**UniversalCsvImport**

**V2.0.0**

Table des matières

[1 Classe UniversalCsvImport 3](#_Toc5625973)

[2 Fichiers CSV 3](#_Toc5625974)

[3 Construction du modèle d’importation 3](#_Toc5625975)

[1) Tests standards 6](#_Toc5625976)

[2) Tests personnalisés 6](#_Toc5625977)

[4 Création d’une méthode d’enregistrement des donnees 7](#_Toc5625978)

[5 Création du script d’import 8](#_Toc5625979)

[3) Instanciation de notre classe et import des données 9](#_Toc5625980)

[4) Test des données 10](#_Toc5625981)

[5) Affichage des données en erreur 11](#_Toc5625982)

[6) Chargement dans la base 12](#_Toc5625983)

[6 Code complet 13](#_Toc5625984)

[7 Méthodes publiques de la classe UniversalCsvImport 14](#_Toc5625985)

# Classe UniversalCsvImport

Ce document est un tutoriel qui va vous permettre d’apprendre à utiliser la classe UniversalCsvImport destinée à aider le développeur à créer des importation de fichiers CSV pour ses applications.

Cette classe fait partie intégrante de l’outil UniversalWeb et à ce titre les exemples utilisés ici se feront sur la base de cet outil.

Pour obtenir la version de cette classe, tapez echo UniversalCsvImport::VERSION.

# Fichiers CSV

La classe permet d’importer des fichiers CSV. Ces fichiers sont des fichiers souvent issus de Microsoft Excel dont les champs sont séparés par un point-virgule (;). Il permettent de charger une base de données applicative à avec un grand nombre de données présentes sous cette forme. On parle alors d’import CSV.

Pour ce tutoriel, et pour poursuivre dans la lignée des exemples « vidéothèque » utilisé dans les documents « UniversalForm » et « UniversalList » nous allons importer un fichier CSV contenant une liste de nouveaux films. Voici le contenu de ce fichier.

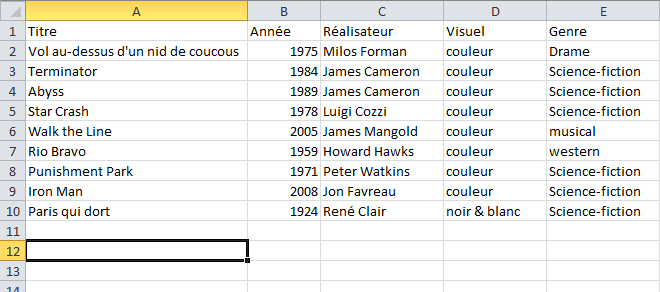


Figure 1

# Construction du modèle d’importation

Première chose nous allons avoir besoin de définir le modèle d’importation, c’est-à-dire définir pour PHP la structure du fichier CSV qui doit être importée en précisant quelle colonne doit être importée et quels sont les tests à réaliser pour valider l’importation de chacune de ces colonnes. Pour ce faire nous allons créer une classe Exemple\_csvimport que nous allons faire hériter de la classe UniversalCsvImport.

Créons donc cette classe dans le fichier associé Exemple\_csvimport.php. Puis bâtissons le modèle d’importation attendu dans une méthode publique que nous allons appeler buildModele()[[1]](#footnote-1). A l’intérieur de cette méthode, grâce à la méthode createColonne(), nous allons préciser pour chaque colonne à importer :

1 – Un identifiant de colonne qui servira à accéder à la colonne.

2 – L’indice de la colonne dans le fichier CSV.

3 – Le libellé de la colonne.

4 – Le champ SQL qui correspond à la colonne à importer.

5 – Une liste des tests à effectuer sur la colonne.

6 – Un commentaire sur la colonne (par exemple pour spécifier à l’utilisateur final la saisie attendue).

7 – Le code CSS à affecter au commentaire si celui-ci doit être affiché.

8 – Une valeur par défaut à affecter dans le cas ou le test ‘DEFAULT’ serait positionné.

9 – Un booléen qui dit si on prendre en compte (ou pas) la colonne en question.

Voici à quoi pourrait ressembler le modèle correspondant à notre fichier « vidéothèque » présenté à la Figure 1.

<?php

//----------------------------------------------------------------------

// Classe d'import CSV pour la videothèque

//----------------------------------------------------------------------

class Exemple\_csvimport extends UniversalCsvImport {

public function **buildModele**() {

$this->**createColonne**('titre', array(

'colonne' => 0,

'libelle' => 'Titre',

'sqlField' => 'titre',

'match' => array('REQUIRED'),

'commentaire' => 'Titre du film (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('annee', array(

'colonne' => 1,

'libelle' => 'Année',

'sqlField' => 'annee',

'match' => array('REQUIRED', 'CHECK\_INTEGER\_4'),

'commentaire' => 'Année de sortie du film (4 chiffres obligatoires)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('realisateur', array(

'colonne' => 2,

'libelle' => 'Réalisateur',

'sqlField' => 'realisateur',

'match' => array('DEFAULT'),

'commentaire' => 'Nom du réalisateur (saisie obligatoire)',

'defaut' => 'Inconnu',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('visuel', array(

'colonne' => 3,

'libelle' => 'Visuel',

'sqlField' => 'visuel',

'match' => array('REQUIRED', 'PARMI\_VISUEL'),

'commentaire' => 'Type de visuel (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('genre', array(

'colonne' => 4,

'libelle' => 'Genre',

'sqlField' => 'genre',

'match' => array('REQUIRED', 'PARMI\_GENRES'),

'commentaire' => 'Genre cinématographique (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

}

La méthode createColonne() contient 2 paramètres :

1 – L’identifiant donné à la colonne et qui sera réutilisé pour y accéder. Celui-ci sera utilisé dans tout le reste du code PHP pour faire référence à cette colonne de fichier CSV. Il doit dont être unique et bien entendu parlant.

2 – Un tableau associatif définissant la colonne CSV attendue. Ce tableau peut contenir les champs suivants :

|  |  |
| --- | --- |
| Nom du champ | Usage |
| colonne  integer | L’indice de la colonne dans le fichier CSV dans lequel puiser l’information de la colonne (entier). C’est le seul champ obligatoire pour spécifier une colonne de fichier CSV.  **Attention :** la première colonne du fichier est la colonne indicée 0 (zéro) et non 1. |
| libelle  string | Le libellé de la colonne qui sera utilisé dans l’application (chaîne de caractères). Il ne faut pas confondre avec le libellé d’entête présent dans le fichier CSV, ceci n’a rien à voir. Ici c’est à vous de préciser le libeller que **VOUS** voulez donner à la colonne x |
| sqlField  string | Le champ SQL qui correspond à la colonne à importer dans la base de données. Actuellement la classe UniversalCsvImport n’utilise pas cette information, mais dans un souci d’évolution ultérieur, nous pouvons d’ores et déjà préciser cette information ici. Et qui sait peut-être que vous utiliserez cette information dans votre classe Exemple\_csvimport. |
| match  array | La liste des tests caractéristiques (array) à laquelle doit répondre le contenu de la colonne pour pouvoir être importée dans votre base de données. Clairement, il s’agit d’une liste de tests prédéfinis qui pourront être exécutés par la classe sur demande afin de valider les données. Cette liste est composée d’un tableau de tests (voir ci-dessous). Noter que ce paramètre n’est pas obligatoire. S’il n’est pas présent dans les paramètre de création de la colonne, alors il ne sera réalisé aucun test sur la colonne et celle-ci sera importée telle quelle, y compris si elle est vide. Il faut donc bien réfléchir au résultat attendu par l’importation. |
| commentaire  string | Commentaire associé à la colonne CSV. C’est un commentaire libre. Il peut être utilisé pour afficher à l’utilisateur final les informations de saisie attendues et / ou le type de données attendues. |
| css  string | Classe CSS associées au commentaire dans le cas ou celui-ci serait affiché. Dans notre exemple la colonne « titre », affiche son commentaire en rouge (text-danger) pour spécifier le caractère obligatoire de la saisie. |
| defaut  string | Valeur par défaut qui sera saisie comme valeur de la colonne si celle-ci est vide et si le match ‘DEFAULT’ est spécifié. Dans notre exemple, la colonne réalisateur n’est pas obligatoire et si celle-ci est vide (non renseignée), la valeur par défaut ‘Inconnu’ sera alors affectée à la colonne. |
| active  boolean | Booléen qui spécifie si la classe doit prendre en compte la colonne CSV.  true (valeur par défaut) : la colonne est lue  false : la colonne du fichier CSV est ignorée |

## Tests standards

Voici les tests disponibles en standard :

|  |  |
| --- | --- |
| Test | Usage |
| DEFAULT | Seul test qui n’en soit pas un : Si spécifié, le contenu du champ ‘defaut’ défini lors de la création de la colonne sera transmis comme données de la colonne si celle-ci est vide et non obligatoire. |
| REQUIRED | Le contenu de la colonne à importer ne doit pas être vide. |
| NUMERIC | Le contenu de la colonne doit être du type numérique (ou vide). Les virgules sont donc acceptées. |
| NOT\_ZERO | La saisie doit être différente de 0 (ou vide). |
| CHECK\_INTEGER | Entier obligatoire (signes + et - autorisés) (ou vide). |
| CHECK\_INTEGER\_1OU2 | Entier à 1 ou 2 chiffre obligatoire (ou vide). |
| CHECK\_UNSIGNED\_INTEGER | Entier obligatoire (signe interdit) (ou vide). |
| CHECK\_SIGNED\_INTEGER | Entier obligatoire (signes + et - autorisés) (ou vide). |
| CHECK\_INTEGER\_4 | 4 chiffres obligatoires (ou vide). Par exemple pour une année. |
| CHECK\_INTEGER\_8 | 8 chiffres obligatoires (ou vide). |
| CHECK\_FLOAT | Nombre à virgule flottante avec décimales optionnelles (ou vide). |
| CHECK\_FLOAT\_2DEC | Nombre à virgule flottante avec 2 décimales optionnelles (ou vide). |
| CHECK\_BOOLEAN | 0 ou 1 (ou vide). |
| CHECK\_DATETIME | Entrée Datetime au format 0000-00-00 00:00 (Y-m-d) (les secondes sont optionnelles) (ou vide). |
| CHECK\_EMAIL | eMail (ou vide). |
| CHECK\_EMAIL\_APOSTROPHE | eMail avec apostrophe acceptée (ou vide). |
| CHECK\_ALPHA\_SIMPLE | chiffres + minuscules + espace + "-" (ou vide). |
| CHECK\_ALPHA\_NOMS | majuscules + minuscules + accents + espace (ou vide). |
| CHECK\_FILE\_NAME | Vérifie si saisie compatible avec le nommage d'un fichier (ou vide). |
| CHECK\_URL | url du style http://serveur.ext/page/ (ou vide). |
| CHECK\_IPV4 | Adresse IP V4 (eg : 192.168.1.0) (ou vide) |
| CHECK\_MAC | Adresse Mac (eg : 12:45:af:20:d8:00) (ou vide) |
| CHECK\_SHA1 | 40 caractères hexadécimaux (ou vide) |

Par exemple si l’on souhaite que le contenu d’une colonne soit un booléen non vide, il faudra utiliser les tests REQUIRED et CHECK\_BOOLEAN de telle façon :

array('**REQUIRED**', '**CHECK\_BOOLEAN**');

## Tests personnalisés

Il est possible de créer des tests personnalisés dans votre classe (ici classe Exemple\_csvimport). Dans la version actuelle, il n’est possible de créer des tests personnalisés QUE du type « PARMI », c’est-à-dire un test qui ne retienne qu’une valeur parmi une liste de valeurs.

**IMPORTANT** : Chaque test personnalisé « PARMI » doit commencer par le mot-clé « PARMI ». Ainsi pour créer un test de choix parmi plusieurs possibles, il faut écrire :

public **$PARMI**xxxxxx = **array**(choix1, choix2, choix3, etc.);

Par exemple le test personnalisé suivant :

public **$PARMI\_MF** = **array**('Masculin', 'Féminin');

Permettra de n’accepter QUE les valeurs 'Masculin' ou 'Féminin' pour la colonne désignée.

**NB** : Le mot-clé « public » est obligatoire.

Pour intégrer votre test « PARMI » il suffit alors de l’ajouter dans votre liste de tests :

array('**REQUIRED**', **'PARMI\_MF**');

Pour reprendre notre exemple, on peut donc affiner la construction de notre modèle d’importation de la sorte :

public **$PARMI\_VISUEL** = **array**('couleur', 'noir & blanc');

public **$PARMI\_GENRES** = **array**('drame', 'science-fiction', 'comédie', 'western', 'fantastique');

$this->**createColonne**('visuel', array(

'colonne' => 3,

'libelle' => 'Visuel',

'sqlField' => 'visuel',

'match' => array('REQUIRED', '**PARMI\_VISUEL**'),

'commentaire' => 'Type de visuel (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('genre', array(

'colonne' => 4,

'libelle' => 'Genre',

'sqlField' => 'genre',

'match' => array('REQUIRED', '**PARMI\_GENRES**'),

'commentaire' => 'Genre cinématographique (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

# Création d’une méthode d’enregistrement des donnees

Nous allons profiter de notre classe Exemple\_csvimport pour lui ajouter tout de suite la méthode qui sera chargée d’implanter dans la base de données les données à importer (à noter qu’il n’est pas nécessaire de la faire ici, mais c’est plus rigoureux). Libre à vous d’écrire la méthode de votre choix. Dans le cas de notre exemple, nous vous proposons d’écrire la méthode import()chargée d’insérer des lignes en masse en renouvelant les requêtes INSERT toutes les x lignes de données. Pour se faire on pourra utiliser la méthode importMany() de la classe SqlSimple. Pour un plus petit nombre de données on pourrait aussi faire appel à la méthode addMany() de la même classe SqlSimple.

Voici donc a quoi ressemble notre classe Exemple\_csvimport.

<?php

//----------------------------------------------------------------------

// Classe d'import CSV pour la videothèque

//----------------------------------------------------------------------

// classe permettant d'utiliser la classe SqlSimple

**class** SqlTableFilms **extends** SqlSimple {

public $\_table = 'films';

public $\_index = 'titre';

public $\_champs = 'titre, annee, realisateur, visuel, genre';

}

class **Exemple\_csvimport** extends **UniversalCsvImport** {

// tests personnalisés pour l'importation

public $PARMI\_VISUEL = array('couleur', 'noir & blanc');

public $PARMI\_GENRES = array('drame', 'science-fiction', 'comédie', 'western', 'fantastique');

public function **buildModele**() {

$this->**createColonne**('titre', array(

'colonne' => 0,

'libelle' => 'Titre',

'sqlField' => 'titre',

'match' => array('REQUIRED'),

'commentaire' => 'Titre du film (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('annee', array(

'colonne' => 1,

'libelle' => 'Année',

'sqlField' => 'annee',

'match' => array('REQUIRED', 'CHECK\_INTEGER\_4'),

'commentaire' => 'Année de sortie du film (4 chiffres obligatoires)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('realisateur', array(

'colonne' => 2,

'libelle' => 'Réalisateur',

'sqlField' => 'realisateur',

'match' => array('DEFAULT'),

'commentaire' => 'Nom du réalisateur (saisie obligatoire)',

'defaut' => 'Inconnu',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('visuel', array(

'colonne' => 3,

'libelle' => 'Visuel',

'sqlField' => 'visuel',

'match' => array('REQUIRED', 'PARMI\_VISUEL'),

'commentaire' => 'Type de visuel (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

$this->**createColonne**('genre', array(

'colonne' => 4,

'libelle' => 'Genre',

'sqlField' => 'genre',

'match' => array('REQUIRED', 'PARMI\_GENRES'),

'commentaire' => 'Genre cinématographique (saisie obligatoire)',

'css' => 'text-danger',

'active' => true

));

}

// importation effective dans la base de données

public function **import**($data) {

$sqlFilms = **new** SqlTableFilms();

$masqueSql = "'[0]', '[1]', '[2]', '[3]', '[4]'";

$nbInsertions = $sqlFilms->**importMany**($masqueSql, $data, 3, true);

return $nbInsertions;

}

}

# Création du script d’import

Il est maintenant tant d’écrire le code qui va faire appel à notre classe pour importer notre fichier csv de la figure 1.

Dans ce tutoriel nous n’allons pas expliquer comment créer le code permettant de sélectionner et de valider le fichier à importer, ceci relève du pure codage php et nous vous invitons à suivre l’exemple fourni qui s’appuie sur les classes UniversalForm de mise en œuvre de formulaires. Nous allons plutôt nous intéresser à la partie importation pure qui est ici traitée par la fonction import(nom\_de\_fichier). Cette fonction a pour principal but de créer un objet d’importation de la classe Exemple\_csvimport que nous venons de créer, puis d’envoyer les données en base.

**function** import($importFile) {

// creation de l'objet ImportCsV\_exemple et chargement fichier CSV

//-------------------------------------------

$csv = **new** Exemple\_csvimport();

$csv->buildModele();

$data = $csv->**charge**($importFile);

// test de chaque colonne de l'enregistrement

// les erreurs sont rapportées directement dans l'enregistrement

//-------------------------------------------

$nbErreur = 0;

**foreach** ($data **as** $numLine => $enreg) {

$erreur = $csv->**testColonnes**($data[$numLine]);

if ($erreur) {

$nbErreur++;

}

}

// affichage des erreurs

//-------------------------------------------

if ($nbErreur != 0) {

//affichage des infos non valide dans un tableau

**echo** $csv->**displayRawErrors**($data, $nbErreur);

**if** ($nbErreur == 1) {

**echo** '<p class="lead text-danger">1 ligne comporte une ou plusieurs erreurs&hellip;</p>';

}

**else** {

**echo** '<p class="lead text-danger">'.$nbErreur.' lignes comportent des erreurs&hellip;</p>';

}

**return** false;

}

//---------------------------------------------

// INSERTION DANS LA BASE DE DONNEES

//---------------------------------------------

if ($nbErreur == 0) {

**echo** '<p>Import en cours&hellip;</p>';

$res = $csv->**import**($data);

**if** ($res !== false) {

**echo** '<p class="lead text-success">Import terminé avec succès ('.$res.' nouvelles entrées).</p>';

**return** true;

}

**else** {

**echo** '<p class="lead text-danger">'.getLib('ERREUR\_SQL').'</p>';

//DEBUG\_('requete', $requete);

**return** false;

}

}

}

## Instanciation de notre classe et import des données

Tout d’abord on instancie notre classe, puis on lance la construction du modèle d’importation qui convient à notre fichier CSV et décrit dans notre classe Exemple\_csvimport.

$csv = **new** Exemple\_csvimport();

$csv->buildModele();

Puis on lance l’import du fichier grâce à la méthode charge(nom\_de\_fichier).

$data = $csv->**charge**($importFile);

$data contient alors un tableau des données chargées de la forme :

[0] => Array

(

[ligne] => 2

[titre] => Vol au-dessus d'un nid de coucous

[annee] => 1975

[realisateur] => Milos Forman

[visuel] => couleur

[genre] => drame

[erreur] =>

)

[1] => Array

(

[ligne] => 3

[titre] => Terminator

[annee] => 1984

[realisateur] => James Cameron

[visuel] => couleur

[genre] => science-fiction

[erreur] =>

)

Chaque colonne importée a pour indice le nom de la colonne fourni lors de la création de notre classe via la méthode createColonne(). La manipulation des données est alors très simple et permet d’effectuer facilement des tests sur chaque cellule du fichier CSV. Pour accéder au réalisateur du film de la deuxième ligne il suffit d’écrire $data[1][ 'realisateur'].

**NB** : Avez-vous remarqué que chaque enregistrement de données contient aussi un champ **ligne** que nous n’avons jamais spécifié ? Et bien celui-ci a été ajouté automatiquement et il contient le numéro de la ligne dans le fichier CSV où se situe l’information importée. C’est très pratique pour renvoyer l’utilisateur final à la ligne exacte dans le fichier CSV en cas d’erreur !

## Test des données

Comme nous l’avons vu lors de la création de notre modèle d’importation, nous avons placé une batterie de tests pour chaque colonne. Maintenant que nous avons chargé notre fichier CSV en mémoire, nous pouvons appliquer ces tests définis. Pour ce faire il suffit d’appeler la méthode testColonnes() pour chacune des lignes importées.

**foreach** ($data **as** $numLine => $enreg) {

$erreur = $csv->**testColonnes**($data[$numLine]);

}

On pourra comptabiliser le nombre de ligne en défaut par l’ajout d’un simple compteur :

$nbErreur = 0;

**foreach** ($data **as** $numLine => $enreg) {

$erreur = $csv->**testColonnes**($data[$numLine]);

if ($erreur) {

$nbErreur++;

}

}

Tous les tests sur toutes les colonnes de chaque ligne sont ainsi exécutés, y compris les tests personnalisés, sur l’ensemble du fichier CSV. En cas d’erreur sur une des colonnes de notre enregistrement, le type d’erreur est affiché en clair (et dans la langue en cours) dans le jeu de données au regard du test demandé.

[5] => Array

(

[ligne] => 7

[titre] => Rio Bravo

[annee] => 1959

[realisateur] => Howard Hawks

[visuel] => couleur

[genre] => Ce champ ne doit pas être vide

[erreur] => **1**

)

Dans notre exemple, si le champ « genre » n’a pas été renseigné dans la 6ème ligne du fichier csv, notre jeu de données affichera en rouge ‘*Ce champ ne doit pas être vide*’. Peut-être avez-vous remarqué la présence du champ ‘erreur’ dans chaque enregistrement importé. Ce champ est automatiquement ajouté (à la fin de l’enregistrement) pour chaque champ importé. En cas d’une ou de plusieurs erreurs lors du test de l’enregistrement, sa valeur passe à 1 (true).

## Affichage des données en erreur

Vous pouvez afficher le jeu de données comme bon vous semble (vous en avez l’accès complet). Cependant, il existe trois méthodes bien pratiques pour faire ce travail à votre place :

**echo** $csv->**displayRawErrors**($data, $nbErreur);

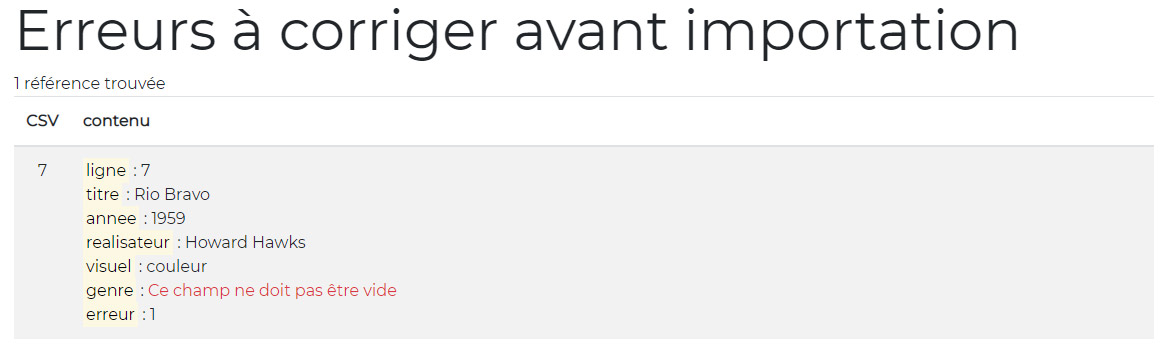
Avec en paramètre, le jeu de donnees et le nombre total d’erreurs rencontrées, la méthode displayRawErrors() affichera les enregistrement en erreur de la manière la plus simple qui soit :



Si vous souhaitez plus de style (compatible Bootstrap 4.3.x) dans l’affichage des enregistrements en erreur, vous pouvez à la place utiliser la méthode displayErrors(). Les paramètres restent identiques à la méthode displayRawErrors().

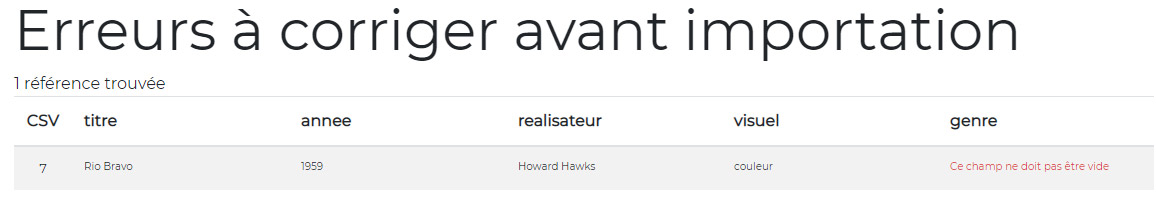
**echo** $csv->**displayErrors**($data, $nbErreur);

Le résultat obtenu est le suivant :



Enfin la troisième méthode, displayErrorsTab(), affiche chaque enregistrement en erreur dans un tableau compatible Boostrap 4.3.x. Les paramètres restent les mêmes.

**echo** $csv->**displayErrorsTab**($data, $nbErreur);



Bien entendu, si aucun de ces affichage ne vous convient, libre à vois d’écrire une nouvelle méthode pour votre classe (ici Exemple\_csvimport) qui fera le travail souhaité.

## Chargement dans la base

Si l’import ne fait plus apparaître d’erreur, vous pouvez alors importer votre jeu de données dans la base en faisant appel à votre propre méthode vue au chapitre 4.

$res = $csv->**import**($data);

# Code complet

Voici un exemple de code complet pour notre import de films …

**function** import($importFile) {

**if** (**file\_exists**($importFile)) {

// creation de l'objet ImportCsV\_exemple et chargement fichier CSV

//-------------------------------------------

$csv = **new** Exemple\_csvimport();

$csv->**buildModele**();

$data = $csv->**charge**($importFile);

// affinage des données avant traitement

//-------------------------------------------

**foreach** ($data as $numLine => $enreg) {

//on transforme les champs visuel et genre en minuscules

$data[$numLine]['visuel'] = **utf8\_strtolower**($data[$numLine]['visuel']);

$data[$numLine]['genre'] = **utf8\_strtolower**($data[$numLine]['genre']);

}

// test de chaque colonne de l'enregistrement

// les erreurs sont rapportées directement dans l'enregistrement

//-------------------------------------------

$nbErreur = 0;

**foreach** ($data **as** $numLine => $enreg) {

$erreur = $csv->**testColonnes**($data[$numLine]);

**if** ($erreur) {

$nbErreur++;

}

}

// traitement postérieur si pas d'erreur

//-------------------------------------------

**if** ($nbErreur == 0) {

**foreach** ($data **as** $numLine => $enreg) {

//transformation couleur / noir & blanc en 1 / O pour le visuel

$data[$numLine]['visuel'] = (($data[$numLine]['visuel']) == 'couleur') ? '1' : '0';

}

}

// affichage des erreurs

//-------------------------------------------

**else** {

//affichage des infos non valide dans un tableau

**echo** $csv->**displayRawErrors**($data, $nbErreur);

**if** ($nbErreur == 1) {

**echo** '<p class="lead text-danger">1 ligne comporte une ou plusieurs erreurs&hellip;</p>';

}

**else** {

**echo** '<p class="lead text-danger">'.$nbErreur.' lignes comportent des erreurs&hellip;</p>';

}

**return** false;

}

//---------------------------------------------

// INSERTION DANS LA BASE DE DONNEES

//---------------------------------------------

**if** ($nbErreur == 0) {

**echo** '<p>Import en cours&hellip;</p>';

$res = $csv->**import**($data);

**if** ($res !== false) {

**echo** '<p class="lead text-success">Import terminé avec succès ('.$res.' entrées).</p>';

**return** true;

}

**else** {

**echo** '<p class="lead text-danger">'.getLib('ERREUR\_SQL').'</p>';

**return** false;

}

}

}

else {

**echo** '<p class="lead text-danger">Fichier d\'importation '.$importFile.' CSV inexistant !</p>';

**return** false;

}

}

# Méthodes publiques de la classe UniversalCsvImport

|  |
| --- |
| getModele() |
| Permet de récupérer le modèle d’importation du fichier CSV.  Le modèle est un tableau de colonnes contenant les informations suivantes :  -> id de la colonne  -> libellé en clair de la colonne  -> champ SQL qui correspond à la colonne à importer  -> tableau de tests à effectuer sur la colonne pour définir sa validité  -> commentaire sur la saisie attendue  -> code CSS a appliquer au commentaire  -> valeur par défaut éventuellement appliquée si le test ‘DEFAUT’ est actif  -> booléen qui détermine si la colonne doit être utilisée.  **Description**  array **getModele**()  **Liste des paramètres**  **Aucun**  **Retour**  **Tableau contenant les information du modèle d’importation. Voici le modèle généré par notre exemple :**  Array  (  [titre] => Array  (  [colonne] => 0  [libelle] => Titre  [sqlField] => titre  [match] => Array  (  [0] => REQUIRED  )  [commentaire] => Titre du film (saisie obligatoire)  [css] => text-danger  [defaut] =>  [active] => 1  )  [annee] => Array  (  [colonne] => 1  [libelle] => Année  [sqlField] => annee  [match] => Array  (  [0] => REQUIRED  [1] => CHECK\_INTEGER\_4  )  [commentaire] => Année de sortie du film (4 chiffres obligatoires)  [css] => text-danger  [defaut] =>  [active] => 1  )  [realisateur] => Array  (  [colonne] => 2  [libelle] => Réalisateur  [sqlField] => realisateur  [match] => Array  (  [0] => DEFAULT  )  [commentaire] => Nom du réalisateur (saisie obligatoire)  [css] => text-danger  [defaut] => Inconnu  [active] => 1  )  [visuel] => Array  (  [colonne] => 3  [libelle] => Visuel  [sqlField] => visuel  [match] => Array  (  [0] => REQUIRED  [1] => PARMI\_VISUEL  )  [commentaire] => Type de visuel (saisie obligatoire)  [css] => text-danger  [defaut] =>  [active] => 1  )  [genre] => Array  (  [colonne] => 4  [libelle] => Genre  [sqlField] => genre  [match] => Array  (  [0] => REQUIRED  [1] => PARMI\_GENRES  )  [commentaire] => Genre cinématographique (saisie obligatoire)  [css] => text-danger  [defaut] =>  [active] => 1  )  ) |

|  |
| --- |
| getColonne() |
| Renvoie le numéro de la colonne CSV pour la colonne id.  **Description**  integer **getColonne**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Le numéro de la colonne correspondant dans le fichier CSV.** |

|  |
| --- |
| getLibelle() |
| Renvoie le libellé de la colonne du modèle identifiée par id. Il s’agit ni plus ni moins que du libellé fourni lors de la création de la colonne du modèle.  **Description**  string **getLibelle**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Le libellé de la colonne.** |

|  |
| --- |
| getSqlField() |
| Renvoie le champ SQL associé à la colonne id.  **Description**  string **getSqlField**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Le champ SQL associé à la colonne. Dans la version actuelle cette information n’est pas utilisée.** |

|  |
| --- |
| getMatch() |
| Renvoie la liste des tests associés à la colonne id.  **Description**  array **getMatch**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Un tableau contenant la liste des identifiants de tests : ex :** REQUIRED**,** CHECK\_INTEGER\_4**.** |

|  |
| --- |
| getActive() |
| Renvoie l’état de la colonne id.  **Description**  boolean **getActive**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  true**: la colonne est active et prise en compte lors de la lecture du fichier CSV.**  false **: la colonne n’est pas active et donc non importée.** |

|  |
| --- |
| getCommentaire() |
| Renvoie le commentaire associé à la colonne id.  **Description**  string **getActive**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Le commentaire.** |

|  |
| --- |
| getCss() |
| Renvoie le code CSS associé au commentaire de la colonne id. Ce code sera utilisé lors de l’affichage du commentaire pour le styler.  **Description**  string **getCss**(**string** id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **Le code CSS.** |

|  |
| --- |
| getDefaut() |
| Renvoie la valeur par défaut du champ CSV si celui-ci est vide et si le test DEFAULT a été positionné.  **Description**  string **getDefaut(string** $id)  **Liste des paramètres**  id**: identifiant de la colonne interrogée.**  **Retour**  **La valeur par défaut.** |

|  |
| --- |
| getNbColonnes() |
| Renvoie le nombre de colonnes définissant le modèle.  **Description**  int **getNbColonnes**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le nombre de colonnes définissant le modèle.** |

|  |
| --- |
| getEntete() |
| Renvoie l’entête de l’import, un tableau contenant la correspondance colonne du modèle / colonne CSV.  **Description**  array **getEntete**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le tableau d’entête. Exemple :**  Array  (  [0] => titre  [1] => annee  [2] => realisateur  [3] => visuel  [4] => genre  ) |

|  |
| --- |
| setColonne() |
| Positionne à valeur le numéro de colonne CSV pour l’identifiant id du modèle.  **Description**  **setColonne**(**string** $id, **integer** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: numéro de la colonne dans le fichier CSV pour la colonne modèle** id**.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.** |

|  |
| --- |
| setLibelle() |
| Positionne à valeur le libellé de la colonne id du modèle.  **Description**  **setLibelle**(**string** $id, **string** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: libellé de la colonne** id **du modèle.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.** |

|  |
| --- |
| setSqlField() |
| Positionne à valeur le champ SQL de la colonne id du modèle.  **Description**  **setSqlField**(**string** $id, **string** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: Champ SQL de la colonne** id **du modèle.**  **Retour**  Aucun. Si id ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister. |

|  |
| --- |
| setMatch() |
| Positionne les tests match pour la colonne du modèle identifiée par id.  **Description**  **setMatch**(**string** $id, **array** $match)  **Liste des paramètres**  id: identifiant de la colonne à positionner.  match : liste de tests prédéfinis ou personnalisés pour la colonne.  **Retour**  Aucun. Si id ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.  Exemple qui ajoute pour la colonne ‘annee’ les tests « saisie obligatoire » et « format sur 4 chiffres »  **$csv->setMatch('annee', array('REQUIRED'));**  **$csv->setMatch('annee', array('CHECK\_INTEGER\_4'));**  Ces deux appels peuvent être combinés dans le même appel.  **$csv->setMatch('annee', array('REQUIRED', 'CHECK\_INTEGER\_4'));**  Pour annuler tous les tests sur une colonne entrez un tableau vide :  **$csv->setMatch('annee', array());** |

|  |
| --- |
| setActive() |
| Active ou désactive la colonne id du modèle. Désactiver la colonne a pour effet d’ignorer l’import de la colonne dans le fichier CSV.  **Description**  **setActive**(**string** $id, **boolean** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: booléen permettant d’activer (**true**) ou de désactiver (**false**) la colonne.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.**  **Par défaut, toutes les colonnes sont actives à leur création**.  **Les méthodes active**() **et desactive**() **ont le même effet.** |

|  |
| --- |
| setCommentaire() |
| Positionne à valeur le commentaire de la colonne id du modèle.  **Description**  **setCommentaire**(**string** $id, **string** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: commentaire de la colonne** id **du modèle.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.** |

|  |
| --- |
| setCss() |
| Positionne à valeur le code CSS de la colonne id du modèle.  **Description**  **setCss**(**string** $id, **string** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: code CSS de la colonne** id **du modèle.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.** |

|  |
| --- |
| setDefaut() |
| Positionne à valeur la valeur par défaut de la colonne id du modèle.  **Description**  **setDefaut**(**string** $id, **string** $valeur)  **Liste des paramètres**  id**: identifiant de la colonne à positionner.**  valeur**: valeur par défaut de la colonne** id **du modèle.**  **Retour**  **Aucun. Si** id **ne correspond à aucune colonne, l’application s’arrête, il n’y a pas de code d’erreur émis. Le numéro de la colonne doit obligatoirement exister.** |

|  |
| --- |
| createColonne() |
| Création d’une colonne dans le modèle d’importation.  **Description**  **createColonne**(**string** $id, **array** $donnees)  **Liste des paramètres**  id**: identifiant de la colonne à créer.**  donnees **: tableau des paramètres de construction de la colonne du modèle.**  **Retour**  **Aucun.**  Exemple  :  **$this->createColonne('realisateur', array(**  **'colonne' => 2,**  **'libelle' => 'Réalisateur',**  **'sqlField' => 'realisateur',**  **'match' => array('DEFAULT'),**  **'commentaire' => 'Nom du réalisateur (saisie obligatoire)',**  **'defaut' => 'Inconnu',**  **'css' => 'text-danger',**  **'active' => true**  **));**  **$this->createColonne('visuel', array(**  **'colonne' => 3,**  **'libelle' => 'Visuel',**  **'sqlField' => 'visuel',**  **'match' => array('REQUIRED', 'PARMI\_VISUEL'),**  **'commentaire' => 'Type de visuel (saisie obligatoire)',**  **'css' => 'text-danger',**  **'active' => true**  **));** |

|  |
| --- |
| active() |
| Clone de la méthode **setActive**(true). |

|  |
| --- |
| desactive() |
| Clone de la méthode **setActive**(false). |

|  |
| --- |
| modeleColCount() |
| Renvoie le nombre de colonnes qui définissent le modèle d’importation.  **Description**  integer **modeleColCount**()  **Liste des paramètres**  Aucun  **Retour**  **Nombre de colonnes définies dans le modèle.** |

|  |
| --- |
| charge() |
| Charge le fichier CSV au regard du modèle d’import.  **Description**  array **charge**(**string** filename)  **Liste des paramètres**  filename**: c**hemin complet du fichier CSV à charger.  **Retour**  **Tableau (construit selon le modèle prédéfini) contenant les enregistrements chargés.**  **A NOTER** :   * Les données chargées sont converties en UTF-8. * Aucun test n’est réalisé sur la validité du fichier au niveau de cette méthode. Il vous appartient de vérifier son existence par vous–même au préalable. |

|  |
| --- |
| testColonnes() |
| Test le contenu de chaque enregistrement au regard de la liste des tests définis pour chaque colonne (voir méthodes setMatch() et getMatch()).  **Description**  boolean **testColonnes**(**array** &$enregistrement)  **Liste des paramètres**  enregistrement**:** Tableau contenant un enregistrement importé (une ligne du fichier CSV).  **Retour**  true**: une ou plusieurs erreurs ont été rencontrées lors du test de l’enregistrement. En retour, chaque champs en erreur est affichés en rouge dans l’**enregistrement**.**  false**: aucune erreur n’a été rencontrée lors des tests de cet enregistrement.**  **A NOTER** :   * Il est important de noter qu’en cas d’erreur l’enregistrement est modifié pour désigner le ou les champ(s) en erreur et le type d’erreur rencontrée. Un affichage via l’une des méthodes d’affichage du jeu de données résultat permettra d’en prendre connaissance. * En cas d’erreur, le champ système intitulé erreur ajouté à l’enregistrement enregistrement prendra comme valeur 1 (true). * Tous les tests prédéfinis ainsi que les tests personnalisés sont effectués sur l’enregistrement. |

|  |
| --- |
| displayRawErrors() |
| Affichage du jeu de données importé de manière simple.  **Description**  string **displayRawErrors**(**array** $data, **int** $nbErreurs)  **Liste des paramètres**  data: Jeu de données importées.  nbErreurs**:** Nombre d’erreurs rencontrées à fournir à l’application. S’agissant d’une fonction d’affichage, il n’y a aucune vérification sur ce paramètre. Il est ici juste utilisé pour afficher en clair le nombre d’erreurs rencontrées. Si vous passez 10, il affichera 10…  **Retour**  **Chaine de caractère HTML à afficher. Exemple :** |

|  |
| --- |
| displayErrors() |
| Affichage du jeu de données importé de manière structurée.  **Description**  string **displayErrors**(**array** $data, **int** $nbErreurs)  **Liste des paramètres**  true**: une ou plusieurs erreurs ont été rencontrées lors du test de l’enregistrement. En retour, chaque champs en erreur est affichés en rouge dans l’**enregistrement**.**  false**: aucune erreur n’a été rencontrée lors des tests de cet enregistrement.**  **Retour**  **Chaine de caractère HTML à afficher. Exemple :** |

|  |
| --- |
| displayErrors() |
| Affichage du jeu de données importé sous forme de tableau.  **Description**  string **displayErrorsTab**(**array** $data, **int** $nbErreurs)  **Liste des paramètres**  true**: une ou plusieurs erreurs ont été rencontrées lors du test de l’enregistrement. En retour, chaque champs en erreur est affichés en rouge dans l’**enregistrement**.**  false**: aucune erreur n’a été rencontrée lors des tests de cet enregistrement.**  **Retour**  **Chaine de caractère HTML à afficher. Exemple :** |

1. Le nom de la méthode est laissée à votre discrétion. Vous pouvez nommer cette méthode comme bon vous semble. [↑](#footnote-ref-1)