Gestionnaire d'upload de fichier en PHP

Présentation

Il est relativement simple d'écrire un script PHP permettant de gérer l'upload de fichier via le protocole HTTP. Lorsqu'il s'agit de charger un seul fi codes suffisent à l'obtention d'un résultat satisfaisant. En revanche, la procédure devient (quelque) peu plus compliqué lorsque l'on souhaite char simultanément.

Quoi qu'il en soit, l'upload de fichiers ne se résume jamais au seul codage d'un formulaire de transfert. De nombreux traitements doivent générale l'emploi de filtres restrictifs pour tester la validité d'un document en rapport avec le contexte du système d'information.

Dans cette optique, la classe suivante permet d'éviter la majorité des erreurs courantes lors de la création d'un système d'upload, mais fournit éga méthodes simples pour appliquer des filtres ou des traitements sur les documents après leur transfert.

Caractéristiques pour la gestion du formulaire de transfert :

- chargement d'un nombre infini de fichiers dans un seul formulaire,
- filtre sur les extensions,
- filtre sur les entêtes de fichiers suivant trois techniques :
  - analyse des entêtes renvoyés par le navigateur,
  - analyse via la librairie mime_magic,
  - analyse via la librairie fileinfo
- filtre sur la hauteur / largeur maximale d'une image

Caractéristiques des post-traitements :

- récupération aisée des informations sur le fichier (poids, nom, emplacement, extension, entête...)
- possibilité de renommer complètement le fichier ou simplement de lui attribuer un préfixe ou un suffixe
- gestion du mode d'écriture (remplacement, copie...)

Sans oublier une gestion des erreurs exhaustive permettant de connaître rapidement les raisons d'un transfert échoué.

Côté technique, la classe fonctionne à partir de la version 5.0.4 de PHP. Du côté client, tout navigateur web fera l'affaire. Elle est compatible dai **safe_mode** actif, pour peu que les droits d'accès au répertoire temporaire d'upload - répertoire de transition des fichiers - vous ait été alloué.

Concernant PHP, vous devrez vérifier quelques lignes de configuration dans le **php.ini**, mais avant de fouiller là dedans, la toute première vérific l'hébergement de votre application autorise l'upload de fichiers.

Pour ce faire, vous devrez afficher les informations de configuration de PHP sur votre serveur distant via la fonction **php_info()** :

error_log	*no value*	*no value*
error_prepend_string	*no value*	*no value*
error_reporting	6135	6135
expose_php	On	On
extension_dir	./	./
file_uploads	On	On
highlight.bg	#FFFFFF	#FFFFFF
highlight.comment	#FF9900	#FF9900
highlight.default	#0000CC	#0000CC
highlight.html	#000000	#000000

Si le paramètre **file_uploads** est positionné à **On**, aucun souci. Autrement, la classe proposée ne vous sera d'aucune utilité, il vous faudra probabl d'un hébergement plus élaboré.

Débuter avec le composant

Pour appréhender le fonctionnement de l'objet, nous partirons d'un modèle rudimentaire alimenté par la suite avec des traitements davantage pe

ELABORATION DU FORMULAIRE

Le cas de figure par lequel nous débuterons consistera à charger un seul fichier sur le serveur web. La base de toute upload réside dans la prépara destiné aux utilisateurs.

A ce titre, la classe interviendra uniquement pour la création du champ **MAX_FILE_SIZE**, définissant la taille maximale d'un fichier autorisée par l **parcourir**, permettant à l'utilisateur de naviguer au sein de l'arborescence de son disque pour trouver le fichier à charger.

Le champ **MAX_FILE_SIZE** est configuré par la classe en analysant la valeur correspondante dans le fichier de configuration de PHP. Par la suite, n modifier cette propriété.

Le reste est à la charge du développeur, qui personnalisera le formulaire suivant ses propres besoins.

Tout code utilisant des librairies externes doit en priorité charger le module avant d'être à même d'instancier l'objet issu de ladite librairie. Une f faudra lancer la méthode **InitForm()** pour configurer les deux champs précédemment cités :

```
<?php
// Chargement de la classe
require_once('../upload.class.php');

// Instanciation d'un objet Upload
$Upload = new Upload();

// Initialise les champs MAX_FILE_SIZE et le champ de type 'FILE'
$Upload->InitForm();
?>
```

Reste à créer de toute pièce le formulaire :

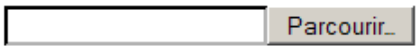
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
    <title>Exemple Classe Upload</title>
</head>

<body>
<form action="un_champ.php" method="post" enctype="multipart/form-data" name="formulaire" id="formulaire">
<?php
print $Upload->Field[0];
print $Upload->Field[1];
?>
<br>
<input type="submit" value="Envoyer" name="submit">
</form>
</body>
</html>
```

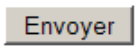
Une remarque de la plus haute importance : la balise HTML **<FORM>** doit **obligatoirement** contenir l'attribut **enctype="multipart/form-data"**. Si bonnement impossible d'envoyer un fichier sur le serveur. Dans le cas où votre upload échoue, il s'agit toujours du premier point à contrôler.

Après l'appel à la méthode **InitForm()**, vos champs seront par la suite manipulables grâce aux propriétés **Field[num_champ]** de l'objet Upload. **Fi** MAX_FILE_SIZE, **Field[1]** le champ de type 'FILE' :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
        <title>Exemple Classe Upload</title>
</head>
```

```
<form method="post" enctype="multipart/form-data" action="sample.php">
<input type="hidden" name="MAX_FILE_SIZE" value="1048576" /> <-> $Upload->Field[(
<input type="file" name="userfile[]" /> <-> $Upload->Field['
<input type="submit" value="Envoyer" name="submit">
</form>
</body>
</html>
```





GESTION DU FORMULAIRE

Pour gérer la soumission du formulaire, nous utiliserons le même fichier. C'est cependant une pratique à éviter en production puisque un rafraîch
nouveau le fichier.

Le formulaire HTML contenant un bouton **submit**, nous pouvons être informé de l'état de la validation en testant l'existence du bouton via la fonc

```php
// Teste la soumission du formulaire
if (!isSet($_POST['submit'])) {

    // Lance et teste le bon déroulement des opérations
    if ($Upload->Execute() === false) {
        print 'Il y a eu une erreur';
    }
    else {
        print 'L\'upload s\'est déroulée normalement';
    }
}
```

Le code ci-dessus effectue bien peu de chose : il teste la soumission du formulaire et lance la procédure de gestion de l'upload par l'emploi de la r

L'appel à cette méthode renvoie toujours un booléen permettant de tester la réussite ou l'échec du chargement. Si la fonction renvoie **false**, le t
différents cas de figure sur l'échec d'une upload seront abordés dans la section "gestion des erreurs" en fin du tutoriel.

Il n'y a aucune autre opération nécessaire pour l'implémentation d'un système d'upload basique. Le script complet se résume donc à ceci :

```php
<?php
// Chargement de la classe
require_once('../upload.class.php');

// Instanciation d'un objet Upload
$Upload = new Upload();

// Teste la soumission du formulaire
if (!isSet($_POST['submit'])) {

    if ($Upload->Execute() === false) {
        print 'Il y a eu une erreur.';
    }
    else {
        print 'L\'upload s\'est déroulée normalement';
    }
}

// Initialise les champs MAX_FILE_SIZE et le champ de type 'FILE'
$Upload->InitForm();
?>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>Exemple Classe Upload</title>
</head>

<body>
<form action="un_champ.php" method="post" enctype="multipart/form-data" name="formulaire" id="formulaire">
<?php
print $Upload-> Field[0];
print $Upload-> Field[1];
?>
<br/>
<input type="submit" value="Envoyer" name="submit"/>
</form>
</body>
</html>
```

A aucun moment nous n'avons paramétré l'objet **$Upload**. Par défaut, il utilise la configuration suivante :

- tout type de fichier est autorisé,
- autorise les fichiers allant jusqu'à la taille maximale spécifiée dans le fichier de configuration **php.ini**,
- les fichiers sont chargés dans le répertoire courant de la librairie **upload.class.php**,
- un seul champ **parcourir** est géré,
- un champ **parcourir** laissé vide ne génère pas d'erreur,
- si un fichier identique existe déjà dans le répertoire de la classe, il sera écrasé par le fichier uploadé,
- aucun filtre n'est appliqué.

Paramétrage de la classe

POUR LE FORMULAIRE

Revenons à notre formulaire. La classe Upload se base sur la configuration du **php.ini** pour configurer le champ **MAX_FILE_SIZE**.

Ce champ indique la taille maximale, exprimée en octet, que votre serveur web autorise pour un transfert HTTP. L'initialisation par défaut de cet

Bien qu'il soit impossible d'augmenter cette valeur via l'objet, l'inverse est possible. Pour limiter les fichiers à 1Mo, il suffit de renseigner, avant l'a **InitForm()**, la propriété **MaxFilesize** :

```
$Upload->MaxFilesize = 1024;
$Upload->InitForm();
```

Le poids s'exprime ici en kilo-octets pour des raisons de commodité, le travail de conversion en octets étant pris en charge par le composant. Suiv document dépassant la limite fixée des un méga octets sera refusé et générera une erreur.

Autre personnalisation possible à ce niveau, l'ajout d'attributs à notre champ de type **file**. Pour ajouter des attributs HTML, comme une classe CSS des évènements... l'objet dispose de la propriété **FieldOptions** :

```
$Upload->FieldOptions = 'style="border: 1px solid black;"';
```

POUR LA GESTION CÔTÉ SERVEUR

Dorénavant, seule la gestion côté serveur des documents sera traitée. Côté client nous avons le fait tour des possibilités, bien que vous pourriez a supplémentaires, comme une barre de progression, mais qui sortent du cadre de la classe puisque nécessitant l'emploi de code JavaScript ou Flash

Comme expliqué précédemment, le répertoire de destination de vos fichiers est par défaut celui de la librairie **upload.class.php**.

Il est bien entendu possible de modifier ce paramétrage pour utiliser un chemin différent, si tant est que le dossier convenu dispose des droits d'é (**IUSER_xxx** sous IIS).

Le paramètrage du dossier de destination peut être décrit sous sa forme absolue ou relative. Concernant la règle d'écriture, une utilisation sous W
slashs, tandis que Linux nécessitera des antislashs. Pour un code multi-plateforme, je ne saurais trop vous conseiller l'emploi de la constante pré-
DIRECTORY_SEPARATOR.

La présence en fin de chemin du séparateur de dossier est automatiquement gérée par l'objet. S'il est absent, il sera ajouté. La propriété à config
doit être précisée avant l'appel à la méthode **Execute()** :

```
// Définition du répertoire de destination (ici le répertoire courant du script)
$Upload->DirUpload = '.';
$Upload->Execute();
```

Le reste du paramétrage sera abordé dans un autre chapitre. Nous allons poursuivre avec l'étude des informations retournées par la classe après u
succès.

Informations disponibles sur les documents transférés

Avoir le document sur le serveur web est déjà une bonne chose. Reste dorénavant à retrouver les informations sur le document, pour alimenter u
autre.

Voici la liste des informations disponibles sur un document après un transfert réussi :

- le nom original du document (nous verrons pourquoi des différences peuvent apparaître par la suite),
- le nom tel qu'il a été sauvegardé sur le serveur,
- l'entête MIME du fichier,
- son extension,
- le chemin d'accès complet,
- son poids exprimé en kilo-octets avec une une précision à 3 chiffres

Il existe deux méthodes pour obtenir ces informations.

MÉTHODE GETSUMMARY()

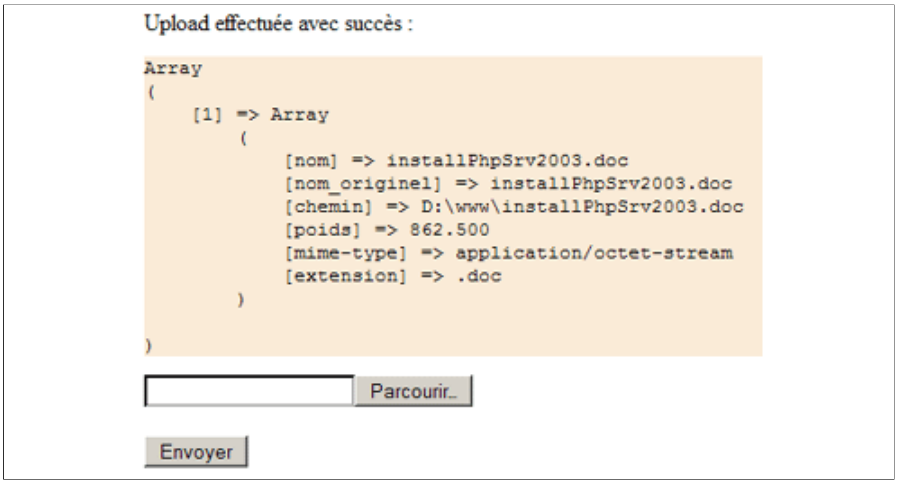
L'appel à la méthode **GetSummary()** retourne un tableau à deux dimensions :

```
Array
(
    [numero_du_champ] => Array
        (
            [nom] => ''
            [nom_originel] => ''
            [chemin] => ''
            [poids] => ''
            [mime-type] => ''
            [extension] => ''
        )

)
```

Pour obtenir ce tableau, modifiez votre code comme suit :

```php
// Teste la soumission du formulaire
if (!Empty($_POST['submit'])) {

    if ($Upload->Execute() === false) {
        print 'Il y a eu une erreur.';
    }
    else {
        print 'L\'upload s\'est déroulée normalement.<xmp>';
        print_r($Upload->GetSummary());
        print '</xmp>';
    }
}
```

Après le transfert d'un fichier, vous obtiendrez ce résultat :



Notez que le tableau débute à l'index **1**, ce qui correspond au premier champ de type **file** de notre formulaire. La classe sachant gérer plusieurs ch même formulaire, il était indispensable d'utiliser un tableau à deux dimensions.

A ce stade, la récupération du chemin d'accès au fichier peut s'obtenir en écrivant :

```php
$infos = $Upload->GetSummary();
echo $infos[1]['chemin'];
```

Un paramètre optionnel à la méthode **GetSummary()** vous permet d'obtenir directement les informations sur le champ souhaité. Il suffit de passer intéressé pour acquérir un tableau limité à une seule dimension :

```php
print '<xmp>';
print_r($Upload->GetSummary(1));
print '</xmp>';

/*
Aboutira à l'affichage de :
Array
(
    [nom] => installPhpSrv2003.doc
    [nom_originel] => installPhpSrv2003.doc
    [chemin] => D:\www\installPhpSrv2003.doc
    [poids] => 862.500
    [mime-type] => application/octet-stream
    [extension] => .doc
)
*/
```

Dorénavant, le chemin d'accès peut être retrouvé plus simplement :

```php
$fichier1 = $Upload-> GetSummary(1);
echo $fichier1['chemin'];
```

Il en ira de même pour les autres informations.

PROPRIÉTÉ INFOS[]

Une autre solution assez simple permet d'accéder à vos informations. Elle consiste à passer directement par la propriété **Infos[]** de l'objet :

```
// Affichage du poids
echo $Upload->Infos[1][poids];

// Affichage du chemin
echo $Upload->Infos[1][chemin];

// Affichage du nom
echo $Upload->Infos[1][nom];

//etc...
```

L'index **1** précisant la volonté de relever les informations du premier champ du formulaire.

DIFFÉRENCE ENTRE LES PROPRIÉTÉS "NOM_ORIGINEL" ET "NOM"

Vous aurez certainement remarqué dans ce chapitre consacré à la récupération des données la présence de deux propriétés en apparence similair

Leur différence est subtile : lors d'un transfert, l'objet upload analyse le nom du fichier et supprimera automatiquement tous les caractères spéci les suivantes :

- un caractère accentué sera substitué par son équivalent sans accent,
- un point sera remplacé par un souligné bas (underscore),
- un espace sera également remplacé par un souligné bas.

Ce formatage a pour dessein de rendre le document compatible avec le système de fichier implémenté sur le serveur web. La propriété **nom** repr tel qu'il fut écrit sur le serveur, tandis que **nom_originel** mémorise l'appellation du fichier avant son passage dans la moulinette de nettoyage des sur l'ordinateur client.

Par exemple, un document nommé "guide.définitif de PHP.pdf" sera renommé en "guide_definitif_de_php.pdf".

Travailler les fichiers

PERSONNALISER LE NOM DU DOCUMENT

L'objet **Upload** met à votre disposition trois propriétés combinables pour modifier le nom du fichier sur le serveur :

- **Filename** : modification du nom complet
- **Prefixe** : ajout d'un préfixe
- **Suffixe** : ajout d'un suffixe

```
// Attribution d'un nouveau nom
$Upload->Filename = 'document';

// Ajout d'un préfixe
$Upload->Prefixe = 'pre_';

// Ajout d'un suffixe
$Upload->Suffixe = '_suf';

// Au final, le fichier sur le serveur s'appellera : pre_document_suf.ext
// où ext prendra la valeur de l'extension du fichier uploadé
```

Avec cet extrait de code, le fichier sur le serveur sera nommé "pre_document_suf.ext", où **ext** représente l'extension original du document charg

MODES D'ÉCRITURES

Seul le nom du fichier uploadé permet d'identifier un document sur le serveur. Si un second fichier nommé identiquement est transféré sur le ser
l'existant. Mais ce n'est pas une fatalité, juste un comportement par défaut de l'objet upload.

Notre objet peut également travailler dans d'autres modes d'écritures basiques. Pour intervenir sur le mode d'écriture, vous devrez jouer sur la pr

- définie à **Upload::CST_UPL_WRITE_ERASE**, le nouveau fichier remplace celui présent sur le serveur,
- définie à **Upload::CST_UPL_WRITE_IGNORE**, le nouveau fichier est tout simplement ignoré, sans générer d'erreur,
- définie à **Upload::CST_UPL_WRITE_COPY**, le nom du nouveau fichier sera précédé de la mention **copie_de**.

```
// Pour faire une copie du fichier
$Upload->WriteMode = Upload::CST_UPL_WRITE_COPY;
$Upload->Execute();
```

MODIFIER LES PERMISSIONS

Pour éviter toute expérience malheureuse lors de l'utilisation du fichier après transfert, le composant active systématiquement les permissions ma
lecture, écriture et exécution pour le propriétaire, le groupe et les utilisateurs.

Cette opération est effectuée en utilisant la fonction **chmod(0666)** de PHP. Pour modifier l'attribution des droits, utilisez la propriété **Permissio**

Ainsi, pour que le propriétaire du fichier puisse lire et écrire un fichier tout en interdisant cette fonctionnalité pour le groupe et les autres utilisa

```
$Upload->Permission = 0644;
$Upload->Execute();
```

Critères de réussite

La classe Upload autorise la mise en place de vérifications sommaires pour tester la validité d'un document. Outre la vérification à mon sens la plu
des fichiers étudiée dans un chapitre à part, il est possible de mettre en place deux types de critères : la vérification sur les dimensions maximale
l'obligation de renseigner un fichier pour valider le formulaire.

LE CHAMP "PARCOURIR" EST OBLIGATOIRE

Par défaut, un champ de fichier laissé vide n'est pas considéré comme l'échec d'une upload. Pour forcer l'utilisateur à choisir un fichier, donc acti
vide, il est nécessaire de renseigner la propriété **Required** :

```
$Upload->Required = true;
```

Par conséquent, la soumission d'un formulaire avec un champ de fichier vide générera une erreur lors de l'appel à la méthode **Execute()**. Dès lors,
l'échec du transfert et prendre les mesures qui s'imposent.

VÉRIFIER LES DIMENSIONS D'UNE IMAGE

Si vous travaillez sur un formulaire d'upload d'images, vous pouvez mettre en place des critères de restriction par hauteur/largeur minimales et m
combinables dont l'unité est le pixel sont accessibles :

- **ImgMaxWidth**
- **ImgMinWidth**
- **ImgMaxHeight**
- **ImgMinHeight**

```
// Pour vérifier que l'image ne dépassera pas les 200 pixels en largeur
$Upload->ImgMaxWidth  = 200;

// Pour vérifier que l'image ne dépassera pas les 200 pixels en hauteur
$Upload->ImgMaxHeight = 200;
```

Lors de la vérification des fichiers, si le document est détecté en tant qu'image, les procédures d'analyse sur les dimensions seront automatiqueme

Pour faciliter l'affectation de valeurs à ces propriétés, il est possible de passer par la méthode **SetImgDim()** :

```
SetImgDim($maxHeight = null, $minHeight = null, $maxWidth = null, $minWidth = null)
```

Tous les paramètres sont optionnels. Pour modifier les propriétés **ImgMaxWidth** et **ImgMinWidth** seulement, ajoutez cette ligne :

```
$Upload->SetImgDim(200, null, null, 100);
```

Les types d'images supportées sont ceux acceptés par la fonction PHP **GetImageSize()**. Pour en obtenir la liste complète, référez-vous à la docum
GetImageSize().

Sécuriser le transfert: appliquer des filtres

Jusqu'à présent, nous avons étudié la création d'un gestionnaire d'upload sans restriction sur les types de documents. De fait, n'importe quel fichi
serveur web. Le composant **Upload** peut vous aider à mettre en place des restrictions performantes dont nous allons examiner les méthodes.

FILTRER PAR EXTENSION

La première catégorie de filtre disponible consiste à analyser l'extension du document chargé. Prenons le cas concret d'un site de galerie d'images
extensions non souhaitées, on utilise la logique inverse en interdisant tout type d'extension, puis en spécifiant uniquement celle correspondant à

Pour ce faire, il faut renseigner la propriété **Extension**. Elle consiste en une suite d'extensions autorisées (**précédées d'un point**) et séparées par
:

```
// Liste des extensions autorisées
$Upload->Extension = '.gif;.jpg;.jpeg;.bmp;.png';
$Upload->Execute();
```

Tout document outrepassant ce filtre sera ignoré lors de la phase d'écriture du fichier sur le serveur et activera une erreur.

Toutefois, si ce filtre est très utile, il n'est pas totalement efficace. Si l'utilisateur modifie l'extension d'un fichier indésirable par **.jpg**, le fichier c
sera chargé sans générer d'erreur. Pour bien faire, il est indispensable d'analyser l'entête du fichier.

FILTRER PAR TYPE MIME

Les entêtes de fichiers sont le seul moyen efficace pour vérifier la véritable nature des documents. Le principe avec la classe Upload est similaire
Nous utiliserons la propriété **MimeType** pour définir les seuls type de documents autorisés :

```
// Liste des entêtes de fichiers autorisés
$Upload->MimeType = 'image/gif;image/pjpeg;image/jpeg;image/bmp;image/x-png';
$Upload->Execute();
```

Désormais, le transfert d'un document autre qu'une image est impossible. Enfin presque. En effet, il faut savoir que par défaut, l'entête de fichie
navigateur du client. Deux constats s'imposent :

- les informations sur les entêtes de fichiers peuvent différer d'un navigateur à un autre,
- comme toute information envoyée par le client, cette dernière peut être falsifiée.

Pour obtenir un résultat satisfaisant, il faut passer outre la récupération des entêtes grâce au client, et les analyser côté serveur. Le composant U
techniques de vérification des entêtes décrites ci-après. La propriété **$phpDetectMimeType** permet le basculement d'un mode opératoire à un au

ANALYSE DES ENTÊTES VIA LE NAVIGATEUR

```
$Upload->phpDetectMimeType = Upload::CST_UPL_HEADER_BROWSER;
```

Il s'agit du comportement par défaut de l'objet, comme expliqué en introduction aux filtres par type MIME.

ANALYSE DES ENTÊTES VIA L'EXTENSION MIMEMAGIC

```
$Upload->phpDetectMimeType = Upload::CST_UPL_HEADER_MIMEMAGIC;
```

Le principe est simple : détecter l'encodage du fichier en recherchant certaines séquences à certaines positions dans le document (dixit la docum

Manuel PHP : Fonctions Mimetype).

Cette extension n'est pas compilée dans le core de PHP. Pour l'activer, PHP doit être compilé avec l'option **--enable-mime-magic**. Sous Windows, l'existence de la dll **php_mime_magic.dll** et de vérifier sa présence dans la section **Windows Extensions** du fichier **php.ini** :

```
extension=php_mime_magic.dll
```

Toujours dans le fichier de configuration **php.ini**, il faudra déclarer une nouvelle section comme suit :

```
[MIME_MAGIC]
;PHP_INI_SYSTEM Disponible depuis PHP 5.0.0.
mime_magic.debug = 0
;PHP_INI_SYSTEM Disponible depuis PHP 4.3.0.
mime_magic.magicfile = "$PHP_INSTALL_DIR\magic.mime" où $PHP_INSTALL_DIR fait référence à  votre chemin jusqu'à l'ex
```

Le fichier **magic.mime** n'est pas fourni avec PHP. Il est téléchargeable sur la page suivante : http://gnuwin32.sourceforge.net/packages/file.htm \file\).

Néanmoins, pour faciliter la tâche de tout le monde, ce fichier est inclu dans la distribution du composant upload dans le répertoire **mime_magic** copier à la racine de l'exécutable PHP (étape nécessaire sous windows, pas sûr pour les autres OS).

Comme vous pouvez le constater, l'installation de cette extension n'est pas évidente. Sur le site de l'éditeur, il est déconseillé de l'utiliser, non du propres tests démontrent qu'elle est tout à fait efficiente
-, mais parce qu'il existe une solution plus simple à mettre en oeuvre via l'extension **PECL fileinfo**.

ANALYSE DES ENTÊTES VIA L'EXTENSION FILEINFO

```
$Upload->phpDetectMimeType = Upload::CST_UPL_HEADER_FILEINFO;
```

Voici donc la méthode préconisée. C'est la plus simple à mettre en oeuvre et celle garantissant la meilleure protection pour vérifier la nature d'un

Pour commencer, vous devrez télécharger la collection de modules PECL depuis la page de téléchargement générale de PHP : Collection of PECL choisir la collection en adéquation avec la version PHP de votre serveur).

Sous Windows, déplacez la dll **php_fileinfo.dll** dans le répertoire des extensions de php puis déclarez dans la section **Windows Extensions** du php

```
extension=php_fileinfo.dll
```

Assurez-vous que la dll dispose des permissions suffisantes pour être exécutée par votre serveur web. Pour éviter des erreurs à la limite du compr fichier "magic" est livré avec la classe upload. Il est issu de l'installation d'Apache 2.0.59.

Une note à propos de ce fichier : que ce soit pour une mise en place sur un site FTP ou sur un CVS, **archivez le en mode binaire** et non en fichier Visual SourceSafe : sa détection automatique du type de fichier lors d'un archivage place le fichier en mode texte. Cela change le format du fichi suite le fonctionnement de la librairie **fileinfo**.

Gestion des erreurs

Eu égard à ce que nous avons vu, une dizaine d'erreurs peuvent être interceptée en cas d'échec d'un transfert. Elles sont accessibles via les const

- **CST_UPL_ERR_EXCEED_INI_FILESIZE** : le document excède la directive [upload_max_filesize] du fichier de configuration [php.ini].
- **CST_UPL_ERR_EXCEED_FORM_FILESIZE** : le document excède la directive MAX_FILE_SIZE spécifiée dans le formulaire.
- **CST_UPL_ERR_CORRUPT_FILE** : document corrompu.
- **CST_UPL_ERR_EMPTY_FILE** : le champ [parcourir] du formulaire d'upload n'a pas été renseigné.
- **CST_UPL_ERR_WRONG_MIMETYPE** : le document n'est pas conforme à la liste des entêtes autorisés.
- **CST_UPL_ERR_WRONG_EXTENSION** : le document n'est pas conforme à la liste des extensions autorisées.
- **CST_UPL_ERR_IMG_EXCEED_MAX_WIDTH** : la largeur de l'image excède celle autorisée.
- **CST_UPL_ERR_IMG_EXCEED_MAX_HEIGHT** : la hauteur de l'image excède celle autorisée.

- **CST_UPL_ERR_IMG_EXCEED_MIN_WIDTH** : la largeur de l'image est inférieure à celle autorisée.
- **CST_UPL_ERR_IMG_EXCEED_MIN_HEIGHT** : la hauteur de l'image est inférieure à celle autorisée.

RÉCUPÉRER UNE ERREUR

Pour obtenir la cause réelle d'une défaillance lors d'un transfert, faîtes appel à la méthode **GetError()**. Elle fonctionne de manière similaire à **Get**

```
if (!$Upload->Execute()) {
    print 'Il y a eu une erreur :';
    print '<xmp>';
    print_r($Upload->GetError());
    print '</xmp>';
}

/* Produira un résultat similaire à :
Array
(
    [1] => Array
        (
            [6] => Le fichier n'est pas conforme à la liste des entêtes autorisés (test.php)
            [7] => Le fichier n'est pas conforme à la liste des extensions autorisées (test.php)
        )
)*/
```

A l'instar de la méthode **GetSummary()**, la spécification en paramètre optionnel du numéro du champ concerné permet de réduire le tableau à un

```
print '<xmp>';
print_r($Upload->GetError(1));
print '</xmp>';

/*
Array
(
    [6] => Le fichier n'est pas conforme à la liste des entêtes autorisés (test.php)
    [7] => Le fichier n'est pas conforme à la liste des extensions autorisées (test.php)
)
*/
```

APPLIQUER DES TRAITEMENTS SPÉCIFIQUES

Vous aurez peut être la nécessité d'entreprendre des actions spécifiques en fonction de l'erreur retournée. Il faut alors manipuler le tableau des e
code pour vous aider dans son utilisation :

```
if (!$Upload->Execute()) {

    // Récupère les erreurs sur le premier champ de formulaire
    $erreurs = $Upload->GetError(1);

    // Parcours du tableau, ajustement des traitements
    foreach ($erreurs as $code_erreur=>$lib_erreur) {

        switch ($code_erreur) {

            case Upload::CST_UPL_ERROR_WRONG_EXTENSION :
                 // Vos traitements en cas d'erreurs sur une mauvaise extension de fichier.
                 echo $lib_erreur;
            break;
        }
    }

    // Stoppe le script en cours d'exécution
    exit;
}
```

MODIFICATION DES LIBELLÉS

Si les messages d'erreurs proposés par le composant upload ne vous satisfont pas, vous pouvez les personnaliser en utilisant la méthode **SetMsgErr**

Au sein de votre nouveau libellé, vous pouvez inclure la syntaxe **%FILENAME%** pour afficher le nom du document en cause de l'erreur :

```
$Upload->SetMsgError(Upload::CST_UPL_ERROR_WRONG_EXTENSION, 'Extension du fichier %FILENAME% erronée.');
```

Transfert de plusieurs fichiers simultanément

Dernier chapitre dédié à l'utilisation du composant Upload, la gestion de plusieurs champs d'upload dans un même formulaire.

Revenons à l'étape de création du formulaire. Une propriété n'a pas encore été abordée : **Fields**. Elle permet de préciser le nombre de champs de utiliser deux champs **parcourir** :

```
$Upload->Fields = 2;
$Upload->InitForm();
```

```
<form enctype="multipart/form-data">
<?php
// Le champ MAX_FILE_SIZE
print $Upload->Field[0];

// Affichage du premier champ de fichier
print $Upload->Field[1] . '<br/>';

// Affichage du second
print $Upload->Field[2];
?>
<br/>
<input type="submit" value="Envoyer" name="submit"/>
</form>
```

Cette simple manipulation suffit, la suite du fonctionnement du composant Upload étant similaire au transfert d'un seul fichier.

Lorsque vous déterminez des propriétés à l'objet Upload, elles sont appliquées à l'ensemble des champs de type **file**. Voici donc la seule limitation impossible de spécifier des propriétés pour un champ précis (par exemple le premier champ acceptera uniquement des images tandis que le secon uniquement des fichiers textes).

Conclusion, téléchargement

Ceux qui utilisaient la précédente mouture de l'objet - anciennement disponible sur le décédé miasmatik.net et toujours accessible sur le site php quelques différences :

Disparition de la variable globale **$UploadError**
Elle est tout simplement remplacée par le retour du booléen sur la méthode **Execute**. Néanmoins, pour garder la compatibilité descendante, la va toujours. Son utilisation est dépréciée.

Disparition du test de sécurité sur le **REFERER**
Ce test partait d'un bon sentiment, mais comme l'indique un commentaire sur le site phpCS, il est aussi inutile que foireux. Quoi qu'il advienne, ce pas incomber à la classe mais à une pratique globale de sécurité du site web.

Disparition de la propriété **SecurityMax**
Cette propriété filtrait tout fichier uploadé possédant l'entête **application/octet-stream** ou tout document de scripts PHP (par vérification de l'ex part cela ne permettait pas de filtrer véritablement les fichiers "dangereux", d'autre part il y avait là une redondance tout à fait inutile des métho extension/entête. A l'instar de la variable $UploadError, la propriété existe toujours dans cette nouvelle version mais son utilisation est égalemen

Enfin, pour conclure avec cette présentation, je vous invite à tester le fichier **sample.php** fourni dans la distribution du composant. Il reprend de toutes les propriétés abordées dans ce tutoriel.

Le téléchargement est disponible sur la page dédiée : http://www.miasmatech.net/scripts/upload/upload.php

#1. **Par Duncan**

Génial!!
Ca fonctionne à merveille, super simple à utiliser et puissant, merci beaucoup pour cette classe et pour le tutoriel clair et limpide!

Que du bonheur...

#2. **Par The End**

Pas très concluant, j'ai essayé de modifier les champs $Upload->Filename en "test" & $Upload->DirUpload en "./docs/" (ds lexemple !) sans le moin
déplacent toujours vers la racine de upload.class.php sans changer de nom...

Je ne sais pas si le NTFS à quelque chose à voir la dedant (pas d'erreurs) mais je pense que je vais devoir trouver autre chose...

Dommage ça avait l'air vachement sympa.

Je viens d'essayer les deux manipulations décrites sans rencontrer le moindre problème (en ayant pris soin de faire les tests sur le fichier pro

*Le changement de répertoire et le nommage du fichier ont tout deux réussis sans manipulation autre que de désactiver les commentaires pré
l'état, sans plus d'informations, je ne peux aller plus loin dans mes recherches. La piste du NTFS est à ignorer. Le problème ne peut être lié a
(qui plus est c'est celui que j'emploie).*

*Pour tout message relatant un problème, merci de laisser votre adresse mail pour que je puisse prendre contact directement avec vous et vo
place de l'objet. Tout commentaire pouvant aider à l'amélioration des créations que je maintiens est accepté avec plaisir.*

*Néanmoins, il me semble dommage de laisser ce genre de commentaire sans suite car ils peuvent influer de manière négative les lecteurs de
cause véritable du problème ait pu être identifiée. Est-ce un simple problème de configuration du serveur, un problème de droits? Les causes
l'environnement et non à l'objet, c'est une éventualité à prendre en considération.*

*Si d'autres utilisateurs rencontrent le même genre de soucis, je vous invite à le faire savoir ici. Je ne suis (hélas) pas à l'abri d'un code défect
reproduire les erreurs avant de les corriger.*

*Je rappelle que laisser son adresse email ne représente aucun risque : elle n'est jamais publiée sur le site, et croyez moi, j'ai bien d'autres ch
un catalogue d'adresse mail pour les revendre ;).
Elle n'a pour but qu'entrer en contact avec les lecteurs le désirant.*

#3. **Par The End**

Toutes mes excuses !
je me suis débattus avec ce code pendant deux heures et je pense que j'ai été un peu trop injuste...

Donc libre à vous d'effacer ce commentaire, car après avoir attribué tout les droits au groupe "tout le monde" de windows sur le dossier source et
cas ou), les problèmes semblent avoir disparus... Les droits donc semblent être la cause de mes soucis.

Maintenant, une question me turlupine, j'ai plusieurs champs (4) qui attendent des images, pour éviter les doublons, j'aimerais les renommer grac
incrémetation vu que les timestamps sont en sec. et que les scripts prennent bien moins de temps à s'effectuer), le problème donc, c'est que dès
exemple sont mis, une erreur survient vu qu'il re nommé en même temps !

Est-ce quelque chose à été prévu pour ce cas de figure ?

Merci, et désolé encore pour les commentaires précédents sortits un peu trop vite et sous l'effet de la rage ^^ .

Aucun soucis, je peux comprendre la frustration...

*Pour ton problème, je conseille de réaliser le nommage des fichiers en post traitement de l'upload.
Une simple boucle foreach sur le tableau retourné par la méthode GetSummary devrait suffire, vu que l'on dispose des propriétés [chemin] e*

*Pour renommer les fichiers, la fonction "rename" de php semble toute indiquée. Pour être sûr d'avoir un nom unique, je pense que la fonctio
l'affaire.*

#4. **Par The End**

Merci pour l'astuce ;)

Et surtout merci pour cette super class, vraiment efficace, ...quand ça marche ^^

#5. **Par lebossarnaud**

C'est super ton tuto

Merci, ça fait plaisir un petit commentaire comme ça de temps en temps...

#6. **Par Rykian**

Pas mal, pas mal :)
Par contre, deux choses me choquent un peu (même si ça ne change que peu de chose au script). Sans m'attarder sur le script, j'ai vu une variable
tellement dit que c'était mal que j'ai fini par en être convaincu.
Deuxième chose, ayant surtout travailler en POO sur Java, les majuscules en première lettre pour les fonctions j'trouve ça assez troublant. Habitu
majuscule aux classes, mais pour les fonctions et variables, on écrit façon camel case avec une minuscule obligatoirement en premier.

Ce ne sont que des conventions, mais voilà :)

> j'ai vu une variable au moins en public
N'hésite pas à en faire part, je suis preneur de tout conseil ou amélioration.

> les majuscules en première lettre pour les fonctions j'trouve ça assez troublant.
Certes, mais il s'agit là d'une classe dont la première ébauche date déjà de quelques années (à l'heure des premières versions de PHP4), où les
étaient encore mal établies (enfin, personnellement j'entends). Je n'ai jamais eu le courage de modifier cette mauvaise initiative avec le te
la ressource est partagée avec d'autres utilisateurs, il serait malvenu de changer...

> Ce ne sont que des conventions
mais qu'il est important d'en établir et de s'y tenir, on est bien d'accord...