

2013

**Webframework 1.7**

**Document Technique**

**Thomas AUGUEY**

Table des matières

[Introduction 2](#_Toc347827880)

[Développement 3](#_Toc347827881)

[Application 3](#_Toc347827882)

[Paramètres 4](#_Toc347827883)

[Résultats de procédure 6](#_Toc347827884)

[Format de données 9](#_Toc347827885)

[Base de données 11](#_Toc347827886)

[Windows spécifique 12](#_Toc347827887)

# Introduction

**WebFrameWork** est un kit de développement orienté web. L’intérêt de ce **Framework** est de proposer une librairie de code adaptable permettant de développer des sites web dynamique.

**Webframework** n’est pas un **CMS** (Système de gestion de contenu) mais plutôt une base de développement pour développeur confirmé.

Ce document renseigne sur les techniques de programmation et les technologies utilisées pour programmer la librairie **Webframework**. La lecture de ce document est réservée aux développeurs voulant lui apporter des modifications, le guide utilisateur étant plus approprié aux développeurs « utilisateur » de la librairie.

# Développement

## Application

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

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

* Paramètres de configuration
* Génération de templates
* Gestion des codes d’erreurs
* Connexion à la base de données

Configuration

…

Templates

…

Erreurs

…

Base de données

…

## Paramètres

Voici les différentes constantes de configuration utilisées par l’API Webframework :

|  |  |  |  |
| --- | --- | --- | --- |
| Section | Nom | Plateforme | Usage |
| SCHTASKS | **USER** | **Windows** | cSchTasksMgr::create |
| Nom d’utilisateur utilisé pour programmer une tâche système. | | | |
| SCHTASKS | **PWD** | **Windows** | cSchTasksMgr::create |
| Mot-de-passe utilisé pour programmer une tâche système. | | | |
| windows | **taskmgr\_class** | **Windows** | - |
| Classe utilisé pour le gestionnaire de tâche.  Ce paramètre peut prendre l’une des valeurs suivantes :   * cSchTasksMgr (Recommandé) | | | |
| linux | **taskmgr\_class** | **Linux** | - |
| Classe utilisé pour le gestionnaire de tâche.  Ce paramètre peut prendre l’une des valeurs suivantes :   * cCronTasksMgr (Recommandé) | | | |
| application | **main\_template** | **-** | Application::makeXMLView |
| Template XML/XHTML utilisé par défaut. | | | |
| database | **class** | **-** | Application::getDB |
| Classe utilisé pour le gestionnaire de connexion.  Ce paramètre peut prendre l’une des valeurs suivantes :   * cDataBasePostgres (Recommandé) | | | |
| database | **server** | **-** | Application::getDB |
| Adresse IP du serveur. | | | |
| database | **type** | **-** | Application::getDB |
| Obselete. | | | |
| database | **user** | **-** | Application::getDB |
| Nom d’utilisateur de connexion. | | | |
| database | **pwd** | **-** | Application::getDB |
| Mot-de-passe de connexion. | | | |
| database | **name** | **-** | Application::getDB |
| Nom de la base de données. | | | |
| database | **port** | **-** | Application::makeXMLView |
| Numéro de port du serveur. | | | |
| database | **schema** | **-** | Application::makeXMLView |
| Schéma des tables et fonctions appelées. | | | |
| path | **wfw** | **-** | Application::Application |
| Chemin d’accès vers la librairie Webframework (racine du dossier). | | | |
| path | **yui** | **-** | Application::Application |
| Chemin d’accès vers la librairie Yahoo-UI (racine du dossier). | | | |
| path | **tmp** | **-** | Application::Application |
| Chemin d’accès vers le répertoire temporaire. | | | |
| path | **???** | **-** | Application::Application |
| Ajouter ici les libraires qui devront être chargées au lancement de l’application.  Note : Les chemins d’accès sont disponibles dans les templates par le préfix "\_ LIB\_PATH\_" suivit du nom de paramètre. | | | |

## Résultats de procédure

Pour identifier les erreurs, **Webframework** utilise un système procédural de gestion de résultat. Pour ce faire il existe dans l’application une classe permettant de stocker le dernier résultat en cours, il peut s’agir d’une erreur comme d’un succès.

La structure d’une classe de résultat pourrait ressembler à ceci :

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 « faire passer » 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 ou au contraire de laisser passer le résultat vers la fonction appelante.

Conventions

Valeur de retour :

Les fonctions utilisant le système de résultat doivent retourner une valeur booléenne. En effet, une procédure ne peut avoir que qu’un état final succès ou échec.

Passage de résultat

Dans cet exemple, le code d’erreur remonte de la **Fonction D** vers **Fonction A (traitement).**



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

Surcharge de résultat

Dans cet exemple, **Fonction B** surcharge le code d’erreur renvoyé par **Fonction D.**



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

## Format de données

Les formats de données sont basé sur un format d’expression régulière, ils permettent entre-autre de filtrer les champs de données avant leurs utilisations.

Identificateur

|  |  |
| --- | --- |
| Nom | Identifier |
| Description | Identificateur |
| Expression Régulière | [a-zA-Z\_]{1}[a-zA-Z0-9\_]\* |
| Exemple | my\_objectId |

Entier

|  |  |
| --- | --- |
| Nom | Entier numérique |
| Description | Nombre positif compris entre 0 et *n* |
| Expression Régulière | 0|([1-9]{1}[0-9]\*) |
| Exemple | 145 |

Nom

|  |  |
| --- | --- |
| Nom | Nom |
| Description | Un nom permet plus de liberté qu’un identifiant tout en excluant les caractères d’espacement ou caractères spéciaux. |
| Expression Régulière | [a-zA-Z\_]{1}[a-zA-Z0-9\_\-\.]\* |
| Exemple | Ceci.est-un-nom\_valide |

Mail

|  |  |
| --- | --- |
| Nom | Adresse électronique |
| Description | Adresse de courrier électronique |
| Expression Régulière | RFC-2822 |
| Exemple | hello@world.org |

Nom de fichier (UNIX)

|  |  |
| --- | --- |
| Nom | Nom de fichier |
| Description | Nom de fichier au format UNIX |
| Expression Régulière | - |
| Exemple | dummy.dat |

Mot-de-passe

|  |  |
| --- | --- |
| Nom | Mot-de-passe |
| Description | - |
| Expression Régulière | [a-zA-Z0-9\_\-\@\#\&\+\~]+ |
| Exemple | Mon#mot-De-passe\_50\_ |

Chaine de caractères

|  |  |
| --- | --- |
| Nom | Texte |
| Description | Texte compris entre crochets |
| Expression Régulière | [^"\n\r]\* |
| Exemple | Bienvenue à tous ! |

## Base de données

…

## Windows spécifique

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

## Résultats d’opération

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\_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 dialogues

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 |
|  |  |  |  |

# Guide

## Charte de qualité 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**

## 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
* 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** ».

## 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 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

* 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*)**