2012-2013

**Webframework 1.7**

**Document Technique**

**Thomas AUGUEY**

Table des matières

[Introduction 2](#_Toc346126655)

[Résultats de procédure 3](#_Toc346126656)

[Format de données 6](#_Toc346126657)

[Base de données 8](#_Toc346126658)

# 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.

# 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.

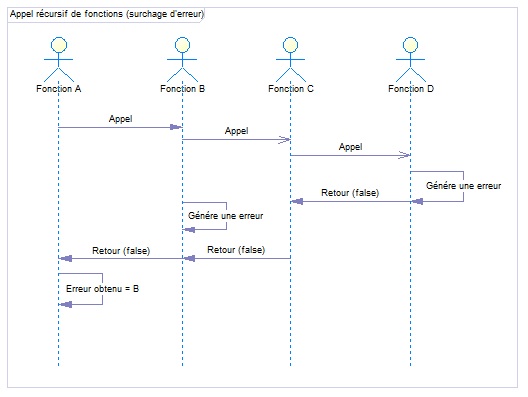
Conventions

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.

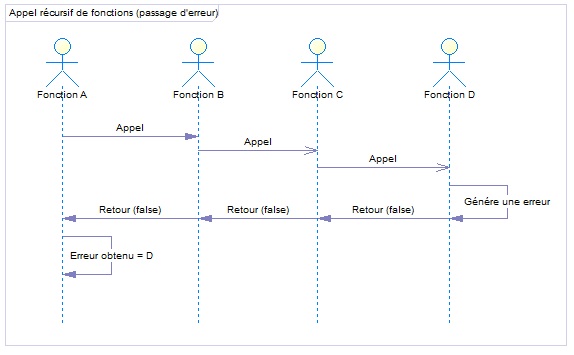
Surcharge de l’information

Dans cet exemple, **Fonction B** surcharge le code d’erreur qu’obtiendra **Fonction A :**



Passage de l’information

Ici, toutes les fonctions ne génère pas de nouveau code de résultat et laisse remonter l’information vers **Fonction A.**



# 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

…