UniversalCsvImport

V2.2.0

Table des matières

1 Classe UniversalCsvImport 4
2 Fichiers CSV 4
3 Construction du modèle d'importation 4
Tests standards 7
Tests personnalisés 8
Tests PARMI 8
Tests REGEX 9
Tests CUSTOM 9
4 Création d'une méthode d'enregistrement des données 10
5 Création du script d'import 11
Instanciation de notre classe et import des données 12
Test des données 13
Affichage des données en erreur 13
Chargement dans la base 15
6 Code complet 15
7 Méthodes publiques de la classe UniversalCsvImport 17
getModele 17
getColonne 18
getLibelle 18
getSqlField 19
getMatch 19
getMatchValue 19
getActive 19
getCommentaire 20
getCss 20
getDefaut 20
getNbColonnes 20
getEntete 21
setColonne 21
setLibelle 21
setSqlField 22
setMatch 22
setMatchValue 22
setActive 23
setCommentaire 23
setCss 23
setDefaut 24

createColonne....24
active....25
desactive....25
modeleColCount....25
charge....25
testColonnes....25
displayRawErrors....26
displayErrors....26
displayErrors....27

1 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 importations 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.`**

2 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 (;). Ils permettent de charger une base de données applicative 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.

	A	B	C	D	E
1	Titre	Année	Réalisateur	Visuel	Genre
2	Vol au-dessus d'un nid de coucous	1975	Milos Forman	couleur	Drame
3	Terminator	1984	James Cameron	couleur	Science-fiction
4	Abyss	1989	James Cameron	couleur	Science-fiction
5	Star Crash	1978	Luigi Cozzi	couleur	Science-fiction
6	Walk the Line	2005	James Mangold	couleur	musical
7	Rio Bravo	1959	Howard Hawks	couleur	western
8	Punishment Park	1971	Peter Watkins	couleur	Science-fiction
9	Iron Man	2008	Jon Favreau	couleur	Science-fiction
10	Paris qui dort	1924	René Clair	noir & blanc	Science-fiction
11					
12					
13					

Figure 1

3 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]. 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 paramètre éventuel dans le cas d'un test personnalisé.
7 – Un commentaire sur la colonne (par exemple pour spécifier à l'utilisateur final la saisie attendue).
8 – Le code CSS à affecter au commentaire si celui-ci doit être affiché.
9 – Une valeur par défaut à affecter dans le cas ou le test 'DEFAULT' serait positionné.
10 – 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',
```

[1] Le nom de la méthode est laissée à votre discrétion. Vous pouvez nommer cette méthode comme bon vous semble.

```
        '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 :

Paramètre	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.
matchValue string	Lorsque le paramètre `match` contient `REGEX` ou `CUSTOM`, donner à ce paramètre la valeur nécessaire a l'exécution du test personnalisé.
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	Valeur par défaut qui sera saisie comme valeur de la colonne si celle-ci est

string	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** : la colonne est lue (valeur par défaut) **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)
MIN_LENGTH_X	Longueur minimale de la chaine limitée à X caractères (V2.1.0)
MAX_LENGTH_X	Longueur maximale de la chaine limitée à X caractères (V2.1.0)

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');
```

Ou encore, si l'on souhaite qu'une chaine de caractère possède au moins 5 caractères, il faudra utiliser le test **MIN_LENGTH_X** de telle façon :

```
array('MIN_LENGTH_5');
```

Ou encore si l'on souhaite que la chaine contienne obligatoirement entre 8 et 12 caractères, on écrira :

```
array('REQUIRED', 'MIN_LENGTH_8', 'MAX_LENGTH_12');
```

Tests personnalisés

Tests PARMI

Il est possible de créer des tests personnalisés dans votre classe (ici classe `Exemple_csvimport`). La première personnalisation possible sont des tests dits « PARMI ». Il s'agit de tests simples qui vont vérifier si la colonne CSV contient un élément 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 $PARMIxxxxxx = 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 (sensible à la casse).

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
));
```

Attention : Si le tableau de valeurs de comparaison doit être créé dynamiquement (par exemple à partir d'une table de la base de données), celui-ci doit être créé à l'intérieur du constructeur de votre classe :

```
public function __construct() {
    //construction du test personnalisé PARMI_SOUS_CATEGORIE
    $this->PARMI_SOUS_CATEGORIE = SqlSimple::distinct(_PT_.'sous_categories', 'id_sous_categorie');
}
```

Tests REGEX

Si vous souhaitez réaliser un test personnalisé à partir d'une expression régulière, il faut le signaler en intégrant au paramètre match de votre champ le mot-clé `REGEX`. L'expression régulière devra alors être passée au paramètre `matchValue`. En reprenant notre exemple, on aurait pu tester la validité de la date de nos films en utilisant une expression régulière personnalisée. Le code aurait alors été le suivant :

```
//autre possibilité : test avec un REGEX personnalisé
$this->createColonne('annee', array(
    'colonne' => 1,
    'libelle' => 'Année',
    'sqlField' => 'annee',
    'match' => array('REQUIRED', 'REGEX'),
    'matchValue' => '#^[0-9]{4}$#',
    'commentaire' => 'Année de sortie du film (4 chiffres obligatoires)',
    'css' => 'text-danger',
    'active' => true
));
```

Tests CUSTOM

On peut enfin créer n'importe quel autre type de test en faisant appel à une méthode callback. Pour réaliser un tel test personnalisé il faut le signaler en intégrant au paramètre match de votre champ le mot-clé `CUSTOM`. La méthode appelée en retour (callback) qui va réaliser le test devra alors être passée au paramètre `matchValue`. En reprenant encore une fois notre exemple, on aurait pu tester la validité du visuel de nos films en utilisant une méthode callback personnalisée. Le code aurait alors été le suivant :

```
//autre possibilité : test avec une méthode personnalisée
```

```
$this->createColonne('visuel', array(
    'colonne' => 3,
    'libelle' => 'Visuel',
    'sqlField' => 'visuel',
    'match' => array('REQUIRED', 'CUSTOM'),
    'matchValue' => 'testVisuel',
    'commentaire' => 'Type de visuel (saisie obligatoire)',
    'css' => 'text-danger',
    'active' => true
));
```

Avec la méthode `testVisuel` suivante :

```
public function testVisuel($valeur) {
    return in_array($valeur, $this->PARMI_VISUEL);
}
```

Attention, la méthode doit obligatoirement être publique !

4 Création d'une méthode d'enregistrement des données

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;
    }

}
```

5 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 (prédéfinis ou personnalisés) pour chaque colonne. Maintenant que nous avons chargé notre fichier CSV en mémoire, nous pouvons appliquer ces tests. 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 enregistrements 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);
```

6 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 / 0 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;
    }
}
```

7 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

Valeurs de 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

```
int getColonne(string $id)
```

Liste des paramètres

id : identifiant de la colonne interrogée.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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.

Valeurs de retour
Un tableau contenant la liste des identifiants de tests : ex : **`REQUIRED`**, **`CHECK_INTEGER_4`**.

getMatchValue

Renvoie la valeur de paramètre pour un match de type `REGEX` ou `CUSTOM`.

Description
```
string getMatchValue(string $id)
```

Liste des paramètres
`id` : identifiant de la colonne interrogée.

Valeurs de retour
Une chaine de caractères contenant soit une expression régulière (dans le cas d'un test `REGEX`), soit une méthode callback (dans le cas d'un test `CUSTOM`).

getActive

Renvoie l'état de la colonne **`id`**.

Description
```
boolean getActive(string $id)
```

Liste des paramètres
`id` : identifiant de la colonne interrogée.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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.

Valeurs de retour
La valeur par défaut.

getNbColonnes

Renvoie le nombre de colonnes définissant le modèle.

Description

```
int getNbColonnes()
```

Liste des paramètres
Aucun

Valeurs de 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

Valeurs de 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`.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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
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());
```

setMatchValue

Positionne le paramètre de valeur pour les tests **match** `REGEX` et `CUSTOM`.

Description
```
setMatchValue(string $id, string $value)
```

Liste des paramètres
id : identifiant de la colonne à positionner.
value : valeur à passer au match `REGEX` ou `CUSTOM`.

Valeurs de retour
Aucun.

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.

Valeurs de 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.

Note :
- 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.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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.

Valeurs de 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

```
int modeleColCount()
```

Liste des paramètres

Aucun

Valeurs de 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 : chemin complet du fichier CSV à charger.

Valeurs de retour

Tableau (construit selon le modèle prédéfini) contenant les enregistrements chargés.

Note

- 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).

Valeurs de retour
true : une ou plusieurs erreurs ont été rencontrées lors du test de l'enregistrement. En retour, chaque champ en erreur est affichés en rouge dans l'`enregistrement`.
false : aucune erreur n'a été rencontrée lors des tests de cet enregistrement.

Notes
- 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…

Valeurs de retour
Chaine de caractère HTML à afficher. Exemple :

displayErrors

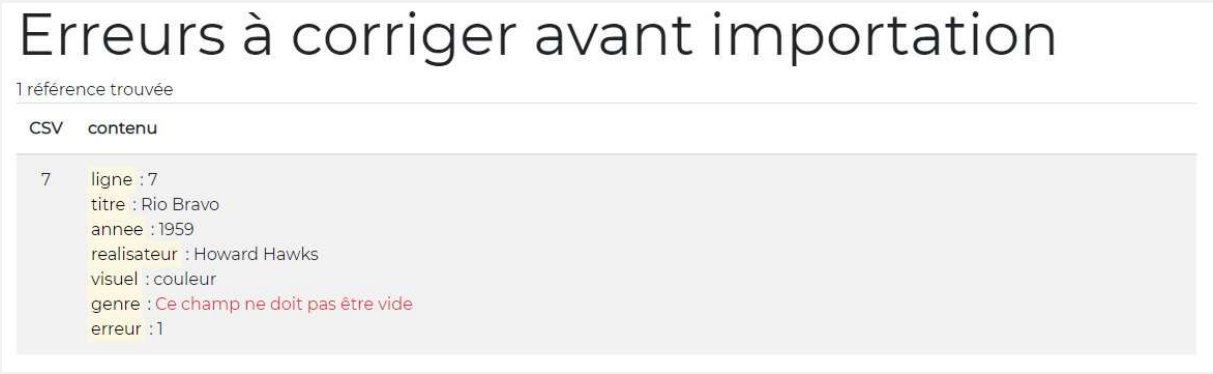
Affichage du jeu de données importé de manière structurée.

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.

Valeurs de 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.

Valeurs de retour
Chaine de caractère HTML à afficher. Exemple :