : MA_CONTANTE) à l'exception d'une constante avec une catégorie ou section, le nom débute avec la catégorie suivi de "_" et le reste des mots avec une majuscule seulement devant les mots (ex. : LANG_TexteIntroduction).

Les variables

Voilà un gros morceau! Toutes les variables doivent commencer avec un préfixe en minuscule qui détermine le type de la variable. Voici un tableau non exhaustif des types de variables.

Bien que PHP ne soit pas un langage typé, il est donc deux fois plus important de bien les identifier pour ne pas les confondre et affecter une mauvaise valeur. Si pour une raison ou une autre les valeurs son de type variable utilisé le type variant "v" pour le signaler

Variables de bases

Préfixe	Description
b	Boolean, Vrai ou Faux
bi	Binaire, Base 2
by	bytes, 1octet
c	Char, Caractère unique
cu	Currency, $$$
d	Décimal
dt	Date / heure
hx	Hexadécimal, Base 16
i	Integer, le i utilisé seul = conteur standard pour un for par exemple.
o	Objet
oc	Octodécimal, Base 8
s	String, chaine de caractères
t + (un préfixe)	Tableau avec type de donnée du tableau tiMonTab (tableau d'integer)
v	Variant, mixte
Il existe également les prépréfixes de portée : _ : déjà mentionné plus haut pour indiquer une variable globale. m_ : variable de module, private x_ : variable statique p_ : Propriété privée _get, _set *Exemple : m_iMaVariable (Integer private), p_sMaVariable (String, property private)*	

Règles de base générales

Voici maintenant, quelques règles d'ordre général.

Classes

Techniquement, il ne devrait jamais avoir plus d'une classe dans un fichier et le nom du fichier porte le nom de la classe qu'il contient.

Dossier web (PHP, XHTML)

Lors d'une application web, il ne doit, en aucun cas, y avoir un dossier sans un fichier "index.htm" ou "index.php" et ce fichier doit contenir au minimum le code suivant :

```
<script>history.go(-1);</script>

ou

<?php
    header("Location: http://www.unsite.com");
?>
```

Ceci empêche de voir la liste des fichiers d'un dossier, cela semble anodin dans un dossier qui ne contient que des images, mais un dossier qui aurait les configurations du site risque d'être plutôt embarrassant.