

2013

**Webframework 1.8.1**

**Document Technique**

**Thomas AUGUEY**

Table des matières

[Introduction 4](#_Toc356912374)

[Vous avez dit sur-mesure ? 4](#_Toc356912375)

[Fonctionnalités 5](#_Toc356912376)

[Requis 6](#_Toc356912377)

[Librairie 7](#_Toc356912378)

[Format de fichier INI avancé 7](#_Toc356912379)

[Constantes 7](#_Toc356912380)

[Inclusions 7](#_Toc356912381)

[Résolution des doublons 7](#_Toc356912382)

[Résultats de procédure 8](#_Toc356912383)

[Structure d’un résultat 8](#_Toc356912384)

[Utiliser dans une fonction 8](#_Toc356912385)

[Passage de résultat 9](#_Toc356912386)

[Surcharge de résultat 10](#_Toc356912387)

[Templates XML 11](#_Toc356912388)

[Introduction 11](#_Toc356912389)

[Principe 11](#_Toc356912390)

[Conseils 11](#_Toc356912391)

[Espace de nommage 12](#_Toc356912392)

[Actions 12](#_Toc356912393)

[Sélection 12](#_Toc356912394)

[Les Marqueurs 13](#_Toc356912395)

[Compiler un template 14](#_Toc356912396)

[Exemple simple (à vérifier) 14](#_Toc356912397)

[Contrôle de champs 16](#_Toc356912398)

[Codes d’erreurs 17](#_Toc356912399)

[Détail des codes 17](#_Toc356912400)

[Détail des messages 17](#_Toc356912401)

[Contenu Obsolète 18](#_Toc356912402)

[Application 19](#_Toc356912403)

[Recommandation et développement 19](#_Toc356912404)

[Architecture 20](#_Toc356912405)

[Paramètres de configuration 21](#_Toc356912406)

[Configuration 22](#_Toc356912407)

[Arborescence des fichiers 23](#_Toc356912408)

[Dictionnaire de données (fields.ini) 24](#_Toc356912409)

[Fichiers SQL (sql.ini) 24](#_Toc356912410)

[Model de données 25](#_Toc356912411)

[Format de champ 25](#_Toc356912412)

[Description de champ 25](#_Toc356912413)

[Champs réservés par Webframework 26](#_Toc356912414)

[Modélisation des données 27](#_Toc356912415)

[Vues 28](#_Toc356912416)

[Paramètres de configuration 28](#_Toc356912417)

[Contrôleurs 29](#_Toc356912418)

[Exemple d’implémentation 29](#_Toc356912419)

[Paramètres de configuration 29](#_Toc356912420)

[Exemple 30](#_Toc356912421)

[BootStrap 31](#_Toc356912422)

[Permissions 31](#_Toc356912423)

[Un système simple et personnalisable 31](#_Toc356912424)

[Implémentation 31](#_Toc356912425)

[Exemple 31](#_Toc356912426)

[Chemins d’accès 34](#_Toc356912427)

[Répertoire de travail (PATH) 34](#_Toc356912428)

[Hôte (URL) 34](#_Toc356912429)

[Pages en lignes (URL) 34](#_Toc356912430)

[Base de données 36](#_Toc356912431)

[Paramètres de configuration 36](#_Toc356912432)

[Résultat de procédure 36](#_Toc356912433)

[Template 36](#_Toc356912434)

[Modules 36](#_Toc356912435)

[Développement 37](#_Toc356912436)

[Point d’entrée de l’application 37](#_Toc356912437)

[API Références 37](#_Toc356912438)

[Intégration Architecture 37](#_Toc356912439)

[Intégration MVC 38](#_Toc356912440)

[Intégration Configuration 39](#_Toc356912441)

[Notes 40](#_Toc356912442)

[Windows spécifique 40](#_Toc356912443)

[FAQ 40](#_Toc356912444)

[DOMDocument::loadHTMLFile et CDATA 40](#_Toc356912445)

[*Solution* 40](#_Toc356912446)

[favicon.ico vide cause de multiples requêtes? 40](#_Toc356912447)

[*Solution* 40](#_Toc356912448)

[A Faire 41](#_Toc356912449)

[php/class/bases/cApplication : translateResult() 41](#_Toc356912450)

[Bugs 41](#_Toc356912451)

[php/ini\_parse.php 41](#_Toc356912452)

# Introduction

**WebFrameWork** rassemble au sein d’un même paquetage des outils de développements cohérent pour aider le développeur à déployer des applications web sur-mesure.

## Vous avez dit sur-mesure ?

En effet, le but de ce paquetage et d’équilibrer au maximum la balance entre contrainte et liberté de développement.

**La Contrainte,** c’estla possibilité pour le programmeur d’exploiter au maximum les fonctionnalités du Framework et de réduire ses coûts de développement.

*« Un Framework fonctionnel et robuste permettra un déploiement rapide et une intervention réduite du programmeur. A l’inverse, le programmeur doit se plier aux règles de développement imposées par le Framework et dispose d’une liberté restreinte en terme de conception. »*

**La Liberté,** c’est la possibilité pour le programmeur d’intégrer certains composants du Framework dans un projet existant et imposant l’utilisation d’autres technologies.

*« Un Framework adaptable permettra d’utiliser certains de ses outils indépendamment les uns des autres, le tout sans entrer en conflit et sans imposer la dépendance d’un système global. Cette méthode offre au programmeur une liberté totale et des possibilités illimitées. A l’inverse, la marge d’action du Framework est réduite et permet moins d’automation laissant plus de travail au développeur vis-à-vis de l’architecture et des technologies utilisées. »*

**Webframework** est basé sur cette philosophie de développement : *Le développeur qui fixe les règles, le Framework comble les trous…*

**Pour répondre à cette philosophie contradictoire, Webframework propose deux niveaux de développement :**

1. Le programmeur à une vision définit de l’architecture de son application. Dans ce cas, les librairies fondamentales pourront l’aider dans sa tâche. Indépendantes de tout système et de dépendances, les librairies fondamentales ne posent aucun problème de compatibilité.
2. Le programmeur dispose d’une vision finale de l’application et prévoit un développement rapide et robuste. Dans ce cas, il pourra s’appuyer sur le modèle d’application existant et toutes les fonctionnalités qui en découlent (sécurité, modèle de données, modules, etc…).

## **Fonctionnalités**

**Si votre projet prévoie d’utiliser une ou plusieurs des technologies suivantes, Webframework est idéalement fait pour vous !**

|  |  |  |
| --- | --- | --- |
| Composant | Fondamentaux  (Librairies) | Etendu  (Application) |
| Moteur de Templates | Transformer des fichiers XML/Textes. | Gestion simplifiée des vues. |
| Interface SGBD | Abstraction avec la base de données. |  |
| Système de résultat | Traçabilité et traitement des erreurs. | Traductions linguistique automatique. |
| Configuration INI avancé | Configuration, inclusions et constantes de champs. | Fusion avec des sous-applications. |
| Contrôle de champs | Nombreux formats existants. | Vérification automatique des entrées de contrôleur.  Dictionnaire de données. |
| Contrôle de permissions utilisateur |  | Contrôle automatique avant exécution de contrôleur. |
| XML Default File | Meta données de l’application (pages, navigation, contact, textes, etc…). | Automatisation des traductions linguistique. |
| X-Arguments | Format d’échange simple de données associatives. | Format de sortie (contrôleur). |
| Gestionnaire de tâches | Programmation des commandes systèmes. | Programmation d’exécution de contrôleurs. |

## **Requis**

**Si votre projet prévoie d’utiliser une ou plusieurs des technologies suivantes, Webframework est idéalement fait pour vous !**

|  |  |  |  |
| --- | --- | --- | --- |
| **SERVEUR** | https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcT6hg33P9QbpX15FZLOHHZkcK3hzJxn-7fJIRmvBVVbzg45SzHi | **Windows / Linux** | **Webframework fonctionne aussi bien sur un système Linux ou Windows.** |
| https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcSOo6cY58KayVDg0qkCcl6YlyvDXH35Iy0_PIoMG1GybzszXvp2Zw | **PHP**  **v5.4+** | **L’Ensemble du Framework coté serveur est basé sur le langage PHP.** |
| **CLIENT** | http://asociaux.fr/public/logos/jquery-logo.png | **JQuery**  **v1.x | v2.x** | **Librairies développées pour les utilisateurs de JQuery.** |
| http://reid.github.com/decks/2011/jsconf/logo.png | **YUI**  **v3.7.3+** | **Librairies développées pour les utilisateurs de YahooUI.** |
| http://upload.wikimedia.org/wikipedia/fr/a/a2/Ext_js_logo.jpeg | **ExtJS**  **v4.1.1+** | **Webframework propose dans ces modules une collection de Widgets et vues réutilisable pour vos applications ExtJS.** |
| **SGBD** | http://pix-mania.dyndns.org/mediawiki/images/b/b5/Postgresql.png | **Postgres**  **v8.4+** | **Webframework utilise une interface d’abstraction avec les bases de données.**  **Cependant, les modules et exemples sont tous basés sur la base de données PostgreSQL.** |

# Librairie

## Format de fichier INI avancé

Depuis la révision « **174** » Webframework a introduit une gestion avancée des fichiers .ini. La fonction parse\_ini\_file\_ex remplace maintenant l’utilisation de la fonction standard de PHP parse\_ini\_file.

### Constantes

La définition de constante permet d’éviter la redondance dans les définitions de chaines, les constantes sont notamment pratique pour définir des bases de chemins d’accès.

@global nom =   "valeur"

Pour utiliser une constante entourez celle-ci des caractères ${ et } :

my\_value =   "${my\_const}"

### Inclusions

La balise @include permet d’inclure le contenu d’un autre fichier de configuration.

@include   "nom\_du\_fichier.ini"

### Résolution des doublons

Si dans un fichier vous définissez plusieurs fois une même section, parse\_ini\_file\_ex fusionnera les paramètres des sections trouvés pour en former plus qu’une.

Les paramètres définit en doubles seront écrasés dans l’ordre de leurs définitions, c’est-à-dire du haut vers le bas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [my\_section]  foo = "bar"  bar = "foo" | **+** | [my\_section]  foo = "not bar"  smile="happiness" | **=** | [my\_section]  foo = "not bar"  bar = "foo"  smile="happiness" |

## Résultats de procédure

Pour identifier les erreurs, **Webframework** utilise un système de résultat simple mais efficace.

Chaque fonction ou méthode utilisant ce système retourne un code de résultat à la classe cResult. cResult retient ce code dans une variable static, ce qui permet au programmeur de savoir si, oui ou non, la dernière procédure exécutée est un succès.

### Structure d’un résultat

Données associatives venant compléter l’erreur

(file=> coin.bin ; cause=>Données invalides à l’offset 51200 ; conseil=>Remplacer le fichier)

Description succincte de l’erreur (ex : Fichier erroné)

Contexte de l’erreur (Succès ; Echec; Erreur du système ; etc…)

Avantages

Remonté de l’information :

Il n’est pas nécessaire pour la fonction appelante de « retourner » l’information en remontant le résultat d’une sous procédure. L’avantage ici, est de conserver la main mise sur les traitements à effectuer en cas d’erreur. Au contraire, le programmeur pourra laisser le traitement à la fonction appelante si il ne modifie pas le statut de résultat.

Conventions

Valeur de retour :

Il est conseillé aux fonctions utilisant le système de toujours retourner une valeur booléenne. Une procédure peut ainsi simplement retourné son état d’exécution : succès ou échec.

L’utilisation d’un booleen n’est pas une obligation du moment que la fonction indique son état de résultat avant le retour de la fonction.

### Utiliser dans une fonction

Chaque fonction qui utilise ce système doit obligatoirement avant de terminé son execution, retourné un code à la classe cResult. Qu’il s’agisse de conserver le résultat précédent ou de définir un nouveau résultat, le programmeur doit pouvoir connaitre le statut de la dernière procédure appelée.

Deux choix :

* Passer le résultat existant
* Surcharger le résultat avec un nouveau code

### Passage de résultat

Dans cet exemple on observe une pile d’appel de fonctions. Le code d’erreur retourné par **Fonction D** remonte vers **Fonction A (qui la traite). Fonction B et Fonction C se contente de conserver le résultat de Fonction D.**



**Fonction A** reçoit le code **E1** comme erreur

Function D() {

return RESULT( cResult::Failed, "E1");

}

Function C() {

if( !D() ) return false; // converse le résultat de D

return RESULT\_OK();

}

Function B() {

if( !C() ) return false; // converse le résultat de C

return RESULT\_OK();

}

Function A() {

if( !B() ){

cResult::getLast()->getError() == "E1";

return false; // converse le résultat de B

}

return RESULT\_OK();

}

### Surcharge de résultat

Dans cet exemple on observe une pile d’appel de fonctions. Le code d’erreur retourné par **Fonction D** est surchargé par **Fonction B. Au final, Fonction A aura uniquement connaissance du code de résultat E2.**



**Fonction A** reçoit le code **E2** comme erreur

Function D() {

return RESULT( cResult::Failed, "E1");

}

Function C() {

if( !D() ) return false; // converse le résultat de D

return RESULT\_OK();

}

Function B() {

if( !C() ) return RESULT( cResult::Failed, "E2"); // surcharge le résultat de C

return RESULT\_OK();

}

Function A() {

if( !B() ){

cResult::getLast()->getError() == "E2";

return false; // converse le résultat de B

}

return RESULT\_OK();

}

## Templates XML

### Introduction

Webframework a introduit un système de Template évolué, basé sur le format XML.

Plusieurs avantages à utiliser ce format :

* Implémenté sous **Javascript** et **PHP**
* Sélecteur de type **CSS**
* Utilisable comme base de données
* Extensible par modules de classes

### Principe

**XMLTemplate** est basé sur un principe simple : l’intégralité des éléments du document est analysé et transformé à la volé. La transformation à effectuer est définit par des balises d’actions et des marqueurs dans le texte du document.

Procédure de transformation d’un document:

1. Le document XML et ouvert comme source en mémoire
2. Chacun des éléments sont analysés l’un après l’autre
   1. Les textes et attributs enfants de l’élément sont analysés à la recherche de marqueurs
      1. Les marqueurs appliquent un traitement et retournent un texte de remplacement
   2. L’élément en cours est analysé à la recherche d’une balise d’action
      1. L’Elément et ses éléments enfants sont transformés
   3. Les attributs qui utilisent l’espace de nommage **Template sont supprimés**
   4. Le processus continue à l’élément suivant
3. **Le document est converti en chaine de caractères**
4. **Un traitement final est effectué sur l’ensemble du texte**
5. **Le nouveau document est créé**

Les concepts fondamentaux d’un Template XML:

* **Le curseur** (Pointe sur l’élément en cours de transformation)
* **La sélection** (Pointe sur un élément externe au document)
* **L’action** (Transforme un élément XML)
* **Le marqueur** (Transforme un texte dans un élément ou un attribut XML)

Dans un document template webframework, le principe est de sélectionner des éléments à insérer et de les transformer par le biais d’une action.

### Conseils

Voici quelques conseils pour rédiger correctement vos documents XML/XHTML.

#### Délimiter les zones de scripts

Certains caractères réservés par HTML peuvent être présents entre les balises de script. Pour éviter un problème de chargement utilisez la balise CDATA pour indiquer à l’interpréteur qu’il ne s’agit pas d’un code HTML.

<script>  
// <![CDATA[  
...CODE JAVASCRIPT...  
// ]]>  
</script>

#### Encodage

Utilisez de préférence un encodage UTF-8 en accord avec votre application

* Format UTF8 (signé ou non-signé)
* Codage des caractères spéciaux (ex: &amp; remplace &)

### Espace de nommage

La première chose à faire consiste à définir l’espace de nommage suivant :

http://www.webframework.fr/last/xmlns/template

Généralement définit dans la balise HTML et le préfixe template :

<html xmlns:template="http://www.webframework.fr/last/xmlns/template">

### Actions

Une action transforme un élément et ses enfants. Les actions peuvent avoir des effets différents suivant sont type.

Pour définir une action utilisez l’attribut action sur l’élément de votre choix:

<node template:action="type"></node>

Voici les actions disponibles :

|  |  |  |
| --- | --- | --- |
| Type | Description | Attributs complémentaires |
| All | Applique une sélection de style CSS. Pour chaque élément trouvé, le nœud est dupliqué et transformé. | selector  Selecteur CSS (voir ) |
|  |  |  |

Les actions transforment le noeud ou ils sont implentés et tous les noeuds enfants. Il existe différent types d'actions, chacune à son utilité et peuvent souvant être combinées ensembles.

[**Voir les types d'actions**](http://www.aceteam.org/wfw-template_actions.html)

### Sélection

La sélection pointe sur un élément externe du document. Elle est source de données pour l’action qui transforme un nœud.

Pour changer de sélection, définissez l’attribut template:path avec le chemin d’accès à l’élément désiré, si path n’est pas définit la sélection en cours reste inchangé.

#### Sélecteur de chemin

Ce sélecteur simple permet de pointer sur un élément XML en définissant successivement ses éléments parents.

Il existe 3 types de sélections :

1. **Absolue**, depuis la racine du document (ajouter un / en début de ligne):

<node template:path="/root/books/blue\_sky" />

1. **Relative**, depuis le nœud en cours (pas de / en début de ligne):

<node template:path="/root/books">

<node template:path="blue\_sky" />

</node>

1. **Absolue depuis un fichier**, depuis un autre document (ajouter un : en début de ligne) :

<node template:path=":biblio.xml:/root/books/blue\_sky" />

La sélection reste constante tant qu’elle n’est pas modifiée par une action ou l’attribut path :

<node template:path="/root">  
 <node template:path="book/bible">   
 Le texte contenu dans l’élément "/root/book/bible/description" est "-{description}"  
 </ node >   
 Le texte contenu dans l’élément "/root/description" est "-{description}"  
</ node >

#### Sélecteur CSS

Vous pouvez utiliser des sélecteur de style **CSS** avec les actions one et all.

### Les Marqueurs

Les marqueurs permettent d’insérer du texte à un emplacement donné, ils peuvent être utilisés dans des textes ou des attributs.

#### Marqueur simple

-{syntaxe}

Marqueur simple avec texte alternatif :

-{syntaxe|'Texte alternatif'}

#### Marqueurs prédéfinit

Certain marqueurs sont prédéfinit dès le début de la transformation:

|  |  |
| --- | --- |
| -{\_\_date\_rfc822\_\_} | Retourne la date en cours au format RFC822 |
| -{\_\_date\_w3c\_\_} | Retourne la date en cours au format W3C |
| -{\_\_timpestamp\_\_} | Retourne la date en cours au format Timestamp UNIX |
| -{\_\_uri\_\_} | Retourne le nom de domaine du site |
| -{\_\_hostname\_\_} | Nom d'hôte du serveur |
| -{\_\_file\_ctime\_\_} | Date de création du fichier (timestamp) (PHP seulement) |
| -{\_\_file\_mtime\_\_} | Date de dernière modification du fichier (timestamp) |
| -{\_\_file\_ctime\_rfc822\_\_} | Date de création du fichier (RFC822) |
| -{\_\_file\_mtime\_rfc822\_\_} | Date de dernière modification du fichier (RFC822) |

#### Définir un marqueur

Les marqueurs peuvent être utilisés dans les textes ou des attributs

<element attribute="ma valeur ici : -{noeud}">  
 Ou ici : –{noeud}  
</element>

#### Imbriquer des marqueurs

Il est possible d’imbriquer les marqueurs en ajoutant une accolade supplémentaire au marqueur de niveau inférieur.

Voici un exemple de deux marqueurs imbriqués :

-{{seconde-{first}}}

Ici, first sera traité en premier, puis seconde en deuxième. Notez que second sera influence par la valeur de first, c’est là que réside tout l’intérêt de l’imbrication des marqueurs.

### Compiler un template

Pour compiler un template plusieurs possibilités :

#### Sous PHP

Depuis une ligne de commande ou un script avec la fonction rexe.

#### Sous Javascript

Avec la fonction **[wfw.ext.template.make](http://www.aceteam.org/wfw-template.html" \l "make)**.

#### Avec une requête HTTP

Depuis le gestionnaire de requête Javascript « **[wfw.request](http://www.aceteam.org/lib-wfw-wfw.request.html)** » ou directement dans l’URI du navigateur.

Pour plus d’information sur la requête template, voir

### Exemple simple (à vérifier)

#### Document en sélection

Le document qui servira de source de données.

<root>  
<titre\_du\_document>Hello World !</titre\_du\_document>  
<my\_list>   
<sub\_element>Item1</sub\_element>  
<sub\_element>Item2</sub\_element>  
<sub\_element>Item3</sub\_element>  
<sub\_element>Item4</sub\_element>  
</my\_list>  
</root>

#### Document à transformer

Ici une page HTML ou sera importé une liste d’élément.

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:template="http://www.webframework.fr/last/xmlns/template">  
<head>  
<title>-{titre\_du\_document}</title>  
</head>  
<body>  
<div template:action="array" select="my\_list">-{sub\_element}</div>   
</body>  
</html>

#### Document transformé

Le résultat final une fois transformé. On remarque que les textes ont été importés puis incorporés au document HTML.

<html xmlns="http://www.w3.org/1999/xhtml" xmlns:template="http://www.webframework.fr/last/xmlns/template">  
<head>  
<title>Hello World !</title>  
</head>  
<body>  
<div>Item1</div>  
<div>Item2</div>  
<div>Item3</div>  
<div>Item4</div>   
</body>  
</html>

## Contrôle de champs

La classe de base cInput disponible dans le dossier php/class/bases/input.php permet d’étendre la collection de formats existants.

Liste des formats existants

Plusieurs classes prédéfinit sont disponibles. Sous PHP les classes sont définis dans le dossier php/inputs/.

|  |  |  |
| --- | --- | --- |
| Identifiant | Description | Expression régulière |
| Identifier | Identificateur  [ static , \_foo\_bar\_ , my\_objectId ] | [a-zA-Z\_]{1}[a-zA-Z0-9\_]\* |
| integer | Entier numérique  [ 0 , 45 , 1565416 ] | 0|([1-9]{1}[0-9]\*) |
| name | Nom, permet plus de liberté qu’un identifiant tout en excluant les caractères d’espacement ou caractères spéciaux.  [ Ceci.est-un-nom\_valide ] | [a-zA-Z\_]{1}[a-zA-Z0-9\_\-\.]\* |
| mail | Adresse de courrier électronique, basé sur le standard RFC-2822.  [ hello@world.org ] | - |
| unixfilename | Nom de fichier au format UNIX (les chemins ne sont acceptés).  dummy.dat | - |
| password | Mot de passe.  [ Mon#mot-De-passe\_50\_ ] | [a-zA-Z0-9\_\-\@\#\&\+\~]+ |
| string | Texte compris entre doubles-quotes  [ Ceci est une string valide ] | [^"\n\r]\* |

## Codes d’erreurs

Les résultats d’opération basés sur la librairie **WebframeWork**.

Ces messages doivent être placés dans le document « **default.xml** » du site web pour être traduit automatiquement.

### Détail des codes

**Contexte Code Description**

ERR\_OK VALID\_INPUT Le champ est valide

ERR\_FAILED NO\_INPUT\_FIELD Aucun champ reçu

ERR\_FAILED MISSING\_FIELD Champ manquant

ERR\_FAILED EMPTY\_TEXT Champ vide

ERR\_FAILED INVALID\_CHAR Champ contenant des caractères interdits

ERR\_FAILED INVALID\_FORMAT Champ mal formé

ERR\_FAILED DB\_CONNECTION Connexion au serveur de base de données a échouée

ERR\_FAILED DB\_SQL\_QUERY Requête SQL mal formée

ERR\_FAILED SOCK\_OPEN\_URL La connexion au serveur a échouée

ERR\_FAILED INVALID\_RANGE Rang de valeur invalide

ERR\_FAILED OVERSIZED Trop de caractères

ERR\_FAILED UNDERSIZED Pas assez de caractères

ERR\_FAILED XML\_TEMPLATE\_NO\_INPUT\_FILE Aucun fichier présent en entrée

ERR\_FAILED XML\_TEMPLATE\_NO\_INPUT\_ELEMENT Pas d’élément en entrée

ERR\_FAILED XML\_TEMPLATE\_CANT\_LOAD\_INPUT\_FILE Le fichier d’entrée ne peut pas être chargé

ERR\_FAILED APP\_UNKNOWN\_FORM\_TEMPLATE\_FILE Le Template de formulaire est indéfini

ERR\_SYSTEM SYS\_TASK\_CREATE La tâche ne peut pas être créée

ERR\_SYSTEM SYS\_TASK\_UPDATE La tâche ne peut pas être mise à jour

### Détail des messages

Les messages ci-dessous sont en relation avec une ou plusieurs des erreurs ci-dessus.

|  |  |  |
| --- | --- | --- |
| **Contexte** | **Code** | **Description** |
| SOCK\_OPEN\_URL | SOCK\_SERVER\_CONNECTION | Cause retourné par le socket #$ERRNO : $ERRSTR  [ $SERVER : $PORT ] |
|  |  |  |

## Contenu Obsolète

Certaines fonctions voir certains fichiers deviennent obsolète au fur-et-a-mesure des développements. Voici la liste du code obsolète en passe d’être supprimé.

|  |  |  |  |
| --- | --- | --- | --- |
| Fichier obsolète | API Concernée | Nouveau fichier | Nouvelle API |
| wfw/php/error.php | Fichier | **wfw/php/class/base/cResult.php** | Fichier |
|  |  |  |  |

# Application

Webframework propose une implémentation générique de l’application. Il est recommandé d’utiliser et détendre la classe **cApplication** définit dans la librairie **Webframework**. **cApplication** implémente l’interface **iApplication** est permet au programmeur une base solide à son application web.

La classe **cApplication** permet la gestion de:

* Paramètres de configuration
* Génération de templates assistée
* Gestion des erreurs
* Interface générique avec la base de données
* Dictionnaire de données

## Recommandation et 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 (**MCD/UML**).
2. Les fonctionnalités ne doivent pas être dépendante des vues, ainsi, si certaines fonctions écrites en **JavaScript** sont nécessaire à l’utilisation du module, elles doivent être incluses dans les fichiers de librairies et non directement dans les vues **HTML**. Un programmeur utilisera rarement les vues proposer en exemple, 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**

## Architecture

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

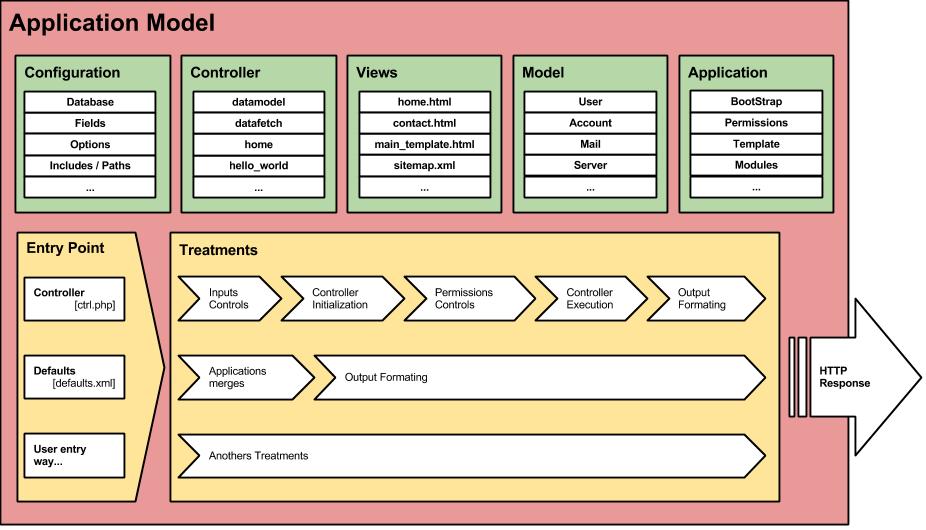
L’Application doit implémenter les interfaces suivantes :

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

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

* cApplication Application générique
* cDataBasePostgres Interface avec la Base de données **PostgreSQL 8**

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



## Paramètres de configuration

L’application est modifiable depuis le fichier de configuration.

|  |  |  |  |
| --- | --- | --- | --- |
| [APPLICATION] | Requis | Définition | Exemple |
| main\_template | **NON** | Définit le nom du fichier Template utilisé par défaut.  Remplace le paramètre $template\_file de la méthode Application::makeXMLView | "view/template.html" |
| [WINDOWS] | **Requis** | **Définition** (Windows uniquement) | **Valeurs possibles** |
| taskmgr\_class | **OUI** | Nom de la classe PHP à utiliser pour gérer le gestionnaire de tâche. | cSchTasksMgr |
| [LINUX] | **Requis** | **Définition** (Linux uniquement) | **Valeurs possibles** |
| taskmgr\_class | **OUI** | Nom de la classe PHP à utiliser pour gérer le gestionnaire de tâche. | cCronTasksMgr |

Paramètres du gestionnaire de tâches cSchTasksMgr (Windows uniquement):

|  |  |  |  |
| --- | --- | --- | --- |
| [SCHTASKS] | Requis | Définition | Exemple |
| user | **NON** | Nom d’utilisateur propriétaire. | system |
| pwd | **NON** | Mot-de-passe du compte utilisateur. |  |

## Configuration

Les données de configuration sont basées sur des fichiers au format « .ini ». Ses données permettent de paramétrer le comportement des fonctions et de l’application.

La configuration permet principalement :

* Définir les chemins d’accès aux librairies associées
* Définir les inclusions globales
* Définir les paramètres de connexion à la base de données
* Définir le dictionnaire de données
* Définir les paramètres de l’application

La configuration est un élément important de l’application, elle permet au développeur de modifier l’environnement d’exécution et le comportement de l’application sans modifier le code.

Depuis la révision « **174** » Webframework a introduit une gestion avancée des fichiers .ini. Reportez-vous au chapitre <Format de fichier INI avancé > pour plus d’informations.

### Arborescence des fichiers

Pour faciliter l’intégration de configuration, Webframework recommande de sectionner la configuration d’une application en plusieurs fichiers:

|  |  |  |
| --- | --- | --- |
| 1 | cfg/config.ini | Configuration principale (database, templates, …).  Cette configuration concerne uniquement l’application locale et ne doit pas être incluse par une autre application.  [Inclus l’ensemble des fichiers suivants] |
| 2 | cfg/all.ini | Configuration exportable de l’application.  Ce fichier se contente d’inclure l’ensemble des fichiers exportables de l’application.  [Inclus l’ensemble des fichiers suivants] |
| 3 | cfg/fields.ini | Définit le dictionnaire de données.  Tous les champs d’entrées et du model de données sont définis ici. |
| 4 | cfg/options.ini | Définit les options spécifiques à l’application.  Le nom de section doit correspondre au nom de votre application convertie au format d’un identificateur, par exemple : [my\_application] |
| 5 | cfg/sql.init | Définit les fichiers d’installation SQL (tables, fonctions, jeu d’essai, …) |

Vous trouverez un exemple complet d’intégration dans le répertoire Webframework/wfw/minimal proposant une base générique d’application.

### Dictionnaire de données (fields.ini)

Le fichier fields.ini stock le dictionnaire de données dans une unique section: [fields\_formats].

Chaque champ est identifié par un nom et un format, exemple :

birth\_date = date

Pour savoir quel sont les types supportés, reportez-vous à la section <Format de champ>.

L’utilisation d’un unique nom de section permet de fusionner plusieurs listes de champs et ainsi de former un unique dictionnaire de données.

Il est impératif de prévenir tout conflit de noms et de types (voir convention de nommage).

La plupart des champs définit, sont un copié-collé des colonnes de table de la base de données. Il est donc aisé grâce à des logiciels de modélisation de maintenir un modèle de données cohérant et générer le dictionnaire de données.

### Fichiers SQL (sql.ini)

Les fichiers SQL permettent aux applications hôtes d’inclure les dépendances de script pour configurer la base de données. Cette configuration inclue plusieurs sections :

[sql\_tables]

Correspond généralement au fichier sql/tables.sql généré par le model de données. Ce fichier crée les tables, domaines et types SQL.

[sql\_func]

Correspond généralement au fichier sql/func.sql maintenu par le développeur. Ce fichier crée les procédures stockées.

[sql\_init]

Correspond généralement au fichier sql/ini.sql maintenu par le développeur. Ce fichier initialise divers modification sur les objets avant utilisation (insertions de données, modification de contraintes, etc…).

[sql\_populate]

Correspond généralement au fichier sql/jeu\_essai.sql maintenu par le développeur. Ce fichier insert un jeu d’essai global à la BDD permettant de réaliser des tests de fonctionnement.

[sql\_remove]

Correspond généralement au fichier sql/remove.sql maintenu par le développeur. Ce fichier permet de supprimer l’ensemble des objets et insertions du module sans affecter le reste de la BDD.

## Model de données

Le model de données est une des composantes les plus importantes de votre application :

1. Il définit sous forme de dictionnaires tous les champs utilisés dans les requêtes et les tables de votre base de données
2. Il définit sous forme de classes toutes les entités de votre application

Un champ définit :

* Un Identifiant
* Une Description
* Un Format

Chaque champ est unique et possède un usage spécifique. **Webframework** en définit certains (voir **Champs réservés par Webframework**). Les modules comme toutes applications, en définissent aussi, pour plus d’informations reportez-vous à la documentation concernée.

### Format de champ

Le format, test la validité d’une chaine de caractères. C’est une composante de sécurité importante car s’est-elle qui filtre les champs reçu par l’utilisateur avant leurs utilisations.

Par convention, votre application définit les formats associés à vos champs dans la section [fields\_formats] du document cfg/fileds.ini , comme dans l’exemple suivant :

[fields\_formats]

field\_name = format

contact\_mail = mail ; Une adresse électronique

Le contrôleur fera ensuite la relation avec la classe de base cInput correspondante en ajoutant le préfix cInput. Par exemple le format **mail** utilisera la classe cInputMail et le format **identifier** utilisera la classe cInputIdentifier, ainsi de suite… Vous pouvez donc aisément ajouter vos propres formats en définissant de nouvelles classes étendant la base cInput.

### Description de champ

Chaque est décrit par une courte description, cette initiative permet de communiquer plus facilement avec un utilisateur lambda. Ce texte est également utilisé dans la génération de formulaires et les résultats de procédures.

Chaque texte peut être écrit dans différents langages, dans le cas ou votre application doit être multi-langage. Le langage par défaut est définit par un paramètre de configuration (voir chapitre Application).

Les textes sont définit dans le fichier default.xml à la racine de votre application :

<results lang="fr">

<fields>

<user\_account\_id>Nom d'utilisateur</user\_account\_id>

<user\_pwd>Mot-de-passe</user\_pwd>

<user\_mail>Adresse eMail</user\_mail>

<token>Jeton</token>

<life\_time>Temps d'inactivité avant déconnexion</life\_time>

<cid>Identifiant de connexion</cid>

<firstname>Prénom</firstname>

<lastname>Nom</lastname>

<birthday>Date de naissance</birthday>

</fields>

</results>

### Champs réservés par Webframework

Webframework se réserve une partie du dictionnaire de données pour son fonctionnement interne. Il est donc déconseillé d’utiliser ces identifiants dans un contexte différent de celui définit ci-dessous.

Résultat de procédure :

|  |  |  |
| --- | --- | --- |
| Identifiant | Description | Format |
| result | Contexte de résultat | Identifier |
| error | Code de l’erreur | Identifier |
| message | Message de l’erreur | Identifier |
| txt\_result | Texte du contexte de résultat | String |
| txt\_error | Texte de l’erreur | String |
| txt\_message | Texte du message de l’erreur | String |

Model-Vue-Contrôleur :

|  |  |  |
| --- | --- | --- |
| Identifiant | Description | Format |
| app | Nom de l’application | Identifier |
| ctrl | Nom du contrôleur | Identifier |
| output | Format de sortie du document | Identifier |

### Modélisation des données

Il est vivement recommandé d’utiliser un logiciel spécialisé pour modéliser votre modèle de données.

Cela apporte plusieurs avantages :

1. Une modélisation intuitive des entités
2. Exportation du model de données (scripts SQL)
3. Exportation du model orienté objet (classes PHP, C++, Java, etc…)
4. Maintenance facilitée
5. Gain de temps

#### PowerAMC (Sybase)

Les applications tierces de Webframework ont été construites avec le logiciel **PowerAMC** de **Sybase**.

Si vous êtes familier de ce logiciel vous pourrez utiliser les extensions suivantes pour faciliter l’exportation de vos données :

|  |  |
| --- | --- |
| Webframework/documents/sybase/php-wfw.xol | Extension de langage pour PHP. Permet de générer les classes d’entité et les fonctions d’interaction avec la base de données. |
| Webframework/documents/sybase/pgsql8-wfw.xdb | Extension SGBD pour PostgreSQL-8 |

Reportez-vous au fichier Webframework/documents/sybase/readme.md pour savoir comment installer cette extension dans **PowerAMC**.

## Vues

…

### Paramètres de configuration

Paramètres modifiable depuis le fichier de configuration.

|  |  |  |  |
| --- | --- | --- | --- |
| [APPLICATION] | Requis | Définition | Exemple |
| main\_template | **OUI** | Chemin d’accès au Template principale | crtl |

### Bundle

Les bundles offre un système simple pour inclure des sous-templates à une vue.

…

[Usage]

[Exemple]

## Contrôleurs

L’implémentation de contrôleurs se fait simplement par l’écriture d’un classe cApplicationCtrl.

Par défaut, les contrôleurs sont placés dans le dossier /ctrl de votre application. Cependant, le paramètre de configuration [*application\_name* : ctrl\_path] définissant ce chemin peut être changé.

L’utilisation de contrôleur apporte divers avantages :

* Contrôle automatique des champs en entrées
* Choix des sources de données ( $GET, $POST, $COOKIES, User Array, … )
* Fusion des champs dans un unique objet StdClass
* Exécution depuis une application tierce (module)

Note d’implémentation :

* Le format change en type mime (ex: html devient text/html)
* Chaque contrôleur est implémenté dans un espace de nom propre (ex: namespace application/my\_use\_case; )
* Les classes du framework sont définit dans l'espace de nom \webframework

### Exemple d’implémentation

namespace application\my\_use\_case;

use \webframework; // Pour les classes de bases

class Ctrl extends cApplicationCtrl {

…

}

### Paramètres de configuration

Paramètres modifiable depuis le fichier de configuration.

|  |  |  |  |
| --- | --- | --- | --- |
| [APPLICATION] | Requis | Définition | Exemple |
| ctrl\_path | **OUI** | Chemin d’accès aux contrôleurss | crtl |

### Exemple

**[ctrl/secret.php]**

Implémente le contrôleur

class **Ctrl** extends **cApplicationCtrl** {

public $fields = null; // Identifiants des champs requis

public $op\_fields = null; // Identifiants des champs optionnels

public $att = null; // Source des champs en entrée. Si NULL, $\_REQUEST est utilisé

function main(

iApplication $app, // Main Application interface

$app\_path, // Path to owner application

$p // StdClass of parameters

)

{

If( $p->secret\_word != “hello world” )

return RESULT( cResult::Failed, “SORRY\_IS\_NOT\_THIS\_WORD” );

return RESULT\_OK() ;

}

function output( iApplication $app, $format, $att, $result ) {

switch( $format ){

case “text/plain”:

echo(“My output text…”);

return true;

default:

return parent::output($app, $format, $att, $result);

}

}

};

**[cfg/fields.php]**

Définit le format au champ

[fields\_formats]

secret\_word = string

**[default.xml]**

Définit la traduction du champ

<results lang="fr">

<fields>

<secret\_word>Mot secret</secret\_word>

</fields>

</results>

## BootStrap

Le **bootStrap** est le premier fichier qui doit être inclue dans tous les points d’entrées de votre application.

Son rôle consiste à initialiser la classe Application et permettre ainsi de l’exploiter. Ce fichier se trouve initialement dans le répertoire inc/global.php. Son emplacement est important via à vis du fichier de configuration (chargé relativement).

Voici la façon dont s’initialise l’application :

* Le fichier de configuration est chargé
* Le script récupère le chemin d’accès vers la librairie Webframework et le place dans les chemins d’inclusions PHP
* La librairie minimal est inclue
* L’instance globale de l’application est initialisé dans la variable $app

## Permissions

Tous les contrôleurs de l’application font l’objet d’un contrôle de permission avant leurs exécutions. Ce contrôle permet de différencier le rôle des utilisateurs et d’en restreindre leurs possibilités.

### Un système simple et personnalisable

* Une permission est basée sur un masque de bits
* Chaque bit représente un rôle
* Une permission peut combiner un ou plusieurs rôles

Ainsi une permission fait office de filtre avec l’exécution du contrôleur mais aussi d’identificateur de rôle.

### Implémentation

1. On définit les rôles
2. On assigne une permission aux contrôleurs
3. On écrit le contrôleur suivant le type de rôle rencontré

### Exemple

Nous définissions 2 rôles pour notre application : **Administrateur** et **Utilisateur**

class Application extends cApplication {

//roles

const AnyRole = 0xffffffff;

const AdminRole = 0x10000000;

const UserRole = 0x20000000;

…

}

Notre application différenciera un **Administrateur** d’un **Utilisateur** par une variable de cookie nommée is\_admin. Si cette variable est présente nous en déduirons que l’utilisateur est un administrateur et qu’il dispose de permissions privilégié.

(Attention : Il est fortement déconseillé d’utilisé ce type de vérification basé sur un simple cookie pour votre site Internet. En effet, cette sécurité est bien trop simple à contourné pour un utilisateur mal veillant, mais pour notre exemple cela suffira…)

class Application extends cApplication {

/\*\*

\* @brief Détermine le rôle de l'utilisateur en cours

\*/

function determinateRoles(){

// par défaut : Utilisateur

$role = Application::UserRole;

// c’est un Admin ?

if( isset($\_COOKIE["is\_admin"])){

$role |= Application::AdminRole;

}

//redéfinit le rôle global en cours

$this->setRole($role);

}

}

Maintenant et lors de l’exécution de notre contrôleur (si il a lieur) il nous suffira de test le type de rôle rencontré pour guider notre action… Dans notre cas il s’agit simplement de retourné "Hello World" dans le cas d’un **utilisateur** et "Hello World Admin !" dans le cas d’un **administrateur**.

class application\_hello\_world\_ctrl extends cApplicationCtrl {

// Nous acceptons seulement deux types de rôles (Admin et User), les autres

// ne pourrons pas exécuter ce contrôleur et recevront un beau message d’erreur !

function acceptedRole() {

return Application::UserRole | Application::AdminRole;

}

// Affiche un message de bienvenue

function output(iApplication $app, $format, $att, $result)

{

$output = "Hello World";

// Admin ?

if( $this->hasRole() & Application::AdminRole )

{

$output .= " Admin !";

}

return $output;

}

Exécutez le contrôleur hello\_world avec l’URI suivante \_\_ et vous obtenez :

[image ecran…]

Maintenant, ajoutez le paramètre is\_admin aux cookies….

[image ecran…]

Le contrôle à bien eu lieu.

***Cet exemple est disponible en test dans le pack d’exemples.***

## Chemins d’accès

Ce chapitre détail les bonnes pratiques d’utilisations des chemins d’accès dans votre application.

### Répertoire de travail (PATH)

Le répertoire de travail correspond à l’emplacement des points d’entrées de l’application (généralement le fichier index.php ou ctrl.php). Il sagit du répertoire de base de l’application.

Depuis la classe Application :

global $app ;

$app->getRootPath() ; // chemin absolu vers le répertoire de base de l’application

Depuis un contrôleur :

class Ctrl extends cApplicationCtrl {

function main(iApplication $app, $app\_path, $p)

{

$app\_path ; // chemin d’accès vers le répertoire de l’application qui à définit le contrôleur

}

} ;

### Hôte (URL)

Les URL d’accès à l’hôte sont définit dans le fichier default.xml. Ce travail d’édition est réalisé par le contrôleur Defaults ce qui évite la pénible tâche de le faire à la main.

Cependant, si vous souhaitez définir explicitement ces valeurs vous devrez éditer le fichier default.xml avec les valeurs suivantes.

<site>

<host id="***HOST\_NAME***">

<domain>***localhost***</domain>

<path>***APP/MAIN***</path>

<base\_path>***APP***</base\_path>

</host>

</site>

### Pages en lignes (URL)

La gestion des chemins d’accès est automatisée via la configuration et le fichier default.xml.

Chaque page de l’application est référencé dans le fichier default.xml.

La technique consiste à associer un **identifiant** à chacune des adresses (**URL)**. L’Identifiant est ensuite réutilisé dans les **Template XML** via le marqueur de texte default. Ansi, il est possible de modifier les **URI** des pages sans apporter de modifications aux **Templates XML**.

1. Définir le lien dans le fichier defaut.xml :

<site>

<index>

<page id="google">http://www.google.fr</page>

</index>

</site>

2. Utiliser le marqueur Default dans un **Template XML**

<html>

<body>

<a href="-{page:google}">Google Web Site</a>

</body>

</html>

## Base de données

Pour interagir avec différent types de bases de données, Webframework implémente deux classes d’interface :

* iDatabase : Permet de se connecter au serveur SGBD
* iDatabaseQuery : Permet de manipuler le résultat des requêtes SQL

Pour le moment un seul SGBD est fourni avec la librairie, il s’agit de PostgreSQL-8+. C’est pour le moment le SGBD de prédilection pour toutes applications utilisant Webframework.

### Paramètres de configuration

La base de données est modifiable depuis le fichier de configuration.

|  |  |  |  |
| --- | --- | --- | --- |
| [DATABASE] | Requis | Définition | Exemple |
| class | **OUI** | Nom de la classe PHP à utiliser | cDataBasePostgres |
| server | **OUI** | Adresse du serveur | 127.0.0.1 |
| type | **OUI** | Nom du type de SGBD | PostgreSQL |
| user | **OUI** | Nom du compte utilisateur | toto |
| pwd | **OUI** | Mot-de-passe de connexion | 65q78812d |
| name | **OUI** | Nom de la base de données | my\_site\_web |
| port | **NON** | Numéro de port | 5432 |
| schema | **NON** | Schéma à utiliser | public |

## Résultat de procédure

La classe cApplication utilise le système de résultat de procédure pour :

* Traduire les messages et codes d’erreurs

## Template

La création de templates peut se faire de façon simple avec les méthodes suivantes :

cApplication::showXMLView

cApplication::makeXMLView

cApplication::showHTMLView

cApplication::makeXMLView

# Modules

Le développement de module sous Webframework est assez simple. Lorsque vous créez un module sous Webframework, vous le développez comme une application normale.

Vous implémentez les vues, les contrôleurs et le modèle de données. Cette application pourra être ensuite fusionnée aisément via le système de configuration.

Avantages :

* Pas de nouveaux concepts à apprendre vis-à-vis d’une application ordinaire
* Application indépendante (test unitaires, vues, …)

## 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.

## Point d’entrée de l’application

Conformément au projet **Webframework**, ce projet implémente l’interface IModule.

Vous trouverez le code de la classe MailModule dans le dossier « wfw-1.x/lib » avec l’ensemble des classes du modèle, vous devez inclure le fichier « lib/mail/mail.php » à votre application pour pouvoir utiliser les fonctionnalités du module **Mailing**.

## API Références

La documentation du code source est disponible dans le dossier « doc », il contient une arborescence web (HTML) généré à partir du code source.

Les fichiers de projet **Doxygen** est situé dans le dossier «  doxygen ».

## Intégration Architecture

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

## Intégration MVC

Le projet est construit sur le modèle <**Modèle-Vue-Contrôleur**>.

Les objets sont stockées dans le dossier « wfw-1.x/lib **»**.

Les vues sont stockées dans le dossier « wfw-1.x/view »

Les contrôleurs sont stockées à la racine du dossier « wfw-1.x/ctrl**».**

* Définir chaque pages dans le fichier default.xml (préfixé l’identifiant du nom du module pour éviter les conflits ex : <page id="user\_create" role="administrator" name="Créer un utilisateur">user.php?page=create</page>**)**
* **Intégrer les fonctionnalités dans un unique contrôleur principale à la racine du site. Le nom du module est utilisé comme nom de fichier (ex : user.php). Les sous-contrôleurs sont placés dans le dossier <*ctrl/nom\_de\_module/\**>, ils sont inclues via le contrôleur principale.**

**Toujours utiliser un dossier intermédiaire nommé avec le nom du module dans les dossiers standards (*view, lib, ctrl, bin*)**

## Intégration Configuration

Chaque application possède son propre jeu de configuration (dictionnaire de données, paramètres d’exécution, etc…). Muter une application de module avec une autre application consiste à copier l’ensemble de sa configuration dans l’application principale.

Pour faciliter l’intégration, chacun des modules de Webframework ,suit les recommandations de configuration définit dans la section : Arborescence des fichiers. Vous pouvez donc vous baser sur une arborescence commune et des noms de constantes génériques.

Prenons un exemple, pour inclure la configuration du module utilisateur :

1. Définir le chemin d’accès à l’application :

@global user\_module\_path =   "../../Webframework-User-Module/wfw"

Dans le cas d’un module est important d’utiliser exactement le même nom de constante que celle définit par le module. Elle sera réutilisée par les différents fichiers de configuration.

1. Inclure la configuration :

@include "${user\_module\_path}/cfg/all.ini"

Notez l’usage de la constante ${user\_module\_path} dans la balise include.

Le fichier all.ini est une recommandation de nommage utilisé par tous les modules, il permet d’inclure le dictionnaire de données, les options locales et les définitions SQL.

1. Redéfinir les paramètres

Une bonne pratique consiste à inclure l’ensemble de la configuration d’un module en tête de fichier puis ensuite de redéfinir uniquement les paramètres qui nous intéressent.

@global user\_module\_path =   "../../Webframework-User-Module/wfw"

@include "${user\_module\_path}/cfg/all.ini"

[user\_module]

use\_client\_module = true

# Notes

## Windows spécifique

* **Le chemin d’accès vers le fichier PHP.EXE** doit être définit dans la variable système **PATH**.

# FAQ

## DOMDocument::loadHTMLFile et CDATA

Problème

Dans certaines conditions la méthode **PHP** DOMDocument::loadHTMLFile peut produire l’erreur **Javascript** suivante: **Uncaught SyntaxError: Unexpected token <** à proximité d’une balise script.

L’Erreur vient probablement du fait que la méthode loadHTMLFile encadre automatiquement avec des balises **<!CDATA** les éléments <script>. En conséquence, les balises **CDATA** (si elles ne sont pas encadrée par des commentaires) produisent une erreur de syntaxe de l’interpréteur **Javascript**.

Solution

Remplacez les éléments <script> par des éléments <link> permet d’externaliser les scripts dans des fichiers à part entière et d’éviter cette erreur.

## favicon.ico vide cause de multiples requêtes?

Problème

La page se recharge plusieurs fois.

Dans un document HTML si l’attribut href de la balise d’icône link (favicon) est vide ex : <link rel="icon" href="" type="image/png"> et/ou <link rel="shortcut icon" href="" type="image/x-icon"> cela peut causer l’actualisation de la page par le navigateur (1 fois, 2 fois voir plusieurs fois de suite).

Solution

Ne laissez pas le champ href vide, si aucune valeur n’est possible n’insérez pas de balise <link>.

# A Faire

## php/class/bases/cApplication : translateResult()

Obtenir le message automatiquement lors de la translation d'un resultat

# Bugs

## php/ini\_parse.php

Si une section est définit dans un commentaire le parsing est erroné

Exemple :

[sectionA]

Item1 ="blabla" **; Comment in [sectionA]**

Item2 ="blabla"