**v1.7**

2012-2013

**Webframework**

**Guide du développeur**

**Thomas AUGUEY**

# Charte de qualité du développement

Webframework est orienté développement, ce n’est pas un CMS définissant les limites du modèle de votre application. Dans cette optique il est essentiel de suivre certaines règles de développement pour permettre une bonne réutilisation du code.

1. Un développeur sera souvent amené à étendre les fonctionnalités d’un module pour les besoins de son application. Dans un tel cas, il doit pouvoir reprendre aisément le développement au niveau conceptuel (UML) pour l’intégrer à son projet.
2. Les fonctionnalités de doivent pas être dépendante des vues, ainsi, si certaines fonctions écrites en JavaScript par exemple sont nécessaire à l’utilisation du module, elles doivent être incluses dans les fichiers de librairies de code et non directement dans les vues HTML. Un programmeur utilisera rarement les vues proposer en exemple de test, c’est pourquoi l’API doit être totalement séparée de l’implémentation visuel.
3. Le code source doit être documenté (**grammaire Doxygen**)
4. Le model données/objet doit être construit sur la méthode **UML**

# Model d’application

# Model d’application étendu

L’Utilisation des librairies étendues de **Webframework** impose une architecture minimale de votre l’application.

L’Application doit implémenter les interfaces suivantes :

* iApplication Point d’entrée de l’application
* iDatabase Communication avec le système de base de données

Pour vous aider, **Webframework** propose les implémentations suivantes :

* cApplication Application générique avec fichier de configuration
* cDataBasePostgres Interface avec une Base de données PostgreSQL 8

Vous trouverez un exemple complet d’intégration minimal dans le répertoire « **minimal** ».

# Modules

Le développement de module sous Webframework est motivé par la possibilité pour le programmeur de réutiliser des bases de fonctionnalités pour son programme.

Chaque module doit être développé sur la **charte de qualité** vue plus haut.

## Développement

Le développement d’un module doit être le plus indépendant possible de l’application avec laquelle elle fonctionnera.

Un module implémente l’interface iModule. C’est l’implémentation de cette classe qui servira de contrôleur intermédiaire pour l’application.

L’Application ne doit pas être dépendante de l’architecture d’un module, celui-ci étant indépendant et pouvant être utilisé par plusieurs applications simultanément.

La résolution des chemins d’accès passe par l’implémentation de l’interface iApplication définissant dans sa configuration les chemins d’accès aux différentes librairies et modules qu’elle utilise.

Un module ne stock jamais dans son arborescence des informations relatives à l’application qui l’utilise.

## Intégration

* Définir l’implémentation de l’interface iApplication dans une variable global nommée $app.
* Inclure le fichier **module.php** inclue à la racine de l’arborescence du module.
* Appeler en début de script la méthode statique ModuleClassName::load("chemin/d’accès/relatif/au/module");

# Résultat de procédure

Webframework propose un modèle de gestion des erreurs.

Une erreur est toujours composée de deux éléments :

* Une classe
* Un identifiant

La classe définit le niveau d’abstraction auquel se trouve l’erreur (Système, Application, Base de données, etc..).

L’identifiant précise le type d’erreur qui a été rencontrée (Opération sur un fichier, Fichier introuvable, Requête SQL invalide, etc…).

## Erreur au niveau procédurale

**function myFunction(){**

**if(Error)**

**return RESULT( MY\_CLASS, ’MY\_ERROR’ );**

**return RESULT(0);**

**}**

**if( !myFunction() ){**

**// hum, quelle erreur ?**

**switch( LAST\_ERROR() ){**

**case ‘NO\_USER\_DETECTED’:**

**break;**

**default:**

**return;**

**}**

**}**

## Erreur au niveau procédurale (SQL)