

PHP 4 et 5 : l'extension ZIP

par [julp](#) ([Autres articles](#))

Date de publication : 28/04/2007

Dernière mise à jour :

Le format ZIP fait indéniablement partie des standards en matière de compression. Il mérite donc que nous nous y attardions au travers d'un article d'autant plus que son support varie d'une version de PHP à une autre.

1 - PHP4

1.1 - Installation

1.1.1 - Windows

1.1.2 - Linux

1.1.2.1 - Statique

1.1.2.2 - Dynamique

1.2 - Utilisation

1.2.1 - Lister le contenu d'une archive

1.2.2 - Extraire une archive

2 - PHP5

2.1 - Installation

2.1.1 - Windows

2.1.2 - Linux/Unix

2.1.2.1 - Statique

2.1.2.2 - Dynamique

2.2 - Aperçu de la classe ZipArchive

2.3 - Utilisation de la classe ZipArchive

2.3.1 - Créer une archive

2.3.2 - Ajouter/Modifier un fichier à une archive

2.3.3 - Supprimer un fichier d'une archive

2.3.4 - Lister une archive

2.3.5 - Obtenir des informations sur un fichier particulier de l'archive

2.3.6 - Extraire une archive

2.4 - Lecture d'une archive comme d'un flux

3 - Conclusion

3.1 - Epilogue

3.2 - Remerciements

1 - PHP4

L'extension Zip pour ces versions de PHP est limitée à l'accès en lecture sur l'archive. Vous ne pouvez en outre que lire le contenu d'une archive et obtenir diverses informations sur les fichiers contenus dans celle-ci ou encore les extraire mais en aucun cas en créer ou modifier une.

1.1 - Installation

1.1.1 - Windows

Récupérez la **version binaire de PHP pour Windows** ou uniquement cette extension précise sur le site **PECL4WIN**. Placez la avec les autres (répertoire souvent nommé *extensions*) puis chargez l'extension dans le fichier `php.ini` à l'aide de la ligne suivante :

```
extension=php_zip.dll
```

Un redémarrage de votre serveur Apache sera requis afin de prendre en charge immédiatement cette nouvelle extension.

1.1.2 - Linux

Réservez l'installation de PHP à partir de ses sources à des besoins particuliers surtout si vous êtes débutants dans ce domaine, privilégiez dans la mesure du possible les paquetages binaires officiels mis à votre disposition par le distributeur de votre système Linux auquel cas il ne devrait vous rester qu'à charger la nouvelle extension en éditant le fichier `php.ini` (référez vous à la partie intitulée Dynamique ci-dessous pour cela).

Sur une distribution Mandriva avec des médias correctement renseignés, la commande **urpmi php4-zip** suffira.

La librairie **ZZIPLib** (et ses fichiers d'en-tête), version minimale 0.10.6, vous sera requise.

1.1.2.1 - Statique

L'extension ZIP étant intégrée aux sources de PHP, il nous suffit de manifester notre intérêt en ajoutant l'option `--with-zip` au script de configuration :

```
./configure --prefix=/usr/local/php4 ... --with-zip  
make  
make install
```

1.1.2.2 - Dynamique

Le recours à une compilation dynamique peut se faire après la compilation du coeur de PHP vous évitant de repasser par une recompilation complète de PHP mais permet également de mettre à jour l'un ou l'autre de façon plus ou moins indépendante. Les commandes à saisir sont alors les suivantes :

```
cd /répertoire/des/sources/de/l/extension
phpize
./configure
make
make install
```

Il est nécessaire d'indiquer à PHP de charger l'extension ZIP en éditant le fichier `php.ini` pour y ajouter la ligne suivante :

```
extension=zip.so
```

La directive `extension_dir` du même fichier doit être correcte sous peine d'erreurs car PHP sera alors incapable de trouver la librairie. Si PHP se présente en tant que module du serveur web, il vous sera nécessaire de redémarrer ce dernier afin de disposer du support de l'extension ZIP.

1.2 - Utilisation

1.2.1 - Lister le contenu d'une archive

Obtenir la liste des entrées d'une archive ZIP se résume dans un premier temps à l'ouvrir puis à en parcourir les entrées et pour terminer à la fermer.

L'ouverture se fait à l'aide de la fonction `zip_open` qui n'a pour seul paramètre le nom de l'archive et qui vous renverra `FALSE` en cas d'erreur ou alors une ressource qui vous permettra de l'explorer par la suite.

Le parcours des entrées de l'archive se fait une à une par le biais de la fonction `zip_read`, qui fournit pour chaque entrée une ressource décrivant celle-ci ou `FALSE` si la fin de l'archive a été atteinte. Il est possible d'exploiter cette ressource afin d'obtenir diverses informations sur le fichier compressé à raison d'une fonction par critère :

- `zip_entry_name()` : le nom du fichier
- `zip_entry_compressedsize()` : la taille du fichier après compression
- `zip_entry_compressionmethod()` : la méthode de compression utilisée
- `zip_entry_filesize()` : taille non compressée du fichier

N'omettons pas après cela de refermer l'archive avec la fonction `zip_close`.

Puisqu'un exemple vaut toujours mieux qu'un long discours :

```
<style type="text/css">
<!--
table.zip_details {
    border: 3px double black;
    border-collapse: collapse;
}

table.zip_details td,
table.zip_details th {
    border: 1px solid black;
}
```

```

table.zip_fichiers td,
table.zip_fichiers th {
    border: 0px none;
}
-->
</style>

<?php
function formater_taille($taille) {
    $unites = array('o', 'ko', 'Mo', 'Go');

    for ($u = count($unites); $u >= 0; $u--) {
        if (isset($unites[$u]) && $taille >= 1024 * pow(1024, $u - 1)) {
            $taille = $taille / pow(1024, $u);
            $unite = $unites[$u];
            break;
        }
    }

    if ($u > 0) {
        return number_format($taille, 2, ',', ' ') . ' ' . $unite;
    } else {
        return $taille . ' ' . $unite;
    }
}

function afficher_zip($archive) {
    if (($zip = zip_open($archive)) === FALSE) {
        return FALSE;
    }
    echo '<table class="zip_details">';
    echo '<tr><th colspan="2">' . $archive . '</th></tr>';
    echo '<tr><td>Taille :</td><td>' . formater_taille(filesize('sources.zip')) . '</td></tr>';
    $nbEntrees = 0;
    echo '<tr>
    <td>Fichiers archivés :</td>
    <td><table class="zip_fichiers">
    <tr>
        <th>Nom</th>
        <th>Taille compressée</th>
        <th>Taille non compressée</th>
    </tr>';
    while ($entree = zip_read($zip)) {
        echo '<tr>
        <td>' . zip_entry_name($entree) . '</td>
        <td align="center">' . formater_taille(zip_entry_compressedsize($entree)) . '</td>
        <td align="center">' . formater_taille(zip_entry_filesize($entree)) . '</td>
        </tr>';
        $nbEntrees++;
    }
    echo '</table></td></tr>';
    echo '<tr><td>Nombre de fichiers archivés :</td><td>' . $nbEntrees . '</td></tr>';
    echo '</table>';
    zip_close($zip);
    return TRUE;
}

# Exemple d'utilisation
afficher_zip('sources.zip');
?>

```

C'est la fonction `afficher_zip` qu'il faut regarder et comprendre. Le restant, la CSS et la fonction `formater_taille`, ne sont que secondaires et permettent d'améliorer quelque peu l'affichage généré par `afficher_zip()`.

1.2.2 - Extraire une archive

Cette opération implique peu de changements et de nouvelles fonctions par rapport au listing d'une archive. En effet, l'archive est parcourue de la même manière et les seules nouvelles fonctions que nous allons découvrir ont pour but d'accéder au contenu d'une entrée et de la lire :

- `zip_entry_open(zip, entrée)` : ouvrir en lecture le fichier correspondant à la ressource *entrée* (obtenue par la fonction `zip_read` de l'archive représentée par la ressource *zip* (résultat de la fonction `zip_open`)
- `zip_entry_read(entrée, longueur)` : lire le contenu de l'*entrée* (ressource retournée par `zip_read`) à hauteur de *longueur* octets (valeur par défaut 1024).
- `zip_entry_close(entrée)` : libère les ressources pour la lecture du fichier correspondant à la ressource *entrée*

Ces fonctions bien que portant des noms différents, se rapprochent fortement de `fopen`, `fread` couplée à `filesize` (afin de récupérer le contenu du fichier en une fois) et `fclose`.

```
function mkdir_recuratif($dir) {
    $parties = preg_split('#/' . preg_quote(DIRECTORY_SEPARATOR) . '#', $dir, -1,
    PREG_SPLIT_NO_EMPTY);
    $base = '';
    foreach ($parties as $p) {
        if (!file_exists($base . $p)) {
            mkdir($base . $p);
        }
        $base .= $p . DIRECTORY_SEPARATOR;
    }
}

function extractTo($archive, $destination, $ecrase = FALSE, $fichiers = NULL) {
    if (($zip = zip_open($archive)) === FALSE) {
        die(var_dump($zip));
        return FALSE;
    }
    if (!file_exists($destination)) {
        mkdir_recuratif($destination);
    }
    while ($entree = zip_read($zip)) {
        $fichier = zip_entry_name($entree);
        if (is_array($fichiers) && !in_array($fichier, $fichiers)) {
            continue;
        }
        if (zip_entry_open($zip, $entree)) {
            $contenu = zip_entry_read($entree, zip_entry_filesize($entree));
            zip_entry_close($entree);
            if ($ecrase || !file_exists($destination . DIRECTORY_SEPARATOR . $fichier)) {
                if (strpos($fichier, '/') !== FALSE) {
                    mkdir_recuratif($destination . DIRECTORY_SEPARATOR . dirname($fichier));
                }
                $fp = fopen($destination . DIRECTORY_SEPARATOR . $fichier, 'w');
                fwrite($fp, $contenu);
                fclose($fp);
            }
        } else {
            zip_close($zip);
            return FALSE;
        }
    }
    zip_close($zip);
    return TRUE;
}
```

```
# Exemples d'utilisation
# Extraction de l'ensemble des fichiers qui composent l'archive
extractTo('sources.zip', '/home/julp/www/zip');
# Extraction limitée aux fichiers fichier1 et fichier2. S'ils existent déjà ils seront écrasés.
```

```
extractTo('mon_archive.zip', '/home/julp', TRUE, array('fichier1', 'fichier2'));
```

2 - PHP5

2.1 - Installation

2.1.1 - Windows

Téléchargez la **version binaire pour Windows** ou allez sur la page **PECL4WIN**, puis placez cette extension avec les autres (répertoire `ext` en général) et enfin activez l'extension dans le fichier `php.ini` :

```
extension=php_zip.dll
```

Redémarrez finalement votre serveur Apache pour à présent profiter du support de l'extension ZIP.

2.1.2 - Linux/Unix

L'étape de compilation ne concerne uniquement une installation de PHP à partir de ses sources qui s'adresse aux initiés pour répondre à des besoins particuliers (application de patches par exemple). Dans les autres cas, il est recommandé d'employer les paquetages fournis par votre système ou distribution auquel cas vous devriez simplement avoir besoin d'activer l'extension via votre fichier `php.ini` (voir ci-dessous dans la partie "Dynamique").

Par exemple l'installation sur la distribution Mandriva se fait avec la commande **urpmi php5-zip** si vos médias sont convenablement configurés.

Vous devez disposer de la librairie `zlib` (librairies et fichiers d'en-têtes) avant d'aller plus loin.

2.1.2.1 - Statique

Ceux qui utilisent une version antérieure à 5.2.0 doivent, au préalable, effectuer quelques opérations supplémentaires car l'extension Zip n'était pas fournie avec PHP :

- 1 Télécharger les sources de cette extension sur le site **PECL**.
- 2 Les extraire dans le répertoire `ext` des sources de PHP :

```
tar xzf zip-<version> -C /usr/local/src/php-5.X.Y/ext  
ln -s /usr/local/src/php-5.X.Y/ext/zip-<version> /usr/local/src/php-5.X.Y/ext/zip
```

- 3 Les intégrer pour la compilation :

```
cd /usr/local/src/php-5.X.Y  
./buildconf --force
```

Démarrez la compilation en ajoutant l'option `--enable-zip` lors de la configuration :

```
./configure --prefix=/usr/local/php5 ... --enable-zip
```

```
make
make install
```

2.1.2.2 - Dynamique

L'avantage de cette méthode c'est que vous n'avez pas besoin de recompiler PHP. Vous compilez en effet, sous forme dynamique, donc autonome, l'extension Zip. Nous aurons besoin des sources de celle-ci si elles ne sont pas déjà incluses à PHP. Si tel n'est pas le cas, je vous invite à les télécharger du site **PECL** puis à les décompresser. Vous voilà prêts à les compiler :

```
cd /répertoire/des/sources/de/l/extension
phpize
./configure
make
make install
```

Pour terminer, il faut indiquer à PHP de charger cette extension, c'est pourquoi nous rajouterons la ligne suivante à notre fichier de configuration php.ini :

```
extension=zip.so
```

Profitez-en pour vérifier que les chemins vers ces extensions sont correctement renseignés à la directive *extension_dir*. Redémarrez Apache pour prendre cette nouvelle extension en compte.

2.2 - Aperçu de la classe ZipArchive

En PHP 5, la manipulation d'archives est déléguée à une classe nommée ZipArchive dont vous trouverez ci-dessous sa structure documentée :

```
class ZipArchive
{
    /* Constantes */
    // Modes d'ouverture
    const CREATE;           // Crée l'archive si celle-ci n'existe pas
    const EXCL;            // Echoue si l'archive existe déjà
    const CHECKCONS;      // Effectue des contrôles supplémentaires
    const OVERWRITE;      // Remplace l'archive existante
    // Options
    const FL_NOCASE;       // La casse est ignorée sur l'entrée du nom
    const FL_NODIR;       // La partie répertoire est ignorée
    const FL_COMPRESSED;  // Lit les données compressées
    const FL_UNCHANGED;   // Utilise les données originales de l'archive (ignore les
changements)
    // Modes de compression
    const CM_DEFAULT;     // Compression par défaut (offre le meilleur taux)
    const CM_STORE;       // Stockage uniquement (sans compression)
    const CM_SHRINK;      // Une variante de l'algorithme LZW
    const CM_REDUCE_1;    // Taux de compression réduit (niveau 1)
    const CM_REDUCE_2;    // Taux de compression réduit (niveau 2)
    const CM_REDUCE_3;    // Taux de compression réduit (niveau 3)
    const CM_REDUCE_4;    // Taux de compression réduit (niveau 4)
    const CM_IMPLODE;     // Réunit les séquences identiques avant compression
    const CM_DEFLATE;     // Utilisation de l'algorithme Deflate (RFC 1951)
}
```

```
const CM_DEFLATE64; // Variante de l'algorithme Deflate travaillant sur des blocs de 64 ko
au lieu de 32
const CM_PKWARE_IMPLODE; // Méthode officielle
const CM_BZIP2; // Compression utilisant l'algorithme bzip2
// Erreurs
const ER_OK; // Aucune erreur
const ER_MULTIDISK; // Archives multi-disques non supportée
const ER_RENAME; // Erreur lors de la modification du nom du fichier temporaire
const ER_CLOSE; // Erreur lors de la fermeture de l'archive
const ER_SEEK; // Erreur de déplacement
const ER_READ; // Erreur de lecture
const ER_WRITE; // Erreur d'écriture
const ER_CRC; // Erreur dans la somme CRC
const ER_ZIPCLOSED; // L'archive ZIP est fermée
const ER_NOENT; // Fichier inexistant
const ER_EXISTS; // Le fichier est déjà présent
const ER_OPEN; // Ne peut ouvrir le fichier
const ER_TMPOPEN; // Erreur lors de la création du fichier temporaire
const ER_ZLIB; // Erreur interne liée à la librairie Zlib
const ER_MEMORY; // Erreur d'allocation dynamique de mémoire
const ER_CHANGED; // L'entrée a été modifiée
const ER_COMPNOTSUPP; // Méthode de compression non supportée
const ER_EOF; // Fin de fichier rencontrée prématurément
const ER_INVALID; // Argument invalide
const ER_NOZIP; // Ne s'agit pas d'une archive au format ZIP
const ER_INTERNAL; // Erreur interne
const ER_INCONS; // Archive inconsistante
const ER_REMOVE; // Ne peut supprimer un fichier
const ER_DELETED; // Entrée supprimée

/* Propriétés */
public $status; // Numéro de l'erreur de la librairie libzip
public $statusSys; // Numéro de l'erreur système (errno) ou de la librairie zlib
public $numFiles; // Nombre de fichiers dans l'archive
public $filename; // Nom et chemin de l'archive
public $comment; // Commentaire associé à l'archive

/* Méthodes */
/**
 * Ouvrir une archive en vue de sa création ou manipulation
 * @param fichier : le nom de l'archive à créer ou manipuler
 * @param drapeaux : mode d'ouverture de l'archive (CREATE, EXCL, CHECKCONS ou OVERWRITE)
 * @return TRUE ou un code décrivant l'erreur (constantes de classe commençant par ER_)
 */
public function open(chaîne fichier [, entier drapeaux]);

/**
 * Fermer l'archive
 * @return un booléen, TRUE en cas de succès sinon FALSE
 */
public function close();

/**
 * Ajouter un répertoire vide à l'archive
 * @param répertoire : le nom du répertoire à ajouter
 * @return un booléen, FALSE en cas d'échec et TRUE dans le cas contraire
 */
public function addEmptyDir(chaîne répertoire);

/**
 * Ajouter un fichier virtuel (ie qui n'a pas d'existence physique) à l'archive
 * @param nom_local : nom sous lequel figurera le fichier au sein de l'archive
 * @param contenu : le contenu de ce fichier
 * @return un booléen, FALSE si une erreur survient et TRUE si tout se passe normalement
 */
public function addFromString(chaîne nom_local, chaîne contenu);

/**
```

```
* Ajouter un fichier (physique) à l'archive
* @param fichier : le chemin complet vers le fichier à ajouter
* @param nom_local : son nom dans l'archive s'il doit être différent
* @param début : position dans le fichier source à partir de laquelle son contenu doit
être inséré
* @param longueur : nombre de caractères maximal à inclure
* @return un booléen, FALSE en cas d'erreur et TRUE si tout va bien
**/
public function addFile(chaîne fichier [, chaîne nom_local [, entier début [, entier
longueur]]]);

/**
* Renommer le fichier indiqué à partir de son indice
* @param indice : indice du fichier au sein de l'archive (compris entre 0 et
$this->numFiles - 1 inclus)
* @param nouveau_nom : le nouveau nom à donner à ce fichier
* @return un booléen, TRUE si aucune erreur n'est survenue et FALSE dans le cas contraire
**/
public function renameIndex(entier indice, chaîne nouveau_nom);

/**
* Renommer le fichier indiqué à partir de son nom courant
* @param nom : nom actuel du fichier à renommer
* @param nouveau_nom : son nouveau nom
* @return un booléen, TRUE si aucune erreur ne s'est produite sinon FALSE
**/
public function renameName(chaîne nom, nouveau_nom);

/**
* Change le commentaire actuel de l'archive
* @param commentaire : le nouveau commentaire à attribuer à l'archive
* @return un booléen, FALSE en cas d'échec ou bien TRUE si l'opération réussie
**/
public function setArchiveComment(chaîne commentaire);

/**
* Renvoie le commentaire actuel de l'archive
* @return ce commentaire sous la forme d'une chaîne de caractères
**/
public function getArchiveComment();

/**
* Changer le commentaire associé à une entrée de l'archive à partir de son indice
* @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1
inclus)
* @param commentaire : le nouveau commentaire attribué au fichier
* @return un booléen, FALSE en cas d'erreur sinon TRUE
**/
public function setCommentIndex(entier indice, chaîne commentaire);

/**
* Changer le commentaire d'une entrée de l'archive à l'aide de son nom
* @param nom : nom actuel du fichier dans l'archive
* @param commentaire : le nouveau commentaire à associer à ce fichier
* @return un booléen, FALSE si l'opération échoue et TRUE dans le cas contraire
**/
public function setCommentName(chaîne nom, chaîne commentaire);

/**
* Obtenir le commentaire correspondant à une entrée par son indice
* @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
* @param drapeaux : une des options (constantes de classe débutant par FL_)
* @return FALSE en cas d'erreur ou une chaîne correspondant au commentaire (éventuellement
vide s'il n'y a
* aucun commentaire)
**/
public function getCommentIndex(entier indice [, entier drapeaux]);
```

```
/**
 * Obtenir le commentaire associé à un fichier à l'aide du nom de celle-ci
 * @param nom      : nom de l'entrée cible
 * @param drapeaux : une des options (constantes de classe débutant par FL_)
 * @return FALSE en cas d'erreur ou une chaîne correspondant au commentaire (éventuellement
vide s'il n'y a
 *      aucun commentaire)
 */
public function getCommentName(chaîne nom [, entier drapeaux]);

/**
 * Supprimer une entrée de l'archive à partir de son indice
 * @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
 * @return un booléen, TRUE si la suppression a pu être effectuée sinon FALSE
 */
public function deleteIndex(entier indice);

/**
 * Supprimer un fichier de l'archive à l'aide de son nom
 * @param nom : nom de l'entrée à supprimer
 * @return un booléen, FALSE si la suppression a échoué et TRUE dans le cas contraire
 */
public function deleteName(chaîne nom);

/**
 * Obtenir les informations relatives à une entrée à l'aide de son nom
 * @param nom      : le nom de l'entrée visée
 * @param drapeaux : une des options (constantes de classe débutant par FL_)
 * @return FALSE en cas d'échec ou bien un tableau associatif de la forme :
 *      Array (
 *          'name' => nom du fichier
 *          'index' => indice de l'entrée au sein de l'archive
 *          'crc' => sa somme de contrôle
 *          'size' => taille du fichier non compressé
 *          'mtime' => timestamp représentant la date de dernière modification du
fichier
 *          'comp_size' => taille du fichier compressé
 *          'comp_method' => méthode de compression employée (voir les constantes de
classe CM_)
 *      )
 */
public function statName(chaîne nom [, entier drapeaux]);

/**
 * Obtenir les informations relatives à une entrée par son indice
 * @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
 * @param drapeaux : une des options (constantes de classe débutant par FL_)
 * @return FALSE en cas d'échec ou bien un tableau associatif de la forme :
 *      Array (
 *          'name' => nom du fichier
 *          'index' => indice de l'entrée au sein de l'archive
 *          'crc' => sa somme de contrôle
 *          'size' => taille du fichier non compressé
 *          'mtime' => timestamp représentant la date de dernière modification du
fichier
 *          'comp_size' => taille du fichier compressé
 *          'comp_method' => méthode de compression employée (voir les constantes de
classe CM_)
 *      )
 */
public function statIndex(entier indice [, entier drapeaux]);

/**
 * Obtenir l'indice d'une entrée particulière dans l'archive
 * @param nom      : le nom de cette entrée
 * @param drapeaux : une des options (constantes de classe débutant par FL_)
 * @return FALSE si une erreur est survenue ou alors l'indice de l'entrée dans l'archive
```

```
    */
    public function locateName(chaîne nom [, entier drapeaux]);

    /**
     * Obtenir le nom d'une entrée à partir de son indice
     * @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
     * @param drapeaux : une des options (constantes de classe débutant par FL_)
     * @return FALSE en cas d'erreur et l'indice du fichier dans le cas contraire
     */
    public function getNameIndex(entier indice [, entier drapeaux]);

    /**
     * Annule les modifications apportées sur l'archive (son commentaire uniquement)
     * @return un booléen, TRUE si les modifications ont pu être abrogées et FALSE si l'opération a
    échoué
    */
    public function unchangeArchive();

    /**
     * Annuler l'ensemble des modifications apportées à l'archive (ceci inclut son contenu)
     * @return un booléen, FALSE en cas d'impossibilité de revenir à l'état original de l'archive
    sinon TRUE
    */
    public function unchangeAll();

    /**
     * Annuler toutes les modifications apportées au fichier désigné par son indice
     * @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
     * @return un booléen, TRUE si l'annulation a pu être rendue effective et FALSE dans le cas
    contraire
    */
    public function unchangeIndex(entier indice);

    /**
     * Annuler toutes les modifications apportées sur l'entrée indiquée par son nom
     * @param nom : nom du fichier concerné par cette annulation
     * @return un booléen, TRUE si l'opération a pu être menée à bien et FALSE en cas d'erreur
    */
    public function unchangeName(chaîne nom);

    /**
     * Extraire une archive, partiellement ou non
     * @param destination : emplacement sur le système de fichiers où l'extraction aura lieu
     * @param entrées : noms des seules entrées à extraire. S'il y en a qu'une il est possible
    d'utiliser
     *
     * une chaîne de caractères plutôt qu'un tableau
     * @return un booléen, TRUE pour le bon déroulement de l'opération et FALSE si une erreur est
    survenue
    */
    public function extractTo(chaîne destination [, variable entrées]);

    /**
     * Récupérer le contenu d'une entrée à partir de son nom
     * @param nom : nom de l'entrée dont on souhaite obtenir son contenu
     * @param longueur : spécifie le nombre de caractères maximal à renvoyer (une valeur < 1 a pour
    effet de
     *
     * rendre cette fonctionnalité inopérante)
     * @param drapeaux : une des options (constantes de classe débutant par FL_)
     * @return FALSE si une erreur est survenue ou alors le contenu du fichier
    */
    public function getFromName(chaîne nom [, entier longueur [, entier drapeaux]]);

    /**
     * Récupérer le contenu d'une entrée à l'aide de son indice
     * @param indice : indice du fichier concerné (compris entre 0 et $this->numFiles - 1 inclus)
     * @param longueur : spécifie le nombre de caractères maximal attendu (une valeur < 1 a pour
    effet de
     *
     * rendre cette fonctionnalité inopérante)
    */

```

```
* @param drapeaux : une des options (constantes de classe débutant par FL_)
* @return FALSE en cas d'échec sinon le contenu du fichier sous la forme d'une chaîne de
caractères
**/
public function getFromIndex(entier indice [, entier longueur [, entier drapeaux]]);

/**
* Obtenir un descripteur de fichier sur une entrée de l'archive
* @param nom : nom de l'entrée
* @return FALSE en cas d'erreur ou alors une ressource
**/
public function getStream(chaîne nom);
}
```

Quelles que soient vos intentions, il faut commencer par instancier un nouvel objet de type ZipArchive :

```
$zip = new ZipArchive();
```

Puis nous devons l'ouvrir, que ce soit pour créer une nouvelle archive ou bien en ouvrir une existante :

```
if ($zip->open("nom_de_l'archive", "optionnel : mode de création") !== TRUE) {
    die("Impossible d'ouvrir l'archive demandée");
}
```

Les modes d'ouverture peuvent être les suivants :

- ZipArchive::CREATE : l'archive est créée si celle-ci n'existe pas
- ZipArchive::OVERWRITE : écrase l'archive existante (crée toujours une nouvelle archive)
- ZipArchive::EXCL : l'ouverture échoue si une archive du même nom est déjà présente
- ZipArchive::CHECKCONS : effectue des contrôles supplémentaires

A ce moment-là vous pouvez manipuler votre archive à l'aide des fonctions prévues à cet effet puis ...

Terminez enfin en fermant l'archive :

```
$zip->close();
```

Si cette dernière action n'est pas faite explicitement lors de votre script, elle y sera automatiquement procédée en fin de votre script par PHP.

2.3 - Utilisation de la classe ZipArchive

2.3.1 - Créer une archive

La création d'une nouvelle archive débute par l'instanciation d'un nouvel objet ZipArchive sur lequel on applique la méthode open. Open attend au moins un paramètre correspondant au nom de l'archive à créer, dans notre cas, et optionnellement son mode d'ouverture/création. Le mode de création particulier ZipArchive::OVERWRITE, utilisé plus bas, permet de recréer l'archive au lieu de la modifier si elle existe déjà. Cette méthode présente également

une particularité au niveau de la valeur retournée : un entier en cas d'erreur la décrivant et la valeur vraie (TRUE) si l'opération réussie.

Nous ajoutons ensuite les fichiers un par un au fichier zip que nous allons créer avec la fonction `addFile`. Son utilisation est simple puisqu'elle attend en paramètre le chemin du fichier à joindre à l'archive dont on peut éventuellement changer le nom en spécifiant son deuxième paramètre. Comme la majorité des méthodes de cette classe, une valeur booléenne indiquant le succès ou non de l'action vous sera renvoyée.

```
function creer_archive($nom, $fichiers, $commentaire = '')
{
    if (is_array($fichiers)) {
        $zip = new ZipArchive();
        if ($zip->open($nom, ZIPARCHIVE::OVERWRITE) !== TRUE) {
            return FALSE;
        }
        foreach ($fichiers as $k => $f) {
            if (!$zip->addFile($f)) {
                return FALSE;
            }
            if (is_string($k)) {
                $zip->setCommentName($f, $k);
            }
        }
        if ($commentaire) {
            $zip->setArchiveComment($commentaire);
        }
        return $zip->close();
    }
    return FALSE;
}

# Parcours du répertoire courant à la recherche des fichiers php qui constitueront la liste des
# fichiers à zipper
$repertoire = '.';
$fichiers = array();
$dir = opendir($repertoire);
while (($file = readdir($dir)) !== FALSE) {
    if ($file == '.' or $file == '..') {
        continue;
    }
    if (preg_match('/\.[php45]?$/i', $file)) {
        $fichiers[] = $file;
    }
}
closedir($dir);

creer_archive('sources.zip', $fichiers, "Les sources du tutoriel portant sur l'extension ZIP")
or die("Echec lors de la création de l'archive");
```

Les méthodes `setArchiveComment` et `setCommentName` permettent respectivement d'apposer un commentaire sur l'archive elle-même et sur les différentes entrées. Il semble que les principaux logiciels commerciaux supportant ce type d'archive ne permettent pas d'exploiter les commentaires au niveau des fichiers.


Ci-dessous une recette générique et prête à l'emploi ayant pour but d'empaqueter récursivement toute une partie de votre système de fichiers. Cette fonction ne gèrera pas l'ouverture ni la fermeture de l'archive mais vous offre en contrepartie la possibilité d'ajouter avant ou après d'autres fichiers en plus.

```
function zip_recuratif($chemin, $zip, $prefixe = '')
```

```

{
    if (substr($chemin, -1, 1) != DIRECTORY_SEPARATOR) {
        $chemin .= DIRECTORY_SEPARATOR;
    }
    if (file_exists($chemin)) {
        if (is_dir($chemin)) {
            if (!$dh = opendir($chemin)) {
                return FALSE;
            }
            while (($fichier = readdir($dh)) != FALSE) {
                if ($fichier == '.' || $fichier == '..') {
                    continue;
                }
                if (is_dir($chemin . $fichier)) {
                    $zip->addEmptyDir($prefixe . $fichier);
                    if (!zip_recuratif($chemin . $fichier . DIRECTORY_SEPARATOR, $zip, $prefixe .
                    $fichier . DIRECTORY_SEPARATOR)) {
                        return FALSE;
                    }
                } else {
                    if (!$zip->addFile($chemin . $fichier, $prefixe . $fichier)) {
                        return FALSE;
                    }
                }
            }
            closedir($dh);
            return TRUE;
        } else {
            if (!$zip->addFile($chemin)) {
                return FALSE;
            }
            return TRUE;
        }
    }
    return FALSE;
}

# Exemple d'utilisation
$zip = new ZipArchive();
if ($zip->open('sources.zip', ZipArchive::OVERWRITE) != TRUE) {
    die("Impossible de créer l'archive");
}
zip_recuratif('/home/julp/www/zip', $zip);
$zip->close();
    
```

Puisque nous développons en PHP 5, il nous est possible de regrouper ces deux fonctions au sein d'une classe surchargeant la classe native ZipArchive mais également de gérer les éventuelles erreurs par des  **exceptions** :

```

class MyZip extends ZipArchive
{
    public function __construct($nom, $commentaire = '')
    {
        if ($this->open($nom, ZIPARCHIVE::OVERWRITE) != TRUE) {
            throw new Exception("L'archive ne peut être créée");
        }

        if ($commentaire) {
            $this->setArchiveComment($commentaire);
        }
    }

    public function __destruct()
    {
        $this->close();
    }
}
    
```

```
}

public function addFiles(Array $fichiers)
{
    foreach ($fichiers as $k => $f) {
        if (!$this->addFile($f)) {
            throw new Exception("Le fichier '$f' n'a pu être ajouté à l'archive");
        }

        if (is_string($k)) {
            $this->setCommentName($f, $k);
        }
    }
}

public function addRecursive($chemin, $prefixe = '')
{
    $chemin = realpath($chemin) . DIRECTORY_SEPARATOR;

    if (!file_exists($chemin)) {
        throw new Exception("Le chemin '$chemin' n'existe pas");
    }

    if (!is_dir($chemin)) {
        throw new Exception("Le chemin '$chemin' existe mais n'est pas un répertoire");
    }

    if (!$dh = opendir($chemin)) {
        throw new Exception("Le répertoire '$chemin' n'est pas accessible");
    }

    while (($fichier = readdir($dh)) !== FALSE) {
        if ($fichier == '.' || $fichier == '..') {
            continue;
        }
        if (is_dir($chemin . $fichier)) {
            $this->addEmptyDir($prefixe . $fichier);
            $this->addRecursive($chemin . $fichier . DIRECTORY_SEPARATOR, $prefixe . $fichier .
            DIRECTORY_SEPARATOR);
        } else {
            if (!$this->addFile($chemin . $fichier, $prefixe . $fichier)) {
                throw new Exception("Le fichier '$chemin/$fichier' n'a pu être ajouté à
                l'archive");
            }
        }
    }
    closedir($dh);
}
}
```

Un exemple d'utilisation possible serait :

```
require_once('/var/www/offline/librairies/MyZip.php');

$repertoire = '.';
$fichiers = array();
$dir = opendir($repertoire);
while (($file = readdir($dir)) !== FALSE) {
    if ($file == '.' or $file == '..') {
        continue;
    }
    if (preg_match('/\.php[45]?$/i', $file)) {
        $fichiers[] = $file;
    }
}
}
```

```
closedir($dir);

try {
    $sources = new MyZip('sources.zip', "Les sources du tutoriel portant sur l'extension ZIP");
    $sources->addFiles($fichiers);
    $sources->addRecursive('/home/julp/www/zip');
} catch (Exception $e) {
    die("Echec lors de la création de l'archive : " . $e->getMessage());
}
```

2.3.2 - Ajouter/Modifier un fichier à une archive

Les opérations d'ajout ou de modification d'un fichier bien qu'étant différentes ne se distinguent que par la présence ou non du fichier dans l'archive. En d'autres termes, si l'entrée désignée existe déjà au sein de l'archive, celle-ci sera remplacée sans sommation.

En pratique, il y a peu de changements par rapport à la création de l'archive si ce n'est au niveau du mode d'ouverture de cette dernière. Veillez en particulier à ne pas utiliser ZipArchive::OVERWRITE (vous en créeriez une nouvelle écrasant ainsi l'existante) en revanche ZipArchive::CREATE peut être usité sans crainte car il permettra de créer l'archive si celle-ci n'existe pas au préalable.

```
$zip = new ZipArchive();
if ($zip->open('sources.zip') !== TRUE) {
    die("Echec lors de l'ouverture de l'archive");
}
$zip->addFromString('README', "Le tutoriel associé se trouve à l'adresse suivante :
http://julp.developpez.com/php/zip/");
    or die("Impossible de rajouter un fichier à l'archive");
$zip->close() or die("Erreur lors de la fermeture de l'archive");
```

Pour varier parmi les différents exemples donnés dans cet article, la méthode addFromString a été ici employée afin d'ajouter de toute pièce (à partir d'une chaîne) un nouveau fichier mais sa consœur, addFile, aurait tout aussi bien convenu.

2.3.3 - Supprimer un fichier d'une archive

La première étape, incontournable, consiste à ouvrir l'archive. S'en suit la suppression que l'on accomplit avec la méthode deleteName. Comme l'indique son nom, la suppression sera effectuée à l'aide du nom complet de l'entrée et retournera un booléen décrivant l'aboutissement ou non de l'opération.

```
$zip = new ZipArchive();
if ($zip->open('sources.zip') !== TRUE) {
    die("Echec lors de l'ouverture de l'archive");
}
$zip->deleteName('README') or die("Cette entrée n'a pu être supprimé");
$zip->close() or die("Erreur lors de la fermeture de l'archive");
```

La méthode alternative deleteIndex vous permettra de supprimer une entrée à partir de la position qu'elle occupe dans l'archive.

2.3.4 - Lister une archive

C'est le strict équivalent de l'implémentation précédemment codé pour PHP 4 où la classe ZipArchive remplace les fonctions Zip. Par ailleurs, elle nous permet d'obtenir quelques informations supplémentaires sur les entrées de l'archive comme leurs dates de dernière modification.

Après avoir ouvert l'archive, nous la parcourons sur la base des indices des différents fichiers qu'elle contient (de 0 à la valeur de son attribut numFiles moins un). La méthode statIndex permet l'obtention des informations de l'entrée située à la position indiquée sous la forme d'un tableau associatif dont les différentes clés sont :

- name : le nom du fichier
- index : l'indice de l'entrée au sein de l'archive
- crc : sa somme de contrôle
- size : la taille du fichier non compressé
- mtime : un timestamp représentant la date de dernière modification du fichier (à reformater avec la fonction date si besoin)
- comp_size : la taille du fichier après compression
- comp_method : la méthode de compression employée (voir les constantes de classe CM_*)

Les différents commentaires seront affichés par la méthode getArchiveComment en qui concerne celui qui est accolé à l'archive et getCommentIndex pour ceux qui décrivent les différentes entrées.

Ne vous souciez pas de la CSS ou encore de la fonction formater_taille dont le rôle est d'augmenter la lisibilité de l'affichage résultant. Tout ce qui concerne l'archive traitée est effectué par la fonction afficher_zip.

```
<style type="text/css">
<!--
table.zip_details {
    border: 3px double black;
    border-collapse: collapse;
}

table.zip_details td,
table.zip_details th {
    border: 1px solid black;
}

table.zip_fichiers td,
table.zip_fichiers th {
    border: 0px none;
}
-->
</style>

<?php
function formater_taille($taille) {
    $unites = array('o', 'ko', 'Mo', 'Go');

    for ($u = count($unites); $u >= 0; $u--) {
        if (isset($unites[$u]) && $taille >= 1024 * pow(1024, $u - 1)) {
            $taille = $taille / pow(1024, $u);
            $unite = $unites[$u];
            break;
        }
    }

    if ($u > 0) {
        return number_format($taille, 2, ',', ' ') . ' ' . $unite;
    } else {

```


```

        return $taille . ' ' . $unite;
    }
}

function afficher_zip($archive) {
    $zip = new ZipArchive();
    if ($zip->open($archive) !== TRUE) {
        return FALSE;
    }
    echo '<table class="zip_details">';
    echo '<tr><th colspan="2">' . $archive . '</th></tr>';
    $nbEntrees = $zip->numFiles;
    echo '<tr><td>Nombre de fichiers archivés :</td><td>' . $nbEntrees . '</td></tr>';
    echo '<tr><td>Taille :</td><td>' . formater_taille(filesize('sources.zip')) . '</td></tr>';
    if ($commentaire = $zip->getArchiveComment()) {
        echo '<tr><td>Commentaire :</td><td>' . $commentaire . '</td></tr>';
    }
    if ($nbEntrees > 0) {
        echo '<tr>
        <td>Fichiers archivés :</td>
        <td><table class="zip_fichiers">
        <tr>
            <th>Nom</th>
            <th>Taille compressée</th>
            <th>Taille non compressée</th>
            <th>Dernière modification</th>
            <th>Commentaire</th>
        </tr>';
        for ($i = 0; $i < $nbEntrees; $i++) {
            $entree = $zip->statIndex($i);
            $commentaire = $zip->getCommentIndex($i);
            echo '<tr>
            <td>' . $entree['name'] . '</td>
            <td align="center">' . formater_taille($entree['comp_size']) . '</td>
            <td align="center">' . formater_taille($entree['size']) . '</td>
            <td align="center">' . date('d/m/Y', $entree['mtime']) . '</td>
            <td>' . (empty($commentaire) ? '-' : $commentaire) . '</td>
            </tr>';
        }
        echo '</table></td></tr>';
    }
    echo '</table>';
    if (!$zip->close()) {
        return FALSE;
    }
    return TRUE;
}

# Exemple d'utilisation
afficher_zip('sources.zip');
?>

```

 *En théorie, le code équivalent donné précédemment pour les versions PHP 4, bien qu'un peu moins complet est compatible avec les versions 5.*

2.3.5 - Obtenir des informations sur un fichier particulier de l'archive

Voyons maintenant comment atteindre directement l'entrée voulue d'une archive. Après l'instanciation de l'objet ZipArchive puis l'ouverture de l'archive correspondante, nous demandons les informations relatives au fichier souhaité par la méthode statName qui échouera (valeur FALSE renvoyée) si le fichier qui lui est passé en tant que premier paramètre n'existe pas ou un tableau associatif le décrivant (voir la documentation ou la partie précédente pour avoir des détails sur la forme de cette structure). Profitons-en pour voir la méthode getFromName qui à partir du nom d'une entrée (son premier paramètre) vous en renverra son contenu (FALSE en cas d'erreur).

```

<?php
define('ARCHIVE', 'sources.zip');

$entree = isset($_POST['entree']) ? $_POST['entree'] : '';
?>

<form method="POST">
    Fichier à chercher : <input type="text" name="entree" value="<?php echo $entree; ?>" />
    <br/>
    <input type="submit" value="Valider" />
</form>

<?php
if (!empty($entree)) {
    $zip = new ZipArchive();
    if ($zip->open(ARCHIVE) !== TRUE) {
        die("Ouverture impossible de l'archive");
    }
    if (($stat = $zip->statName($entree, ZipArchive::FL_NOCASE)) === FALSE) {
        echo "Le fichier demandé n'est pas présent dans l'archive";
    } else {
        echo sprintf("%s, d'une taille de %d octets, a été modifié pour la dernière fois le %s :",
            $stat['name'], $stat['size'], date('d/m/Y', $stat['mtime']));
        echo '<pre>' . $zip->getFromName($entree, 0, ZipArchive::FL_NOCASE) . '</pre>';
    }
    $zip->close();
}
?>
    
```


Cet exemple possède, j'en conviens, un intérêt limité mais a pour but d'illustrer le recours aux options (ZipArchive::FL_*) puisque ici la recherche sera insensible à la casse au niveau du nom de l'entrée et de montrer qu'il est possible de récupérer directement les informations relatives à un fichier connu.

2.3.6 - Extraire une archive

Une fonction (méthode) native a été prévue à l'inverse de PHP 4. Elle se nomme `extractTo` et son premier paramètre correspond à l'emplacement de votre arborescence où l'archive doit être dépaquetée. Cette tâche se résume dès lors en quelques instructions :

```


$zip = new ZipArchive();
if ($zip->open('sources.zip') !== TRUE) {
    die("Echec lors de l'ouverture de l'archive");
}
$zip->extractTo('/home/julp/www/zip') or die("Erreur rencontrée lors de l'extraction de
l'archive");
$zip->close() or die("Erreur lors de la fermeture de l'archive");
    
```

 **Mise en garde** : cette fonction écrase sans préavis les fichiers déjà présents sur votre système.

En jouant sur le deuxième paramètre optionnel de la méthode `extractTo`, vous pouvez spécifier la liste des fichiers qui doivent être extraits de l'archive. Cette liste prend la forme d'un tableau voir une chaîne si celle-ci n'en comprend qu'un seul :

```

$zip = new ZipArchive();
if ($zip->open('sources.zip') !== TRUE) {
    die("Echec lors de l'ouverture de l'archive");
}
$zip->extractTo('/home/julp/www/zip', array('fichier1', 'fichier2'))
    or die("Erreur rencontrée lors de l'extraction de l'archive");
$zip->close() or die("Erreur lors de la fermeture de l'archive");
    
```

 Théoriquement il vous est possible de réutiliser le code donné dans la partie dédiée aux versions PHP 4.

2.4 - Lecture d'une archive comme d'un flux

Vous saviez probablement déjà qu'il est possible de lire une page distante en ayant recours au protocole HTTP, par exemple. Il suffisait alors dans le cas présent de fournir l'URL de la page à lire à des fonctions comme `fopen` ou encore `file_get_contents`. Ainsi quelque soit la méthode ou le protocole sous-jacent pour atteindre ce fichier les fonctions pour la lecture de celui-ci restent identiques.

Et bien il est possible de faire de même avec une archive ZIP. Le chemin à employer reprenant la forme d'une URL où le protocole est `zip://` suivie du chemin complet de l'archive et enfin on trouve l'entrée ciblée sous la forme d'une ancre. Récapitulons la structure de cette URL atypique : `zip://chemin_complet_vers_l'archive#entrée_ciblée`

Exemple : nous plaçons dans le même répertoire que notre archive, nommée `sources.zip`, le script suivant qui a pour but d'extraire (puis afficher) le contenu de l'entrée REAME :

```

if (!in_array('zip', stream_get_wrappers())) {
    die("La prise en charge des flux ZIP n'est pas disponible");
}

$contentu_fichier = file_get_contents('zip://' . dirname(__FILE__) . DIRECTORY_SEPARATOR .
'sources.zip#README');
echo $contentu_fichier;
    
```

Cet exemple est réducteur car toutes les fonctions PHP pouvant acquérir des données d'un fichier deviennent alors capables d'utiliser une archive comme flux d'entrée. Citons en quelques unes comme les fonctions commençant par `imagecreatefrom` de l'extension GD, la méthode `open` de la classe `XMLReader` (extension éponyme), etc.

Prenons alors un deuxième exemple plus évocateur employant l'extension SimpleXML. Voici le fichier XML qui nous servira d'exemple et qui sera empaqueté sous le nom de `config.xml` dans une archive intitulée `parametres.zip` :

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>
  <directive nom="bdd_hote">localhost</directive>
  <directive nom="bdd_login">julp</directive>
  <directive nom="bdd_mdp"></directive>
  <directive nom="bdd_nom">db_julp</directive>
</configuration>
    
```

Nous allons le parser avec l'extension SimpleXML sans l'extraire afin de créer un tableau associatif comportant un ensemble des directives de configuration et leurs valeurs :

```
$sxml = simplexml_load_file(rawurlencode('zip://' . dirname(__FILE__) . DIRECTORY_SEPARATOR .
'parametres.zip#config.xml'));
$parametres = array();
foreach ($sxml->directive as $directive) {
    $parametres[(string) $directive['nom']] = (string) $directive;
}

echo '<pre>';
print_r($parametres);
echo '</pre>';
```





3 - Conclusion

3.1 - Epilogue

Si on regarde la documentation, nous sommes en droit de penser que les fonctions PHP 4 sont toujours fonctionnelles avec les versions PHP 5. Personnellement, je n'ai eu aucun problème de compatibilité sur mon système Linux mais en ce qui concerne Windows le constat est négatif même après avoir testé plusieurs versions de PHP et de cette extension.

L'extension PHP 5 subi de manière régulière quelques changements, ce qui m'amène à penser qu'elle évoluera petit à petit et très probablement en fonction des besoins des développeurs PHP.

Liens Developpez :

-  [Un exemple concret d'utilisation : lecture des fichiers OpenXML avec PHP 5](#)
-  [Compression ZIP avec la classe incluse dans phpMyAdmin](#)
-  [La compression à l'aide de l'extension bz2](#)
-  [Compression avec l'extension zlib](#)

Liens externes :

-  [Documentation officielle de l'extension ZIP](#)
- [Site de la librairie PclZip](#)

3.2 - Remerciements

J'adresse mes plus vifs remerciements à **Yogui** pour ses conseils avisés et sa relecture.

