Webframework 1.8

Document Technique

Thomas AUGUEY

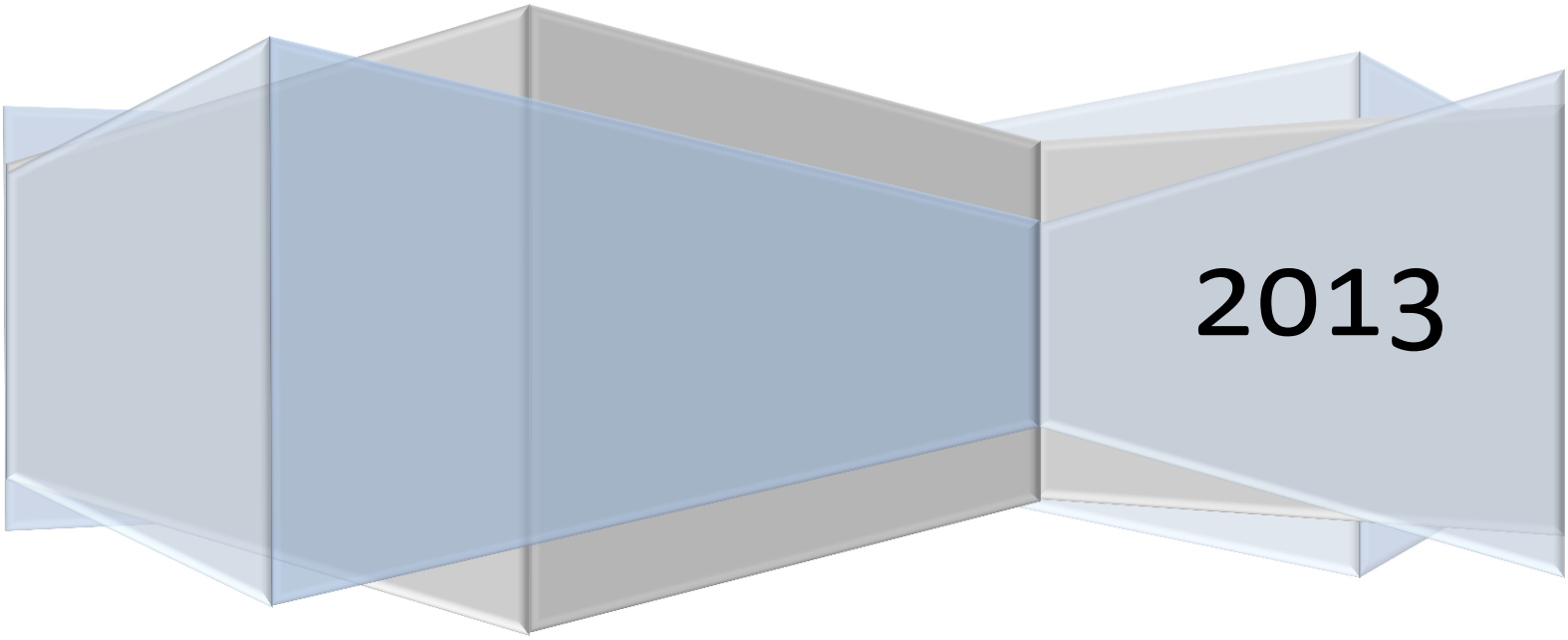


Table des matières

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 assistée
- Gestion des erreurs
- Interface générique avec la base 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.	

Model de données

Le model de données est une des composantes les plus importantes de votre application. Il définit sous forme de dictionnaires tous les champs utilisés dans les requêtes et les tables de votre base de données.

Un champ définit :

- Un Identifiant
- Une Description
- Un Format

Chaque champ est unique et possède un usage spécifique. Webframework en définit certains (voir tableau ci-dessous).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 indiqué ci-dessous :

```
[fields_formats]

field_name = format
contact_mail = mail   ; Une adresse électronique
```

Liste des formats reconnus

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]*

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 serait 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

Champs relatifs aux résultats de procédures :

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

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. Exporter le model de données (scripts SQL)
3. Exporter le 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**.

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 <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

Librairie

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.

Sections Fields

Une seule section est utilisée : **[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.

Sections SQL

Les fichiers SQL permettent aux applications hôtes d'inclure tel ou tel 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.

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

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 définit 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"
```

Doublons de sections

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"
```

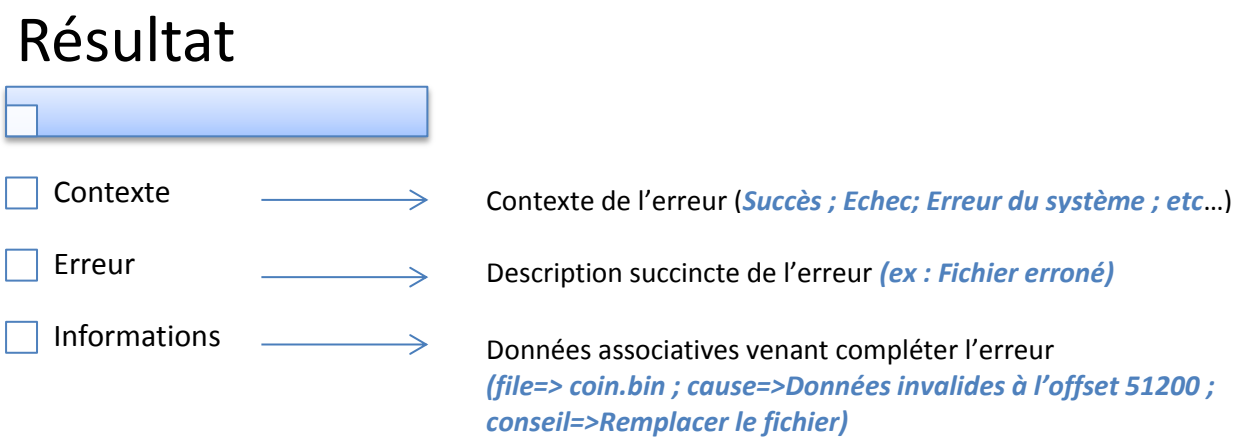
=

```
[my_section]
foo = "not bar"
bar = "foo"
smile="happiness"
```

Résultats de procédure

Pour identifier les erreurs, **Webframework** utilise un système 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 :



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.

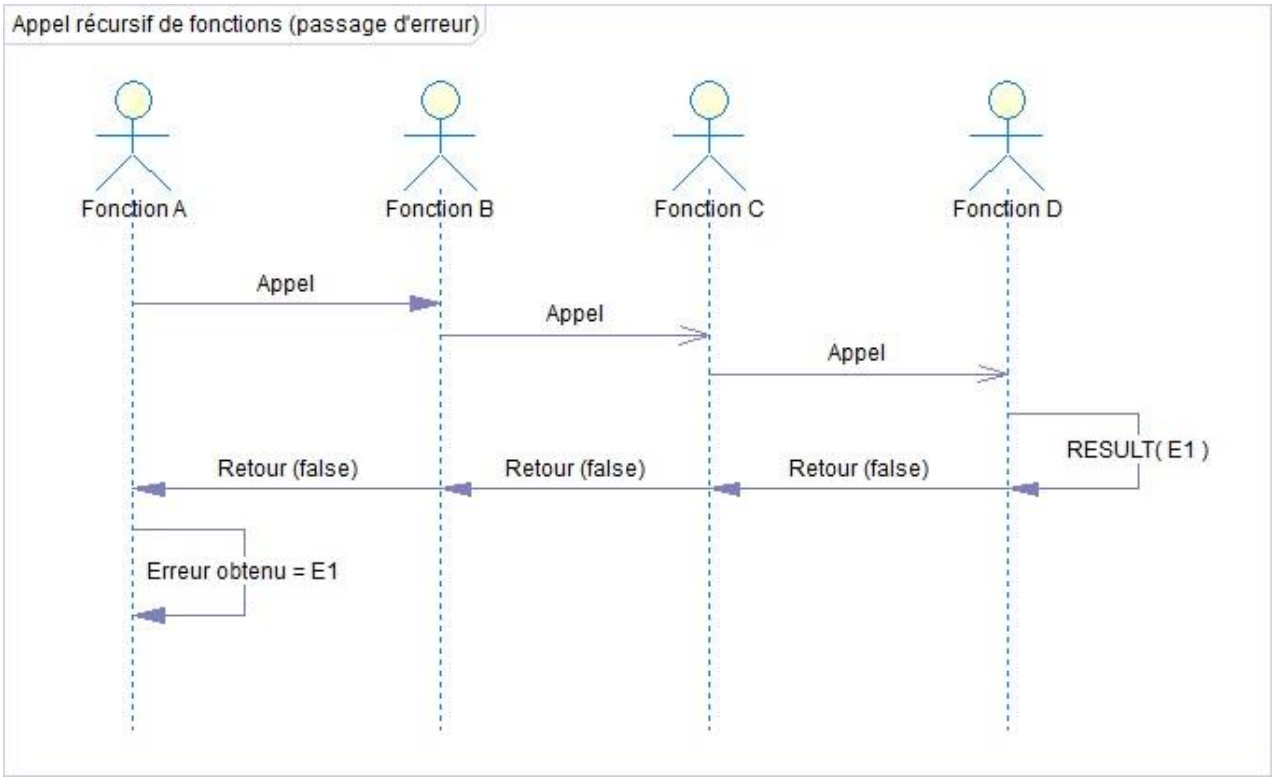
Conventions

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

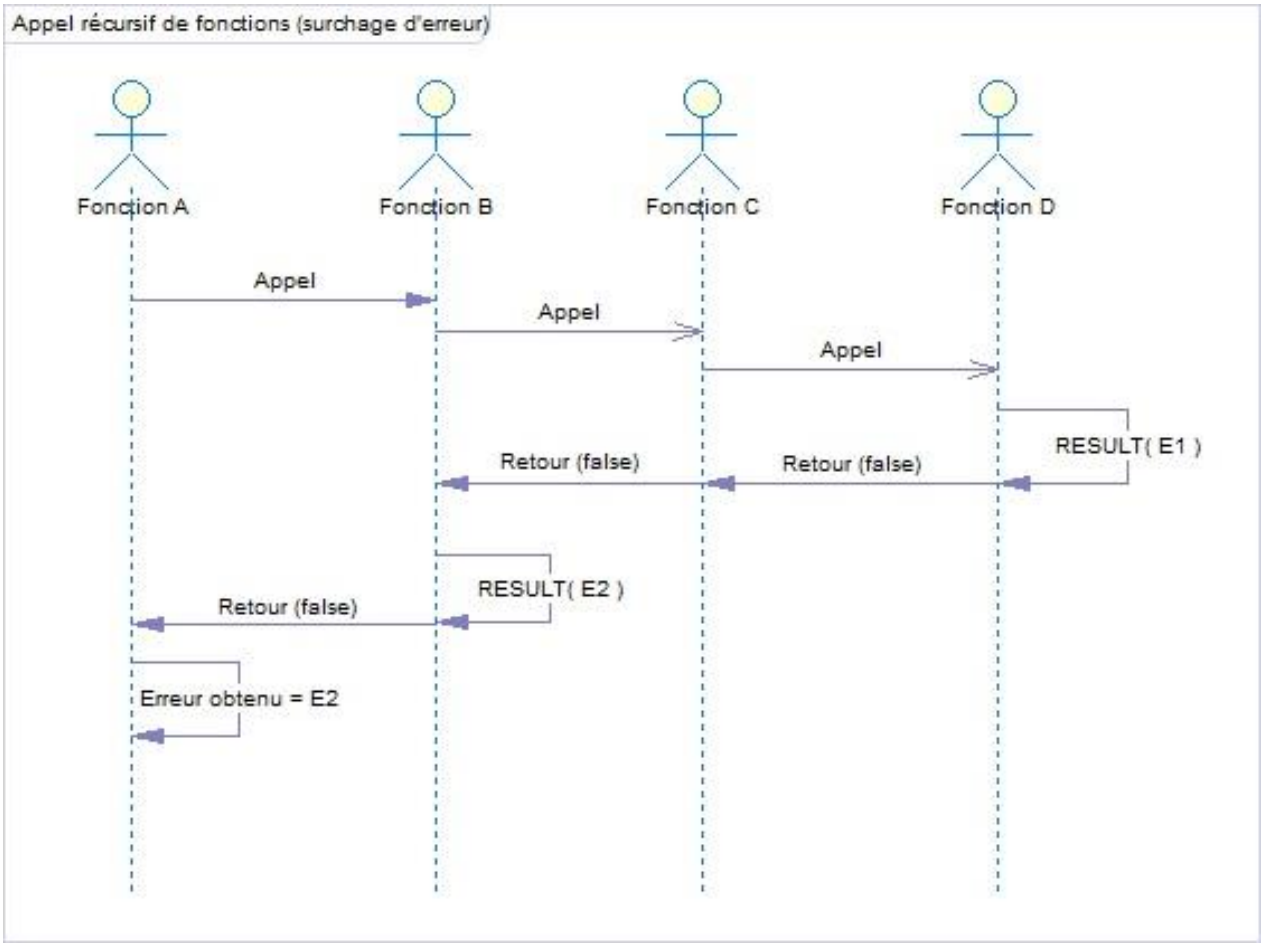
Dans cet exemple, le code d'erreur remonte de la **Fonction D** vers **Fonction A** (traitement)**.**



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*

Templates XML

…

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 indéfinit
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

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***)

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.

2. 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.

3. 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**.