UploadAjaxABCI Version 4.5
auteur Alain Bontemps ABCIWEB

Upload de fichiers, reprise d'upload, affichage des informations en temps réel, styles et fonctions événementiels. Compatible **Navigateur html5** "récents" : Firefox, Chrome, IE >= 10

Fichiers de test
UploadAjaxABCI_Basique.php, **UploadAjaxABCI.php, UploadAjaxABCI_Redimensions.php, UploadAjaxABCI_Custom.php, UploadAjaxABCI_Custom_VerifFileExist.php, UploadAjaxABCI_Crop_multiple.php, UploadAjaxABCI_Crop_multiple_multiple.php, UploadAjaxABCI_Controle_input_text.php.**

Nécessite
- UploadAjaxABCI.js (classe javascript)
- **Jquery.js** (et "jquery.imgareaselect.js" pour les exemples de crop)
- UploadAjaxABCI_Php_Load.php (ou équivalent, destination de la requête Ajax)
- UploadAjaxABCIServeur.php (ou équivalent, classe d'upload serveur)

Upload les fichiers par fragments compilés dans un fichier temporaire quand ils dépassent une valeur paramétrable. L'upload terminé, le fichier temporaire est renommé avec son nom initial (éventuellement corrigé) et déplacé vers son emplacement définitif. Cette technique totalement transparente pour l'utilisateur, permet de télécharger aussi bien des petits fichiers que des très gros :

- Permet de surpasser les limitations serveur "upload_max_filesize" et "post_max_size".

- Permet la reprise d'un téléchargement interrompu après un arrêt volontaire ou une panne interne ou serveur, en utilisant la partie sauvegardée.

- Supporte l'upload multiple avec un système de file d'attente et surpasse la config. "max_file_uploads".

- Customisation des boutons de sélections avec comportements drag and drop et/ou onclick.

- Retour d'informations en temps réel, personnalisable et très souple, en utilisant simplement des classes prédéfinies sur des éléments html de votre choix :
a/ De nombreuses informations sont disponibles durant l'upload : nom, taille, vignette de prévisualisation (si c'est une image), progression graphique, pourcentage, progression textuelle, sauvegarde partielle effectuée, temps passé, temps restant estimé, status (en cours, ok, erreur, arrêt), ainsi que **deux commandes** : arrêter et arrêter-tout, qui permettent l'arrêt optimisé afin de pouvoir effectuer une reprise d'upload ultérieure dans les meilleures conditions.

b/ Dispose de **style événementiels** à définir selon vos besoins et qui s'appliqueront en fonction des différentes étapes du traitement des fichiers et du formulaire.

c/ Dispose de **fonctions javascript événementielles** à définir selon vos besoins et qui seront exécutées à différentes étapes du traitement des fichiers et du formulaire.

d/ Hautement **configurable**

bleu : classes Html, gris : fonctions Javascript et Css événementiels, jaune : config Javascript, vert : Php

Présentation de l'affichage des informations et contrôles des champs

Le plus simple est de prendre le fichier "UploadAjaxABCI_Custom.php" pour avoir une solution complète prête à l'emploi.

Néanmoins pour pouvoir faire son propre module d'upload sur mesure, commencez par lire ce mode d'emploi et regardez les fonctionnalités de base dans le fichier "UploadAjaxABCI_Basique.php" (voir également les commentaires dans le code source des fichiers).

Ensuite se référer au fichier "UploadAjaxABCI.php" qui regroupe d'autres exemples de configuration de la classe javascript et d'utilisation des ***styles css événementiels***.

Le fichier "UploadAjaxABCI_Custom.php" est un exemple plus élaboré qui reprend les fonctionnalités déjà décrites et donne un exemple d'utilisation des ***fonctions javascript événementielles***.

Les **styles css événementiels** et les **fonctions javascript événementielles** sont conçus **pour fournir des commandes** en fonction du retour des informations en temps réel et donc pour la présentation dynamique du formulaire. Ils ne sont pas indispensables pour l'upload, ni pour les informations de retour qui s'affichent suivant les **classes html** que vous utiliserez (ou non) sur des éléments de votre code html.

Par exemple un élément html ayant la classe "UpAbci_name" `<span class="UpAbci_name"></span>` permettra d'afficher le nom du fichier, tandis que `<span class="UpAbci_remainingTime"></span>` affichera le temps restant estimé.

A la fin de l'upload du fichier, l'affichage du temps restant ne sera plus utile. Je pourrais le laisser en place à 0 mais si je veux gagner de la place pour afficher d'autres informations ou simplement épurer pour ne garder que les informations utiles, je peux supprimer cet affichage en utilisant le style css événementiel "data-upabcicss-result" qui sera activé dès la fin de traitement du fichier. Le code html deviendra donc `<span class="UpAbci_remainingTime" data-upabcicss-result="display:none"></span>`.

Le principe est donc simple, c'est la multitude des possibilités et combinaisons qui peut rendre le code plus compliqué (c'est aussi le principe de la programmation).

Mais vous voyez qu'il y a une grande latitude pour personnaliser l'affichage des informations en modifiant uniquement le html sans avoir besoin de modifier la classe javascript elle-même, ce qui serait vraiment beaucoup plus compliqué voire infaisable pour les débutants.

Tout comme les styles événementiels, les ***fonctions événementielles*** sont également déclenchées en fonction des événements du formulaire et du traitement des fichiers. Elles permettent d'aller plus loin dans la programmation de la présentation et d'interfacer le script avec des scripts externes. A ce titre je donne deux exemples de Crop + Upload qui intègrent le module jquery "ImgAreaSelect".

La fonction événementielle "config.func_onFormSubmit" permettra quant à elle d'intercepter la soumission du formulaire. Cela permet de contrôler des champs dans le formulaire indépendamment des champs de type file et d'utiliser cette classe même si l'envoi de fichier est optionnel. Cette fonction est utilisée dans les exemples "UploadAjaxABCI_Php_Controle_input_text.php" et la variante "simple".

1/ SCRIPT PHP QUI ALIMENTE LE FORMULAIRE

- Modifier éventuellement la variable $UpAbci_fragmentSize qui détermine la taille maximale des fragments de sorte que cette valeur ne soit pas supérieure à la configuration serveur 'upload_max_filesize' (en cas de problème voir les NOTES en fin de ce mode d'emploi). Les fichiers en dessous de cette taille n'auront pas de sauvegarde temporaire. Plus cette taille est petite (minimum 1Mo) plus les sauvegardes seront fréquentes mais vous générerez plus de trafic et le temps d'upload total sera plus lent.

Dans l'absolu le script peut fonctionner sans variables php transmises dans le formulaire puisque "$UpAbci_fragmentSize" peut être renseigné par les options de configuration de la classe javascript, de même pour "$UpAbci_uniqidForm" qui défini une variable de session. Mais la variable de session est utilisée comme jeton de sécurité, et il est plus judicieux que la taille du fragment soit définie par le serveur. Dans la plupart des cas vous pouvez laisser le code inchangé.

Actuellement (fin 2015) une valeur de fragment de 8Mo semble être une bonne moyenne. Avec une liaison adsl classique cela permet une sauvegarde toute les 3 minutes environ (dépend aussi du serveur cible).

2/ FORMULAIRE HTML

- Le formulaire doit contenir un ou plusieurs champs de type file, avec ou sans attribut "multiple".
- La classe javascript recherche également deux champs "input" nommés :
  name = "UpAbci_uniqidForm"
  name = "UpAbci_fragmentSize"

Si ces champs sont vides ou inexistants la classe utilisera ses valeurs par défaut ou les valeurs configurées avec les options de configuration javascript. C.f. remarque* dans le paragraphe précédent.

- Pour personnaliser le bouton de sélection des fichiers, utilisez l'option de configuration javascript "config.customFileSelect" décrite plus bas.

- Vous pouvez ajouter d'autres champs dans le formulaire, ils seront automatiquement transmis au script serveur par la requête ajax. Côté serveur vous pourrez les récupérer en php avec la fonction "urldecode()" :

```
$ma_variable = urldecode($_POST['ma_variable']);
```

Pour ajouter des champs spécifiques à chaque fichier (en cas d'upload multiple), voir les fichiers exemples de "Crop".

3/ CONFIGURATION DE L'AFFICHAGE DES INFORMATIONS HTML

(cf fichiers contenant les formulaires)

Toutes les classes définies ci-dessous doivent se situer à l'intérieur du formulaire ou du conteneur des résultats de l'upload (correspondant au premier et troisième paramètre déclarés dans l'initialisation de la classe javascript).

A1/ Retour général d'informations serveur

.UpAbci_infosServer → bloc d'information général du formulaire renseigné par le serveur

Retour d'informations serveur dans le bloc ayant la classe "UpAbci_infosServer". Ce bloc n'est pas spécifique à chaque fichier.

A2/ De nombreuses informations et **deux comportements** peuvent être affichées durant le téléchargement du ou des fichiers. Les informations s'afficheront, ou les comportements seront initialisés, dans les éléments html de votre choix en utilisant les classes suivantes :

.UpAbci_stopAll → stop tous les téléchargements (comportement onclick)

En cas d'upload multiple, arrête le téléchargement du fichier en cours et les fichiers suivants (pas d'incidence sur les téléchargements précédents).

.UpAbci_infosFile → conteneur indispensable pour afficher les informations sur les fichiers

En cas d'upload multiple, ce bloc contenant les informations d'un fichier est dupliqué. En conséquence ne pas utiliser d'id à l'intérieur de cet élément. Liste des classes qui peuvent êtes utilisées à l'intérieur de ce bloc pour afficher les informations ou le comportement correspondant :

.UpAbci_name	→ nom
.UpAbci_size	→ taille
.UpAbci_progressionG	→ progression graphique dans une balise html <progress>
.UpAbci_progressionT	→ progression textuelle
.UpAbci_percentage	→ pourcentage effectué
.UpAbci_backup	→ sauvegarde effectuée (0 si upload terminé ok)
.UpAbci_duration	→ temps passé
.UpAbci_remainingTime	→ temps restant
.UpAbci_status	→ status en cours, ok, arrêt, erreur, éventuellement complété côté serveur
.UpAbci_stop	→ arrêt de l'upload du fichier (comportement onclick)
.UpAbci_imgPreview	→ affiche l'aperçu si c'est une image dans la limite des valeurs "config.imgPreviewMaxSize" et "config.imgPreviewMaxSizeTotal"
.UpAbci_inputFileName	→ affiche le nom de l'input de type file

Vous pouvez écrire des valeurs par défaut dans vos éléments html (ex : 0 Mo), elles seront remplacées automatiquement en temps voulu.

- Les valeurs par défaut des éléments utilisant les classes "UpAbci_progressionT", "UpAbci_backup" et "UpAbci_remainingTime" sont restituées quand l'upload est terminé.

- Les valeurs par défaut sont aussi analysées pour déterminer l'éventuel espacement à appliquer entre la valeur numérique et son unité correspondante. Si non renseignée ou si les premiers caractères ne sont pas numériques, l'espacement utilisé est celui de la configuration info.unitsSpacing (' ' par défaut).

B/ Styles événementiels
Les styles sont spécifiées (si besoin) avec l'attribut "data-upabcicss-..." dans vos éléments html et seront appliquées en fonction des événements suivants :

1/ Styles applicables sur tous les blocs contenus dans le formulaire ou dans le conteneur des résultats spécifié en troisième paramètre de l'initialisation de la classe javascript :

- data-upabcicss-select-file → des fichiers ont été sélectionnés
- data-upabcicss-error-img-prev-total → total images supérieur à "config.imgPreviewMaxSizeTotal"
- data-upabcicss-submit → le formulaire a été envoyé
- data-upabcicss-submit-file → des fichiers ont été envoyés par le formulaire
- data-upabcicss-upload-end → fin de traitement de tous les fichiers
- data-upabcicss-backup-end → sauvegarde pour au moins un fichier en fin de traitement
- data-upabcicss-form-end → fin de traitement du formulaire avec ou sans fichiers traités
- data-upabcicss-infos-server → le serveur a envoyé un message vers .UpAbci_infosServer

2/ Styles applicables sur le bloc .UpAbci_infosServer (mais pas ses éléments contenus), ainsi que sur le conteneur des résultats lui-même (défini en troisième paramètre dans l'initialisation de la classe javascript) et sur tous les blocs contenus dans le bloc d'information de chaque fichier défini avec la classe ".UpAbci_infosFile" (à noter que les éléments html inclus dans le conteneur des résultats mais exclus de ".UpAbci_infosFile" ne seront pas ciblés par les styles ci-dessous).

- data-upabcicss-error-img-prev → taille de l'image supérieure à "config.imgPreviewMaxSize"
- data-upabcicss-error-img-prev-total → total images supérieur à "config.imgPreviewMaxSizeTotal"
- data-upabcicss-image-preview → image de prévisualisation disponible (si respect de config js)
- data-upabcicss-error-user → erreur utilisateur (taille, extension). Exclus "error-img-prev"
- data-upabcicss-in-progress → le fichier est en cours de traitement
- data-upabcicss-backup → sauvegarde partielle disponible (ancienne ou en cours)
- data-upabcicss-result → traitement du fichier terminé quelque soit la cause
- data-upabcicss-result-stop → fichier ou upload arrêté sur demande utilisateur
- data-upabcicss-remaning-time-compute → s'applique après la durée de config.remainingTimeCompute
- data-upabcicss-remaining-time-after → s'applique après la durée de config.remainingTimeDisplayAfter
- data-upabcicss-result-ok → upload terminé ok
- data-upabcicss-result-partial → upload terminé et sauvegarde existante
- data-upabcicss-result-error → upload terminé erreur ou erreur utilisateur

Les styles doivent être écrits en respectant la syntaxe classique, exemple :

```
<span style="display:none;width:15px;height:10px" data-upabcicss-result-ok="display:inline-block;background-color:green" data-upabcicss-result-error="display:inline-block;background-color:red"></span>
```

Ordre d'application des styles
Certains styles peuvent entrer en concurrence, ils sont appliqués dans l'ordre suivant :

'select-file', 'error-img-prev', 'error-img-prev-total', 'image-preview', 'error-user', 'backup', 'submit', 'submit-file', 'in-progress', 'backup', 'result', 'result-stop', 'remaining-time-after', 'remaning-time-compute', 'result-ok', 'result-partial', 'result-error', 'upload-end', 'backup-end', 'form-end'

- "backup" est appliqué avant submit pour les fichiers sélectionnés disposant d'une sauvegarde préalable, et également après le submit si le fichier dispose d'une sauvegarde durant le téléchargement.
- L'ordre d'écriture dans le html n'a aucune incidence.

Styles configurables lors de l'initialisation de la classe

En complément de ces styles événementiels (qui sont à utiliser dans le html), vous disposez également des options de configuration javascript "config.cssSubmitOn" et "config.cssFileSelectOn" (voir plus bas) pour formater les boutons d'envoi du formulaire et les champs de type file durant l'upload. L'upload terminé, ces boutons retrouvent leur état initial.

4/ CONFIGURATION DE LA CLASSE JAVASCRIPT UploadAjaxABCI.js

```
var MyUpload = new UploadAjaxABCI('#param1', 'param2', '#param3');
```

Les deux premiers paramètres sont obligatoires, le troisième est indispensable pour l'affichage des informations d'upload pour chaque fichier.

- 1er paramètre : formulaire cible (id ou class)
- 2eme paramètre : destination de la requête ajax (adresse du script d'upload côté serveur)
- 3eme paramètre : bloc d'information html de l'upload (id ou class)

Le bloc défini par ce troisième paramètre sera réinitialisé automatiquement avec ses valeurs originales à chaque sélection de fichiers.

A noter que le comportement de la fonction en cas de navigateur obsolète incompatible peut être paramétré avec l'option "config.browserOutdeted".

```
var up = new UploadAjaxABCI('#form','Php_Upload/UploadAjaxABCI_Php_Load.php','#reponse_upload');

// Exemples de configuration de certaines options
up.info.status.ok = 'ok';
up.config.customFileSelect = ".bouton_fichiers";
// …

// Pour terminer, utiliser la fonction Start() pour initialiser le script, DOM chargé :
$(function(){up.Start()});
```

Liste des options de configuration javascript (l'ordre d'écriture n'a pas d'incidence)

Les paramètres ci-dessous sont listés avec les valeurs par défaut. Les valeurs numériques doivent être rentrées sans guillemets. A noter que pour les informations de retour "info." vous pouvez rentrer du html plutôt que du texte plat.

OPTIONS de configuration :

- config.ajaxTimeOut = 500; // en milli seconde (sans unités), délai d'envoi de la requête ajax, utile pour éviter de saturer un serveur local poussif. Evitez de mettre cette valeur à zéro. Je ne pourrai pas être tenu pour responsable si une sollicitation importante du système donne le coup de grâce d'un disque dur en cours d'agonie.

- config.filesExtensions = []; // Tableau des extensions autorisées (les variantes majuscules/minuscules sont automatiquement prises en compte). Si le tableau est vide aucune vérification n'est faite. Faire un contrôle supplémentaire côté serveur par sécurité (pas obligatoire mais très vivement conseillé).

- config.filesExtensionsInput = []; // Tableau des extensions autorisées pour chaque input du formulaire (les variantes majuscules/minuscules sont automatiquement prises en compte). La configuration requière le nom de l'input comme index. Par exemple si le champ se nomme "file1", la configuration pourra être : config.filesExtensionsInput["file1"] = ['jpeg','jpg','png']. Cette option peut être utile si plusieurs champs de type file sont inclus dans le même formulaire.

-

- config.fileSizeMax = null; // taille maximale du fichier utilisateur. Si = 0 ou null aucune vérification n'est faite sur la taille du fichier. Accepte un nombre en octets ou une chaine de caractère formatée avec les unités. Les espaces et la lettre "o" sont optionnels et le formatage ne tient pas compte de la casse. Par exemple 1073741824 = '1073741824o' = '1024Mo' = '1024 M' = '1048576k' = '1048576 Ko' = '1 go' = '1 G' (les nombres décimaux sont acceptés). Faire un contrôle supplémentaire côté serveur par sécurité (pas obligatoire mais très vivement conseillé).

- config.customFileSelect = null; /* Un id ou une class ex "#mon_bouton" ou ".mon_bouton" pour identifier un bouton personnalisé de sélection de fichier. Ce bouton (ou ce lien) doit se situer à l'intérieur du formulaire indiqué en premier paramètre dans l'initialisation de la classe. Par ailleurs vous devez mettre un style "display:none" sur votre champ de type file.
S'il existe plusieurs champs d'upload dans le même formulaire, utiliser une même classe pour définir les boutons (ou les liens) et ils seront affectés aux champs de type file dans l'ordre de leur écriture dans le html (le premier bouton sera affecté au premier champ, le second au second champs etc. */

- config.customDragAndDrop = true; // Si et uniquement si "config.customFileSelect" a été défini : true permet le drag and drop, false annule ce comportement et donc seul le comportement onclick sera disponible.

- config.cssSubmitOn = null; // styles avec notation standard (ex : "background:#ccc;border:1px solid blue;...") qui seront appliqués durant le traitement du formulaire sur les boutons de type "submit" inclus dans le formulaire. Les styles initiaux (initialement définis dans le html) seront restitués une fois le traitement terminé.

- config.cssFileSelectOn = null; // styles avec notation standard (ex : "background:#ccc;border:1px solid blue;...") qui seront appliqués durant le traitement du formulaire sur les boutons de type "file" OU les boutons défini par "config.customFileSelect" inclus dans le formulaire. Les styles initiaux (initialement définis dans le html) seront restitués une fois le traitement terminé.

- config.imgPreviewStartAuto = true; // booléen true ou false. true pour un affichage immédiat des vignettes images (si les contrôles sur le poids des images pour prévisualisation sont respectés). Si false vous pouvez utiliser la fonction "func_ImgPreviewStartAll()" pour faire afficher les vignettes dans le

formulaire au moment voulu (voir le fichier d'exemple UploadAjaxABCI_Custom.php et les fichiers de Crop).

- config.imgPreviewMaxSize = null; // en Mo (sans unités). Poids maximal de chaque image pour prévisualisation. Si null, aucun contrôle individuel n'est fait avant la prévisualisation mais vous pouvez néanmoins contrôler le poids total de l'ensemble des images avec la configuration "config.imgPreviewMaxSizeTotal".

- config.imgPreviewMaxSizeCancelAll = false; // true ou false. Si true et que "config.imgPreviewMaxSize" est dépassé pour au moins une image, aucune vignette ne sera affichée.

- config.imgPreviewMaxSizeTotal = null; // en Mo (sans unités). Poids total maximal de toutes les images pour prévisualisation. Si dépassé aucune vignette ne sera affichée. Si null, le contrôle n'est pas fait. Les images dépassant "config.imgPreviewMaxSize" ne sont pas totalisées puisqu'elles ne sont pas affichées. Par ailleurs, "config.imgPreviewMaxSizeCancelAll" à la priorité sur ce contrôle.

- config.imgPreviewMaxWidth = 90; // en pixels (sans unités), largeur maximale de l'aperçu image.

- config.imgPreviewMaxHeight = 60; // en pixels (sans unités), hauteur maximale de l'aperçu image.

- config.infosRefresh = 1; // en seconde (sans unités), fréquence de rafraîchissement des informations textuelles.

- config.remainingTimeCompute = 10; // en seconde, temps de calcul avant l'affichage du temps restant estimé. Si trop court le résultat n'est pas significatif. Durant l'attente c'est le paramètre "info.remainingTimeComputeWaiting" qui s'affiche. A l'issue de cette période, "data-upabcicss-remaning-time-compute" sera appliqué.

- config.remainingTimeDisplayAfter = 30; // en secondes (sans unités). Si le temps restant estimé pour l'upload du fichier est supérieur à cette valeur, "data-upabcicss-remaining-time-after" sera appliqué. Utilisée principalement pour ne pas faire afficher le temps restant (ou éventuellement la barre de progression) pour les petits fichiers.

- config.recoveryBackupConfirm = false; // booléen true ou false si reprise automatique sans demande de confirmation.

- config.submitWithoutFile = true; // booléen true ou false si le formulaire ne doit pas être envoyé en absence de fichiers valides. Si false les fonctions javascript événementielles "config.func..." concernant le suivi de la soumission du formulaire ne seront pas appelées, exceptée "config.func_FormEnd" si l'option ci-dessous est configurée à true.

- config.submitWithoutFileFuncFormEnd = false; // Sert uniquement dans le cas où config.submitWithoutFile = false. Si défini à true et qu'aucun fichier n'est soumis, permet néanmoins le déclenchement de la fonction événementielle "config.func_FormEnd(tableau)" (le tableau des fichiers sélectionnés est passé en paramètre).

- config.queryFormEnd = false; // booléen true ou false. Si = true, une requête additionnelle est envoyée au serveur pour indiquer la fin de traitement du formulaire avec comme paramètre un tableau d'information sur chaque fichier sélectionné (excepté si config.submitWithoutFile = false et qu'aucun

fichier n'a été soumis). Si l'option est égale à true et config.submitWithoutFile = true et qu'aucun fichier n'est soumis un tableau restreint est renvoyé directement dans l'unique requête adressée au serveur.

- config.BackupFormDependency = true; // booléen true ou false. "true" lie les possibilités de reprise d'upload du fichier à l'identifiant du formulaire. Voir les notes en fin du mode d'emploi.

- config.cookiePath = '/'; // path du cookie.

- config.fragmentSize = 4194304; // taille du fragment en octet (sans unités) si le script ne trouve pas de champ nommé "UpAbci_fragmentSize" (défini avec une valeur non nulle et supérieure à 1048576) dans le formulaire. Si une valeur valide est trouvée dans le formulaire alors config.fragmentSize sera égal à cette valeur - 10240. 10240 étant le nombre d'octets que prend arbitrairement la classe comme marge de sécurité pour que le post total (files + données textuelles) ne soit pas supérieur à la config php "post_max_size" dans le cas où l'on choisirait une valeur de fragment égale à "post_max_size".

- config.browserOutdeted = 'Navigateur obsolète incompatible'; /*
Message pour les vieux navigateurs incompatibles.
- Si non vide et différent de null, ce message s'affichera à la sélection des fichiers et à la soumission du formulaire. Le code javascript de la classe ne sera pas exécuté et le formulaire ne sera pas envoyé.
- Si le message est vide ou vaut null, aucun message ne sera affiché, le code javascript de la classe ne sera pas exécuté mais le formulaire sera envoyé à l'adresse définie dans l'attribut html "action" du formulaire. Pour que la sélection de fichiers soit fonctionnelle dans ce cas, n'utilisez pas de boutons de sélection customisés puisque d'une part la fonction de customisation est définie plus loin dans le code javascript de la classe, et que d'autre part les navigateurs obsolètes ne supportent pas nativement le drag and drop.
*/

- config.serverFatalErrorDisplay = true; // Concatène les messages renvoyés par les erreurs fatales du serveur au status javascript "info.status.errorServer" ou "info.queryEndErrorServer" (défini plus bas). Les messages renvoyés dépendent du script côté serveur. Deux fonctions "cathErrorServeur()" et "setModeDebug()" sont disponibles dans la classe d'upload php pour gérer le retour des erreurs fatales (cf fichiers exemples de redimensionnements et de crop). Vous pouvez laisser true pour les exemples fournis car le script serveur ne renvoie que les messages personnalisés définis avec "cathErrorServeur()". "setModeDebug()" renvoie tout mais ne devrait pas être utilisé en production.

- config.uniqidForm = this.SHA1(this.Uniqid('UploadAjaxABCI',true)); // valeur générée à l'affichage du script s'il ne trouve pas de champ nommé "UpAbci_uniqidForm" dans le formulaire.

// **Informations de retour**

// Annonces html, informations fichiers (éventuellement vous pouvez rentrer du html plutôt que du texte plat)
- info.status.ok = 'Téléchargement ok'; // complété éventuellement par un message provenant du script php d'upload.

- info.status.inProgress = 'Téléchargement en cours';

- info.status.stop = 'Arrêt'; // sur demande utilisateur.

- info.status.errorSize = 'Dépassement de la taille maximale autorisée.'; // message si dépassement de "config.fileSizeMax".

- info.status.errorExtension = 'Extension non valide.'; // message si l'extension ne fait pas partie du tableau "config.filesExtensions".

- info.status.errorServer = 'Echec du téléchargement. '; // complété automatiquement par l'erreur serveur (si config.serverFatalErrorDisplay = true) ou par une erreur générée par le script php d'upload.

- info.remainingTimeComputeWaiting = 'calcul en cours'; // s'affiche durant la période définie par la variable "config.remainingTimeCompute".

- info.unitsSpacing = ' '; // espacement par défaut entre les valeurs et leurs unités.

// Annonce javascript si config.recoveryBackupConfirm = true. Libellés de la boite de confirmation javascript pour la reprise de la sauvegarde.
- info.recoveryBackupConfirm.name = 'Récupération du fichier : '; // Annonce javascript suivie du nom du fichier.

- info.recoveryBackupConfirm.size = '\nSauvegarde en cours : '; // Annonce (suite) suivie de la taille de la sauvegarde (\n pour retour ligne car la boite de confirmation est en javascript).

- info.recoveryBackupConfirm.message = '\n\nCliquez sur "OK" pour compléter ce fichier ou sur "Annuler" pour réinitialiser le téléchargement.'; // Annonce (suite) dernière ligne.

// Annonce de fin de traitement du formulaire si config.queryFormEnd = true et erreur serveur lors de cette dernière requête
- info.queryEndErrorServer = 'Echec dans la finalisation du traitement serveur. '; // complété automatiquement par l'erreur serveur si config.serverFatalErrorDisplay = true

- Remarque : la fonction de prévisualisation des images est assez gourmande en ressources (dépend de la résolution des images). Si vous utilisez le bloc de prévisualisation des images ".UpAbci_imgPreview" décrit plus haut et que ce script doit être optimisé également pour les smartphones, vous aurez peut-être intérêt de détecter le périphérique utilisé par le visiteur et le cas échéant, modifier la valeur des options de configuration "config.imgPreviewMaxSize" et / ou "config.imgPreviewMaxSizeTotal".

- Notez également que la valeur de "config.remainingTimeDisplayAfter" sera réellement supérieure à celle indiquée car la vitesse calculée au début du téléchargement pour estimer le temps restant est toujours surévaluée : avec une valeur de 30 secondes, le style défini par "data-upabcicss-remaining-time-after" (décrit précédemment) ne sera peut-être pas appliqué pour des fichiers pouvant mettre jusqu'à un peu plus de 40 secondes pour se télécharger. Ce manque de précision est pratiquement sans importance car ce style est principalement prévu pour éviter d'afficher le temps restant - sans intérêt quand la progression graphique est suffisamment éloquente - pour les fichiers qui sont téléchargés rapidement.

FONCTIONS Javascript événementielles

- func_SubmitForm; // func_SubmitForm(); NE DOIT PAS être configurée, mais uniquement utilisée comme commande pour poursuivre le processus normal et déclencher la soumission du formulaire. Cette commande ne fonctionne que si la fonction événementielle "config.func_onFormSubmit()" a été définie (voir plus bas) et est conçue pour être utilisée à l'intérieur de cette fonction. A noter qu'après l'utilisation de "func_SubmitForm()" le paramètre "tableau" envoyé par la fonction "config.func_onFormSubmit(event,tableau)" est réinitialisé (vide).

- func_ImgPreviewStartAll; // func_ImgPreviewStartAll(); NE DOIT PAS être configurée, mais uniquement utilisée comme commande pour déclencher l'affichage de toutes les images de prévisualisation. Fonctionne uniquement si config.imgPreviewStartAuto = false (voir l'exemple UploadAjaxABCI_Custom.php et les exemples de Crop) sinon la prévisualmisation est automatique.

- config.func_ImgPreviewLoadEach = null; // config.func_ImgPreviewLoadEach = function(objet,objet){} sera déclenchée pour chaque image de prévisualisation quand elle est chargée (voir les exemples de Crop). L'objet image est passé en premier paramètre et le second paramètre contient un objet d'informations générales sur le fichier.

- config.func_FileSelectAllBefore = null; // config.func_FileSelectAllBefore = function(event,tableau){} sera déclenchée à la sélection des fichiers AVANT l'affichage des informations. L'objet "event" est passé en paramètre, ainsi qu'un tableau d'objets contenant des informations générales sur chaque fichier. En cas de sélections successives avant la soumission du formulaire, cette fonction peut être utilisée pour enregistrer les informations des fichiers précédemment soumis (voir l'exemple UploadAjaxABCI_Crop_multiple_multiple.php).

- config.func_FileSelectEach = null; // config.func_FileSelectEach = function(event, objet){} sera déclenchée à la sélection de chaque fichier APRES l'affichage de ses informations (voir l'exemple UploadAjaxABCI_Crop_multiple.php). L'objet "event" est passé en paramètre ainsi qu'un objet contenant des informations générales sur le fichier.

- config.func_FileSelectAll = null; // config.func_FileSelectAll = function(event,tableau){} sera déclenchée à la sélection des fichiers APRES l'affichage des informations de tous les fichiers (voir l'exemple "UploadAjaxABCI_Custom.php" et les exemples de Crop). L'objet "event" est passé en paramètre, ainsi qu'un tableau d'objets contenant des informations générales sur chaque fichier.

- config.func_onFormSubmit = null; // config.func_onFormSubmit = function(event,tableau){} si définie, stoppe l'envoi du formulaire lors de sa soumission. Conçue pour faire des traitements/contrôles complémentaires. L'objet "event" est passé en paramètre ainsi qu'un tableau d'objets contenant des informations générales sur chaque fichier. Vous devrez utiliser la fonction "func_SubmitForm()" pour poursuivre le traitement et soumettre le formulaire.

- config.func_FormSubmit = null; // config.func_FormSubmit = function(event,tableau){} sera déclenchée à la soumission du formulaire (voir l'exemple "UploadAjaxABCI_Custom.php" et les exemples de Crop). L'objet "event" est passé en paramètre ainsi qu'un tableau d'objets contenant des informations générales sur chaque fichier.

- config.func_FileInProgressEach = null; // config.func_FileInProgressEach = function(objet){} sera déclenchée lors de la progression du téléchargement de chaque fichier suivant la fréquence du

paramètre "config.infosRefresh". Toutes les informations en cours sont transmises en paramètre dans un objet.

- config.func_FileEndEach = null; // config.func_FileEndEach = function(objet, info_serveur, mixte_serveur) {} sera déclenchée à la fin du traitement de chaque fichier (voir les exemples de Crop). Un objet contenant des informations générales sur le fichier est passé en paramètre. Voir détail plus bas.

- config.func_FormEnd = null; /* config.func_FormEnd = function(tableau,info_serveur,mixte_serveur) {} sera déclenchée à la fin du traitement du formulaire (voir l'exemple "UploadAjaxABCI_Custom.php" et les exemples de Crop).

Un tableau de tableaux contenant des informations générales sur chaque fichier est passé en paramètre. Mêmes propriétés disponibles pour chaque objet que celles de la fonction func_FileEndEach. Voir détail plus bas.

Le second paramètre "info_serveur" contient la réponse éventuelle du serveur transmise en php avec la fonction addInfosServer(). Peut contenir du texte ou du html. Si une erreur serveur intervient, "info_serveur" sera construit en fonction des options "config.serverFatalErrorDisplay" et "info.queryEndErrorServer" côté javascript et de l'utilisation des fonctions "cathErrorServeur()" et "setModeDebug()" côté php.

Le troisième paramètre "mixte_serveur" contient la réponse éventuelle du serveur transmise en php avec la fonction addMixteServer(). Cette variable n'est pas affichée directement dans le html. Elle peut donc contenir du texte, du html ou encore un tableau que vous exploiterez comme bon vous semble.

Contenu des tableaux de fichiers passés en paramètre dans les fonctions événementielles

- fichier (objet)	contient le nom, la taille, le type... le nom du champ de type file, et le détail des erreurs traitées en javascript AVANT l'upload. A noter que "upabciErrorUser"(1) ne comptabilise pas "upabciErrorPreview".
- uniqid_file	
- obj (objets html)	tous les objets html d'informations de fichier ayant une classe ".Upabci_..."
- infos_html (objet html)	le bloc d'information html ".UpAbci_infosFile"
- cook_name	
- qte_save	quantité sauvegardée en continu
- qte_save_ini	sauvegarde initiale lors de la soumission du fichier
- qte_upload	
- time_start	
- time_end	
- result (2)	"ok_done", "backup_done", "error_done", "backup_fail", "error_fail" ou "0_0" préfixe = état du fichier, suffixe = état de la requête. "0_0" si pas traité.
- stop	0 ou 1 (sur demande utilisateur)
- stop_all	0 ou 1 (sur demande utilisateur)
- iteration	nombre d'itérations de la requête ajax pour le fichier
- img_prev_delayed	0 si immédiat, 1 si l'affichage est différé, -1 si pas de prévisualisation
- img_width	uniquement si prévisualisation (bloc ".UpAbci_imgPreview" utilisé)
- img_height	uniquement si prévisualisation (bloc ".UpAbci_imgPreview" utilisé)

Notes Importantes :
(1) Si vous souhaitez annuler la soumission d'un fichier, vous pouvez définir la propriété "fichier.upabciErrorUser" avec une valeur entière supérieure à 0, ex : "fichier.upabciErrorUser = 1".
(2) "result" qui indique le résultat **final** du fichier est renseigné au retour du traitement serveur sinon il vaut "0_0".

5/ CONFIGURATION DU SCRIPT PHP D'UPLOAD

Fichiers exemples dans le dossier "Php_Upload"

Gardez bien à l'esprit que le script sera appelé plusieurs fois pour le traitement du même post, excepté en cas d'upload simple si la taille du fichier est inférieure à la taille d'un fragment ou si aucun fichier n'est joint.

Configuration de la classe php "UploadAjaxABCIServeur"

```
$up = new UploadAjaxABCIServeur($dossier_destination, $dossier_temporaire, $cookie_heures = null, $cookie_path =null, $adresse_relative = null, $verif_filesize_sup2Go = false)
```

Seuls les deux premiers paramètres sont indispensables, les autres ont des valeurs par défaut.

- $dossier_destination → dossier de destination de l'upload
- $dossier_temporaire → dossier temporaire où sont enregistrées les sauvegardes
- $cookie_heures → la durée de vie du cookie de sauvegarde en heures
- $cookie_path → le path du cookie envoyé en php. Doit pouvoir être récupéré par le script en javascript
- $adresse_relative → adresse relative des dossiers
- $verif_filesize_sup2Go → mettre si possible à true après avoir lu l'avertissement

Les fonctions de la classe php sont documentées dans le fichier "UploadAjaxABCIServeur.php".

Vous pourrez récupérer les valeurs des champs input éventuellement ajoutés dans le formulaire avec la fonction php "urldecode" :

```
$ma_variable = isset($_POST['ma_variable']) ? urldecode($_POST['ma_variable']) : null;
```

Récupération des paramètres spécifiques de la requête Ajax
Des paramètres spécifiques à la classe javascript peuvent être récupérés en complément :

public $UpAbci_form est un tableau de données spécifique à la classe. Il est renseigné dans tous les cas, excepté pour la requête complémentaire de confirmation de fin de traitement du formulaire qui est envoyée si et uniquement si l'option de configuration javascript "config.queryFormEnd" = true. Dans ce cas et uniquement pour cette requête additionnelle, le tableau sera vide et vous pourrez récupérer les informations depuis "public $UpAbci_formEnd" évoqué plus bas.

Liste des index renseignés :

Dans tous les cas :

- id_form
- uniqid_form
- iteration_form — nombre d'itérations de la requête ajax pour le formulaire (remise à zéro pour chaque nouvelle soumission du formulaire)

Si un fichier est joint au formulaire :

- input_name — nom de l'input de type file
- uniqid_file
- cook_name
- name
- size
- type
- lastModified
- qte_save — quantité sauvegardée en continu
- qte_upload
- result * — "ok_done", "backup_done", "error_done", "backup_fail", "error_fail", ou "0_0"
  préfixe = état du fichier, suffixe = état de la requête.
- time_start
- time_end
- iteration — nombre d'itérations de la requête ajax pour le fichier

* A noter que "result" est le résultat final renseigné en javascript au retour de la requête ajax. Sa valeur sera donc toujours 0_0 côté serveur excepté pour la dernière requête si vous configurez l'option config.queryFormEnd = true pour envoyer une requête additionnelle en fin de traitement du formulaire.

Suivant le contexte, la requête ajax envoie d'autres variables :

- public $UpAbci_formEnd — **Uniquement disponible à la fin du traitement du formulaire si l'option de configuration javascript "config.queryFormEnd = true"**.
  Retourne un tableau de tableaux d'informations (1 pour chaque fichier) avec le même contenu que "$UpAbci_form" cité plus haut.
  Si aucun fichier n'est joint, le tableau renverra seulement les trois index "id_form", "uniqid_form" et "iteration_form".
  Si l'option javascript n'a pas été configurée ou si ce n'est pas la fin du traitement du formulaire $UpAbci_formEnd retourne un tableau vide.

protected $UpAbci_blobSlice — renvoie "true" si le fichier est en plusieurs morceaux

protected $UpAbci_fileEnd — renvoie "true" si c'est la fin du fichier (dernière partie).

protected $UpAbci_fragment // le fragment de ficher, ou la valeur 1 si une sauvegarde complète est trouvée lors de la sélection du fichier.

La méthode getParam($index) retourne les valeurs du tableau "$UpAbci_form" sinon éventuellement les valeurs de "$UpAbci_formEnd[0]" si $index = "id_form" ou $index = "uniqid_form" ou $index = "iteration_form" (pour être pratique).

Exemple :
```
$name = $up->getParam('name');  // retourne le nom du fichier
```
(retourne false si l'index n'existe pas)

Gestion des erreurs fatales du serveur :

Les messages d'erreurs seront concaténés au message configuré par la classe javascript avec "info.status.erreur" (voir exemple dans le fichier "UploadAjaxABCI_Php_Load_Redimensions.php")

Si aucune des deux fonctions "cathErrorServeur()" ou "setModeDebug()" n'est utilisée, les erreurs fatales ne seront pas transmises et seul le message javascript défini avec "info.status.erreur" s'affichera.

- "setModeDebug()" affiche les erreurs fatales non catchées par la fonction "cathErrorServeur()". A n'utiliser qu'en phase de développement. Faire précéder par "ini_set('display_errors', 1)" pour afficher toutes les erreurs php dans le formulaire (toujours uniquement en phase de développement).

- "cathErrorServeur($tableau)" demande un tableau en paramètre et permet de personnaliser le retour des erreurs fatales. L'index des valeurs est constitué par une suite de mots génériques renvoyés par l'erreur du serveur, et les valeurs sont constituées soit du message personnalisé à afficher, soit d'un tableau constitué du message personnalisé comme premier élément et de la valeur "true" (ou d'une valeur non nulle et différente de false) comme second élément pour indiquer de supprimer le fichier temporaire et l'éventuel cookie identifiant ce fichier.

Par exemple les erreurs lors des redimensionnements d'images proviennent la plupart du temps d'un dépassement des configurations serveur "memory_limit " et/ou de "max_execution_time". Les messages textuels renvoyés par le serveur commencent respectivement par : 'Allowed memory size …' et 'Maximum execution time...'. On peut donc gérer ces erreurs avec le tableau suivant :

```
$tab_erreurs = array();

$tab_erreurs['Allowed memory size'] = array("Mémoire insuffisante, le fichier est trop gros pour être redimensionné.",true);  // le cookie et le fichier de sauvegarde seront supprimés

$tab_erreurs['Maximum execution time'] = "Le temps d'exécution maximum du script est dépassé, rechargez votre image et réessayez !"; // le cookie et le fichier de sauvegarde seront préservés

$up->cathErrorServeur($tab_erreurs);
```

Vous pouvez bien entendu compléter le tableau d'erreurs par autant de lignes que vous voulez.
Pour l'exemple ci-dessus je fais l'hypothèse (pas forcément juste) que si une erreur mémoire se produit lors du redimensionnement, il est inutile de réessayer donc je peux effacer le fichier temporaire et

l'éventuel cookie enregistrant les coordonnées de la sauvegarde. Cela permet si besoin d'optimiser le nettoyage du dossier temporaire.
Par contre le dépassement de "max_execution_time" peut être dû à un serveur momentanément surchargé. On a donc peut-être intérêt de conserver le fichier éventuellement sauvegardé - disponible pour les fichiers excédant "$UpAbci_fragmentSize" - pour une nouvelle tentative. Cela permettra de le réutiliser directement lors du prochain chargement du fichier sans avoir besoin d'attendre pour son téléchargement.

Concernant les index du tableau, la fonction fait une recherche des mots constituant cet index dans le corps du message d'erreur renvoyé par le serveur (avec strpos($erreur_serveur,$index) !== false). Vous devez donc rentrez une suite de mots génériques qui correspond au texte de cette erreur, suffisamment pour bien cibler l'erreur mais pas trop pour éviter les variables !
Je n'ai pas trouvé d'autre solution car la fonction "error_get_last()" de php qui permet d'attraper les erreurs fatales ne renvoie aucun index correspondant à l'erreur (sinon le type qui est trop imprécis), mais simplement le texte de l'erreur.
Il en résulte que si le message d'erreur php venait à changer, l'erreur ne serait plus gérée par la fonction "cathErrorServeur()". Cependant l'expérience montre que ces messages d'erreur n'évoluent quasiment jamais au fil des versions php. Et si c'était le cas, les conséquences seraient très limitées car l'erreur sera néanmoins interceptée. Simplement aucun message correspondant ne serait trouvé et seul le message configuré en javascript par "info.status.erreur" s'afficherait. L'information visiteur serait donc seulement moins précise, mais rien d'autre (aucune incidence sur la confidentialité ou la sécurité).

note :
la gestion de ces erreurs est surtout utile pour le debug avec "setModeDebug())" lors de la configuration du script php, ou pour personnaliser les éventuels retours d'erreur d'un traitement complémentaire à l'upload avec "cathErrorServeur()". Concernant l'upload lui-même et donc si vous ne faites pas de traitement complémentaire, une fois votre script configuré vous ne devriez plus avoir d'erreur puisque cette solution surpasse les limitations serveurs spécifiques à l'upload avec la fragmentation.
La "seule" erreur possible restante est une perte de connexion (souvent après plusieurs heures d'upload en continu sur un serveur mutualisé). Dans ce cas la vérification du token échouera et vous devriez avoir le message d'erreur associé soit actuellement dans mes exemples : "Connexion non valide ou perdue. Rafraîchissez la page et recharger votre fichier, si celui-ci dispose d'une sauvegarde automatique elle sera utilisée".

6/ NOTES GENERALES

1/ Conditions de test : Pour faire des tests sur un serveur local il est conseillé de ne pas diminuer la valeur du paramètre config.ajaxTimeOut qui permet d'espacer l'envoi des requêtes.. Evitez de mettre cette valeur à zéro. Je ne pourrai pas être tenu pour responsable si une sollicitation importante du système donne le coup de grâce d'un disque dur en cours d'agonie.
Eviter d'utiliser des outils genre console Firebug durant l'upload car dans ce cas l'occupation mémoire monte (ou montait) graduellement et à terme le serveur d'évaluation pourrait ne plus répondre et s'arrêter pour ne pas saturer l'ordinateur ce qui se traduira par une perte de session et le jeton de formulaire ne sera plus valide. En conditions normales, l'occupation mémoire observée sur l'ordinateur - test en local pc wampserver sous W8 - ne dépasse pas 100 Mo maximum et stable, quelque soit la taille du fichier.
Eviter également d'ouvrir plusieurs onglets du même formulaire car vous risquer de vous trouver dans les conditions évoquées au point n° 4 suivant.

2/ Si code javascript complémentaire :
- Si vous souhaitez ajouter des comportements onsubmit, utilisez la fonction javascript "config.func_onFormSubmit", pour intercepter l'événement puis la fonction "func_SubmitForm" pour soumettre le formulaire. Cela permettra de poursuivre le traitement normal sur les fichiers.
- Si vous souhaitez ajouter des comportements onchange sur les boutons de sélection des fichiers, vous aurez des problèmes en cas de soumissions successives du formulaire sans rafraîchissement de la page. En effet pour compenser un bug IE10, le html du champ file est copié, supprimé puis recoller à chaque soumission du formulaire. Etant donné qu'il est impossible de cloner ce champ pour le bon fonctionnement du hack, les événements javascript ne seront pas réintégrés. Utilisez plutôt la fonction javascript "config.func_FileSelectAll" dont les paramètres retournent l'objet "event" et des informations complètes sur tous les fichiers à chaque sélection de fichiers. Des exemples sont fournis dans les fichiers "Custom" et de "Crop".

Notez également que les comportements programmés sur les boutons (ou liens) d'arrêt des fichiers seront désactivés par la classe javascript durant l'upload du formulaire qui imposera ses propres comportements pour optimiser une éventuelle reprise d'upload.

3/ Ecriture des options de configuration : Les options de configuration de la classe javascript ne sont pas testées avant d'être utilisées. Si vous rentrer une chaîne de caractères rompue ou des valeurs numériques entourées de guillemets, vous obtiendrez un dysfonctionnement sans message d'avertissement préalable. Les booléens true et false s'écrivent également sans être entourés de guillemets (sinon ils sont interprétés en tant que chaîne de caractère non signifiante, souvent interprétée comme true).

4/ Conditions de reprise d'upload : l'option "config.BackupFormDependency = true;" lie les possibilités de reprise d'upload des fichiers à l'identifiant du formulaire.
Si false, l'identifiant de formulaire n'est pas utilisé pour créer le nom du cookie de sauvegarde. Cela permet de récupérer l'éventuelle sauvegarde d'un fichier depuis différents formulaires, mais induit un problème dans le cas particulier où l'on télécharge ***simultanément des fichiers identiques depuis des formulaires différents*** : le second upload va retrouver la sauvegarde en cours du premier et le fichier final sera corrompu. Le script renverra alors un message d'erreur en fin d'upload "**Fichier temporaire non valide**").
Même chose si config.BackupFormDependency = true et que l'on ouvre le même formulaire dans différents onglets pour télécharger le même fichier .

5/ La vérification de l'intégrité d'un fichier utilise la fonction php "filesize()" qui peut renvoyer n'importe quoi avec des fichiers de plus de 2Go sur certains serveurs (c.f. doc php). Cette vérification est donc limitée par défaut aux fichiers de moins de 2Go. Si possible affectez la valeur true à la variable "$verif_filesize_sup2Go" (dans l'initialisation de la classe php) après avoir vérifié le bon comportement de votre serveur.

6/ Nettoyage du dossier des fichiers temporaire : Un fichier nommé "Nettoyage_temp.php" (inclus dans le dossier "Php_Upload") contient un exemple de script pour nettoyer les fichiers abandonnés en cours d'upload dans le dossier temporaire. Penser à utiliser ce script (ou équivalent) régulièrement. A noter que si vous limiter la possibilité d'upload des fichiers à une taille inférieure ou égale à la taille du fragment – 10 ko*, aucune sauvegarde n'est faite et donc vous pouvez utiliser cette classe sans avoir besoin de nettoyer ce dossier qui restera vide. (*10 ko sont réservés arbitrairement par la classe d'upload pour les données de type texte de sorte que la taille réelle du fragment sera celle indiquée – 10 Ko).

7/ Taille des fichiers : Il n'y a pas de limite à la taille des fichiers. J'ai testé avec succès des fichiers de plus de 9 Go en local. Sur serveurs mutualisés distants je me suis limité à des tests de 2,5 Go (car faut faire preuve de patience). La vitesse dépend de votre débit ascendant (1Mb actuellement avec adsl classique) mais aussi de la vitesse d'upload limitée par les paramétrages du serveur distant (souvent vers 500 kb pour les mutualisés d'entrée de gamme).

Par ailleurs certains serveurs optimisés pour surveiller drastiquement l'activité (genre mutualisés ovh) renvoient plus ou moins souvent une erreur 500 au bout d'un certain temps. Cela permet toutefois d'envoyer dans les 300 à 500 Mo en une seule passe que l'on pourra éventuellement compléter en réinitialisant le téléchargement. Sur ces mêmes serveurs j'ai néanmoins pu télécharger des fichiers de 1,5 Go en une seule passe, c'est donc très variable et dépend très probablement de l'activité globale du serveur.

8/ Détails techniques sur la sauvegarde : Le script d'upload utilise le fichier temporaire sauvegardé – si disponible (donc pour les fichiers dépassant la taille d'un fragment soit la variable "$UpAbci_fragmentSize" dans les exemples suivants) - pour compléter le téléchargement automatiquement dès l'envoi du même fichier ou après demande de confirmation. La durée de disponibilité de la sauvegarde dépend de la durée de vie du cookie envoyé par le script php d'upload.

Côté serveur, l'unicité du nom du fichier temporaire est assurée par trois uniqid générés à trois moments différents (affichage du formulaire, soumission du formulaire, réception de la requête ajax) et combinés à l'identifiant du formulaire* puis au nom et à la taille du fichier pour garantir une signature unique.

Côté client, le nom du cookie qui contient les informations de sauvegarde associées au fichier est composé de l'identifiant du formulaire*, du nom et de la taille du fichier. Cela est suffisant dans la mesure où il est extrêmement rare qu'on possède sur son ordinateur deux fichiers qui aient exactement le même nom et la même taille tout en étant différents, et que l'on télécharge ensuite le second à la suite de l'échec du téléchargement du premier. Pour cette raison la reprise d'upload est automatique par défaut mais on peut configurer le script pour qu'il demande une confirmation de reprise (cf l'option "config.recoveryBackupConfirm" de la classe javascript décrite plus bas).

Les informations de sauvegarde sont enregistrées dans un cookie, on ne peut donc récupérer un fichier partiellement téléchargé que depuis le même navigateur et le même formulaire* qui a envoyé la partie sauvegardée. Pour les reprises d'upload, faire attention de pouvoir récupérer les cookies envoyés par le

script php. Un cookie envoyé sur http://monsite.net ne pourra pas être récupéré depuis http://www.monsite.net, et inversement. En cas de problème lors du téléchargement, un visiteur qui s'est connecté depuis la première adresse ne pourra donc pas récupérer la partie sauvegardée s'il se connecte depuis la seconde adresse lors de sa visite suivante. Pour éviter ce problème, on peut définir une adresse unique par réécriture d'url dans un .haccess.
* Voir notes sur "config.BackupFormDependency**"** plus haut pour modifier ce comportement.

9/ Debug :

a/ Lancez le script "Php_Upload/Test_dossiers.php" qui est un utilitaire pour vérifier les autorisations des dossiers. Toutes les lignes doivent retourner "status ok", sinon attribuer des droits d'accès suffisants aux dossiers avec votre logiciel FTP.

b/ Une fois le contrôle réussi, si un problème persiste la valeur critique à modifier est la valeur des fragments, soit dans mes exemples : "$UpAbci_fragmentSize".

Tous les fichiers - exceptés "UploadAjaxABCI_Basique.php" et "UploadAjaxABCI_Redimensions.php" où j'ai voulu garder un code le plus concis possible - incluent une vérification de la configuration "upload_max_filesize" du serveur pour contrôler la valeur maximale des fragments. Cela suffit pour un serveur correctement configuré, mais tous ne le sont pas...

Par exemple le "php.ini" de **wampserver avec php 5.5.12** (version 2.5 64bits) affiche une valeur "upload_max_filesize" de 64Mo mais une valeur "post_max_size" de 3M, soit un conteneur plus de 20 fois plus petit que la taille maximale autorisée des fichiers qu'il peut contenir... On met habituellement une valeur de "post_max_size" au moins égale à la valeur de "upload_max_filesize", c'est pour cette raison que je ne teste que cette valeur qui est normalement la plus limitante.

Pour aller plus loin j'aurais pu donner un exemple qui contrôle aussi la valeur de "post_max_size", mais essayer de parer une mauvaise configuration serveur est une lutte sans fin et d'autres valeurs peuvent intervenir. Par exemple **certains serveurs "nginx"** ont (ou avaient) par défaut une configuration serveur "client_max_body_size" de 1Mo qui va également limiter la taille de l'upload.

En cas de problème vous pouvez définir la valeur des fragments à 1Mo, puis augmenter par pas de 1 mega.

Lors des tests si vous obtenez systématiquement le message d'erreur :
"*Connexion non valide ou perdue. Rafraîchissez la page et recharger votre fichier, si celui-ci dispose d'une sauvegarde automatique elle sera utilisée.*"
Deux erreurs sont possibles :
- Les sessions ne fonctionnement pas (très rare).
- Plus probablement la taille des fragments est supérieure à ce que peut supporter le serveur. Appliquez alors une modification de la taille des fragments comme indiqué ci-dessus ou optimisez la configuration de votre serveur.
Si la valeur de "post_max_size" est dépassée alors les super globales $_POST et $_FILES sont nulles. Le jeton de contrôle du formulaire sera invalide puisqu'il transite dans une variable $_POST. D'où le message d'erreur renvoyé par le script php d'upload. Ce message sera pertinent en production car une fois bien configuré le principal problème possible constaté est une perte de connexion qui peut intervenir après plusieurs heures d'uload en continu sur un serveur mutualisé d'entrée de gamme (sans doute une limitation pour économiser des ressources serveur).