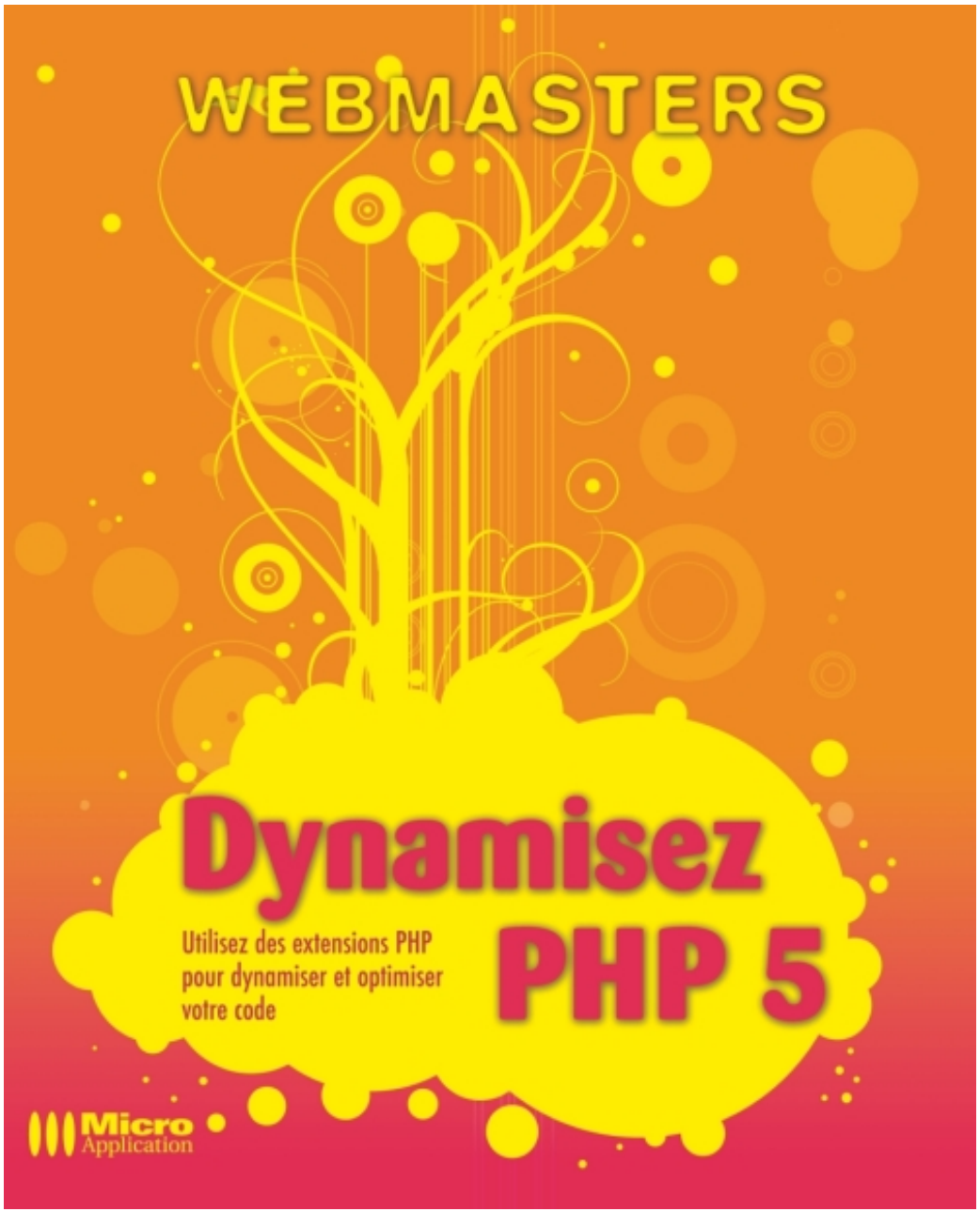

WEBMASTERS
Dynamisez PHP 5
Utilisez des extensions PHP pour dynamiser et optimiser votre code
Micro Application

1ère Édition - Janvier 2008

Auteurs - David DRAPEAU et Frédéric SUIRE

ISBN : 2-978-2-300-01135-1

Couverture réalisée par Sébastien Wiegant

MICRO APPLICATION
20-22, rue des Petits-Hôtels
75010 PARIS
Tél. : 01 53 34 20 20
Fax : 01 53 34 20 00

Support technique :
Également disponible sur www.microapp.com
http://www.microapp.com

Avant-propos

La collection *Atelier du webmaster* s'adresse aux personnes initiées au développement de sites web qui souhaitent découvrir et mettre en pratique les nouvelles technologies Internet. Sans négliger les aspects théoriques, nous donnons toujours priorité à la pratique afin que vous puissiez rapidement être autonome.

À travers les différents titres de cette collection vous découvrirez les technologies qui font le web 2.0 et feront ce que certains nomment déjà le web 3.0.

Conventions typographiques

Afin de faciliter la compréhension des techniques décrites, nous avons adopté les conventions typographiques suivantes :

- **gras** : menu, commande, boîte de dialogue, bouton, onglet.
- *italique* : zone de texte, liste déroulante, case à cocher, bouton radio.
- `Police bâton` : instruction, listing, texte à saisir.
- ➥ : dans les scripts, indique un retour à la ligne volontaire dû aux contraintes de la mise en page.

Il s'agit d'informations complémentaires relatives au sujet traité. Propose conseils et trucs pratiques.

Mise en garde sur un point important à ne pas négliger.

Sommaire

Préface

Bonjour et merci pour l'investissement que vous venez d'effectuer. Vous allez découvrir dans cet ouvrage certaines extensions de PHP5 dont on parle beaucoup, mais qui sont très peu utilisées. Vous allez découvrir également des extensions internes à PHP construites en C et compilées afin d'être intégrées dans le moteur de PHP (Zend). L'avantage de ces extensions par rapport aux bibliothèques programmées en PHP (PEAR par exemple) est leur rapidité. Elles ne seront pas toutes traitées en un seul ouvrage, le manuel PHP en répertoriant, à l'heure où ces lignes sont écrites, cent quatre-vingt cinq. Le choix a été porté uniquement sur celles documentées dans le manuel officiel de PHP. Cette décision tient au fait que nous pensons qu'il est beaucoup plus important et utile d'enseigner à quelqu'un à cuisiner plutôt que de le nourrir toute sa vie. Vous allez donc découvrir beaucoup de cas pratiques et assez peu de théorie, celle-ci étant déjà publiée sur des millions de pages web et dans une innombrable quantité de livres.

Cet ouvrage est divisé en chapitres qui contiennent chacun un thème et plusieurs sections. Certains chapitres sont très courts et d'autres beaucoup plus longs. La raison en est que certaines extensions comme BZIP2 ne comportent qu'une dizaine de fonctions et aucune constante prédéfinie ni directive de configuration à placer dans le fichier *php.ini*, et que d'autres extensions contiennent plus d'une centaine de fonctions ainsi qu'un grand nombre de constantes et des directives de configuration.

Pour certains chapitres, il y a un cas pratique, présenté en fin de chapitre, et celui-ci est parfois expérimenté selon plusieurs méthodes afin de laisser au lecteur le choix pour intégrer une méthode plus qu'une autre dans un projet. Le cas pratique a pour objectif de vous fournir un exemple utilisable à l'intérieur d'un cas réel, et donc concret : site Internet, applications Intranet/extranet ; et cela en utilisant ces fameuses extensions qui sont (trop ?) souvent omises. C'est-à-dire que vous pouvez récupérer le code et le copier/coller. Vous devrez probablement aussi le modifier quelque peu, pour l'utiliser dans un (ou plusieurs) de vos projets.

Le chapitre 1 montre l'intérêt de compresser des fichiers contenant du texte avec l'extension BZIP2 afin de gagner de la place sur le disque dur et d'utiliser ZIP pour placer des fichiers dans une archive, toujours dans l'objectif de gagner de l'espace disque. L'extension ZLIB sera également décrite en fin de ce chapitre, juste avant le cas pratique.

Le chapitre 2 prouve que PHP n'est pas en reste en ce qui concerne la communication avec des logiciels de bureautique tels Word, Excel et les équivalents OpenOffice.org. Nous vous expliquerons comment créer une feuille Microsoft Excel contenant un graphique trouvant ses données dans une base MySQL, ainsi qu'une méthode, certes empirique mais extrêmement efficace, pour interpréter des macros Visual Basic en PHP.

Nous vous présenterons également DOM, puissante extension gérant le XML, dans le but de créer des documents OpenOffice.org.

Le chapitre 3, quant à lui, vous explique comment manipuler les images et vous permettra de prendre conscience des fonctions très puissantes qu'offre PHP dans ce domaine avec la possibilité de créer des dessins complexes.

Le chapitre 4 est consacré à la génération des fichiers PDF : textes dynamiques ou statiques, images, signets… En somme, une petite boîte à outils pour les documents destinés à vos utilisateurs.

Le chapitre 5 est une découverte du monde LDAP et de ses avantages et inconvénients par rapport à une base de données.

Le chapitre 6 aborde justement le domaine BDD (Bases de données) avec PDO (*PHP Data Object*) qui permet avec les mêmes fonctions de se connecter à n'importe quel SGBDR (Système de gestion de base de données) : MySQL, PostgreSQL, Oracle, etc.

Le chapitre 7 décrit les fonctions des extensions Classkit et Runkit. Vous apprendrez comment écrire du code dynamique et l'intérêt de le faire.

Le chapitre 8 poussera un peu plus loin avec `Reflection` qui est une exception à la règle. `Reflection` n'est pas une extension interne à PHP mais une API (*Application Programming Interface*).

Le chapitre 9 se consacre au système de cache et vous montrera comment accélérer l'affichage de pages dynamiques tout en soulageant votre serveur.

Le chapitre 10 montre comment créer du code illisible humainement puisque compilé. C'est l'extension Bcompiler qui s'en charge, le tout pour protéger vos algorithmes.

Le chapitre 11 décrit une méthode pour étudier une nouvelle extension encore non acquise. Vous comprendrez comment les auteurs de ce livre s'y prennent pour étudier et assimiler une extension non encore documentée.

Il ne vous reste plus qu'à tourner les pages, et faire ainsi un magnifique voyage dans le monde des extensions internes à PHP. Nous vous souhaitons de faire des découvertes enrichissantes dans ce domaine.

Objectifs du livre

Ce livre a pour objectif de vous éclairer sur les extensions de PHP très peu usitées. Ces extensions nous ont pourtant été très utiles lors de la création de certains projets effectués dans un passé plus ou moins lointain. Pour découvrir ces extensions et les maîtriser toutes, il nous a fallu du temps. Beaucoup de temps. Et aussi des plantages. Énormément

de plantages. Il est dit que c'est la meilleure façon de progresser. Sans compter le temps que nous avons passé à chercher les informations pour, à la fois trouver les bonnes extensions et les documentations. Le temps, c'est ce que, par le biais de cet ouvrage, nous désirons vous faire économiser. Nous le faisons aussi dans l'esprit PHP qui est de fournir nos découvertes sans rien cacher. Vous permettre de gagner du temps, de l'argent et des efforts parfois inutiles pour trouver des informations qui, au bout du compte, ne sont pas celles dont vous avez besoin.

Nous vous conseillons de considérer ce livre comme un dictionnaire. Dans un dictionnaire, vous cherchez un mot précis. Dans ce livre, c'est la même chose au niveau des informations. Vous désirez une information sur la compression de fichiers ? Rendez-vous au chapitre 1. Vous désirez utiliser l'API de réflexion ? Allez jeter un œil au chapitre 8. Vous voulez savoir comment faire pour trouver les informations et comment on étudie une nouvelle extension afin de la maîtriser ? Vous découvrirez tout dans le chapitre *Méthodologie*. Consultez directement le chapitre consacré au sujet qui vous intéresse. Vous désirez retrouver un cas pratique afin de l'adapter à l'un de vos projets, allez-y, ces exemples sont totalement libres de droits et nous y tenons. Vous désirez nous poser des questions, écrivez-nous, nos adresses e-mail sont à votre disposition (dans la section *À propos des auteurs* de cette préface), ainsi, bien sûr, que nos oreilles.

Tels sont les objectifs de ce livre. Et s'il vous manque des informations, si vous auriez rédigé telle ou telle section autrement, ou si encore vous avez le moindre argument à formuler, n'hésitez pas à nous le faire savoir !

À qui s'adresse ce livre ?

En voilà une drôle de question ! Un livre sur les extensions de PHP ne peut s'adresser qu'aux développeurs PHP. Cela semble logique et d'une évidence certaine ? Rien n'est moins sûr : quels développeurs PHP ? Les bons, les très bons, les débutants ? On ne peut pas dire qu'un certain niveau soit requis, car un développeur débutant pourra trouver son compte dans ce livre, pourvu qu'il sache ce qu'est une fonction ou une boucle conditionnelle. Le développeur confirmé, lui, trouvera des explications claires et rapides sur des extensions que nous avons sélectionnées car elles nous ont permis de répondre à de nombreux problèmes. Le principal intérêt est que cela se traduira par un inévitable gain de temps dans la réalisation de vos projets.

Finalement, ce livre s'adresse à ceux qui veulent en connaître plus sur PHP comme à ceux qui ont une problématique précise et qu'une extension du langage PHP pourrait résoudre. Il peut être vu comme "une formation à l'utilisation des extensions", car il offre des connaissances sur les principales extensions de PHP, puis propose des méthodologies permettant d'appréhender de nouvelles extensions par soi-même et d'élargir encore son

savoir et ses compétences. Ce livre peut également être utilisé comme un dictionnaire, et permettre à chacun de prendre ici et là des morceaux de codes, et la compréhension qui va avec, afin d'enrichir considérablement ses propres réalisations et/ou celles des autres.

À propos des auteurs

David Drapeau

Développeur autodidacte au parcours atypique et éclectique (manutentionnaire, musicien, magasinier, hypnothérapeute, formateur en informatique et en communication…), David Drapeau est tombé dans l'univers de la programmation quand il était petit. Aujourd'hui, il se consacre entièrement à PHP5 et attend avec impatience PHP6 dont il teste déjà les nouveautés depuis le mois d'août 2007. Il étudie également, pendant son temps libre, tout ce qui peut entrer en interaction avec PHP tels OpenLaszlo, AJAX et tout ce qui touche de près ou de loin le domaine Internet : webmarketing, sécurité informatique, algorithmes. Il est également passionné par les mathématiques qu'il pratique pour s'amuser quand il lui reste un peu de temps libre.

Pour le joindre : `david.drapeau@gmail.com`

Frédéric Suire

Frédéric Suire s'est intéressé dès son plus jeune âge, tout comme David Drapeau, au monde de la programmation. Il a lui aussi suivi un parcours atypique, passant d'études littéraires à des études scientifiques, d'études musicales à des études artistiques. Aujourd'hui, il est développeur web indépendant et maîtrise, plus par passion que par nécessité, un grand nombre de langages destinés à l'Internet (du HTML au PHP, du JavaScript à l'ActionScript, et bien d'autres encore). Par là même, peu de logiciels de graphisme lui sont inconnus. Son cheval de bataille favori ? Le développement d'interfaces homme-machine, où il peut s'amuser aussi bien graphiquement qu'en programmation.

Pour le joindre : `frederic@suire.info`

1

Compresser, c'est gagner…

Il est assez fréquent de reprendre des projets qui possèdent une partie de l'application qui comprend la gestion des fichiers. Et ces fichiers sont bien souvent, pour ne pas dire tout le temps, au format texte (*.txt*), PDF (*.pdf*), et même au format Excel (*.xls*). C'est là tout l'intérêt des algorithmes de compression.

1.1 BZIP2 : compresser des données

Évidemment, ces fichiers prennent énormément de place sur le serveur (le hardware) et tout le monde n'a pas les moyens de se payer un serveur dédié. Pourtant, certains sites amateurs sont très bien fournis au niveau du contenu avec des liens internes vers des fichiers *.pdf* ou *.xls* pour des graphiques et des tableaux.

Aujourd'hui, de très nombreux formats de compression existent : *.zip*, *.tar*, *.gz*, *.tgz*, *.rar*, *.bz2*, *.ace* et bien d'autres encore. Dans ce chapitre, vous allez découvrir trois formats compilés dans PHP : BZIP2, ZIP et ZLIB. Leurs avantages : puisque programmés en C et compilés avec PHP, il en ressort une grande rapidité d'exécution. En premier lieu, nous allons vous décrire BZIP2 qui permet de compresser des données en quelques lignes de code et d'y accéder de nouveau ultérieurement tout aussi rapidement. Puis suivra ZIP, la plus conséquente des trois extensions PHP étudiées dans ce chapitre. ZIP contient beaucoup de constantes prédéfinies (plus d'une quarantaine) et de fonctions (environ une quarantaine). ZLIB complètera ce chapitre avec l'étude de toutes ses fonctions.

Comme c'est en forgeant qu'on devient forgeron, vous ne trouverez que très peu de théorie. Beaucoup de sites Internet s'en chargent. Tout au long de ce livre, vous trouverez peu de discours et beaucoup de code.

BZIP2 est une extension interne à PHP (programmée en C donc) qui utilise la bibliothèque créée par Julian Seward et qui a vu le jour au milieu des années 90. Cette bibliothèque crée des fichiers compressés à partir de l'algorithme de Burrows-Wheeler associé au codage de Huffman : l'extension des fichiers créés est *.bz2*.

> **Tous les exemples vérifiés et testés**
>
> Tous les programmes présentés dans ce livre ont été testés avec XAMPP 1.6.2 sur Windows XP avec XAMPP installé dans *C:\Program Files*.

Création d'un document

Ouverture

La première fonction à utiliser pour créer le premier document BZIP est celle d'ouverture d'un fichier *.bz2* en mode Écriture. Pour faire cela, c'est la fonction `bzopen()` qui s'y prête.

- `resource bzopen(chaine $nom_fichier, chaine $mode_ouverture)`

`$mode_ouverture` doit avoir pour valeur `r` (lecture) ou `w` (écriture). Dans le cas du mode Écriture, si le fichier n'existe pas, il est créé.

```
<?php // Début du script créer_bzip2.php
// On ouvre un fichier nommé exemple.bzip2
$bzo = bzopen('./fichiers/exemple.bz2', "w");
```

Fermeture

Puis, comme pour les fichiers texte, le fichier est fermé grâce à la fonction `bzclose()`.

- `int bzclose(resource $bzo)`

Le paramètre `$bzo` est l'identifiant de connexion défini lors de l'appel de la fonction `bzopen()`.

BZ_creer.php

```
// On ferme le fichier bzip2
$bzc = bzclose($bzo);
?>
```

Lors de l'exécution du script *BZ_creer.php* par le biais de votre navigateur favori, le script PHP va créer un fichier nommé *exemple.bz2*.

Compression

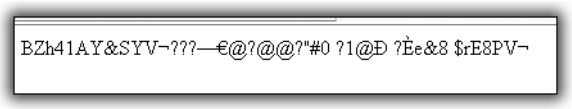
Les fonctions indispensables à tous les fichiers, ouverture et fermeture, ont été étudiées. Entrons maintenant dans le vif du sujet avec l'étude de la fonction de compression `bzcompress()`.

- `mixte bzcompress(chaine $donnees [, entier $taux_compression [, entier $facteur_de_travail]])`

`bzcompress()` compresse la chaîne `$donnees` et retourne les données encodées ou un numéro d'erreur en cas d'échec lors de la procédure de compression. Le taux de compression est affecté au paramètre `$taux_compression` qui possède une valeur qui peut varier de 1 à 9. 9 représente le meilleur taux de compression mais s'avère un peu plus lent que 1 qui a par contre un petit taux de compression. `$facteur_de_travail` contrôle le comportement de la compression. Sa valeur par défaut est 34 et 0 est une valeur spéciale.

BZ_compression.php

```
<?php
$texte =<<<lecontenudufichier
Placez ici un texte de plus d'une dizaine de lignes afin de mieux vous
➥ rendre compte de la compression.
lecontenudufichier;

// Le texte brut
echo $texte .'<br/>';

// Le texte compressé
$bzcomp = bzcompress($texte);
echo $bzcomp;

?>
```



▶ Fig. 1.1 : Résultat dans le navigateur de retour_bzcompress.php

Décompression

Décompresser des données BZIP2 est d'une simplicité enfantine. Il suffit simplement de faire appel à la fonction `bzdecompress()` dont la syntaxe est la suivante :

- `mixed bzdecompress(chaine $donnees_compressees[, entier $small])`

La fonction `bzdecompress()` décompresse les données `$donnees_compresses` qui ont été compressées avec la fonction `bzcompress()`.

`$small` permet de choisir entre le mode par défaut (si `$small = false`) ou un autre mode (si `$small = true`) de décompression qui prend peu de mémoire mais est plus lent à l'exécution.

Documentation officielle

Vous pouvez vous référer à la documentation officielle en anglais à l'adresse suivante :

http://www.bzip.org/1.0.3/bzip2-manual-1.0.3.html.

Écriture

Cependant, `bzcompress()` n'écrit pas les données compressées dans le fichier *.bz2*. La fonction donne juste un aperçu des données telles qu'elles seront intégrées dans le fichier compressé. Il est temps maintenant de voir comment insérer les données dans le fichier *exemple.bz2*. Pour effectuer cette manipulation, l'utilisation de la fonction `bzwrite()` s'avère nécessaire.

`entier bzwrite(resource $bzo, chaine $donnees [, entier $nb_caract_max])` écrit dans le fichier `$bzo` les données `$donnees` dont il est possible d'imposer une limite de caractères grâce au paramètre optionnel `$nb_caract_max`. L'utilisation de la fonction `bzwrite()` est visible dans le script suivant :

BZ_ecriture.php

```
<?php
// Ouverture
$bzo = bzopen('./fichiers/exemple.bz2', "w");

$texte =<<<lecontenudufichier
lecontenudufichier;

// Ecriture dans le fichier compressé
// int bzwrite ( resource bz, string data [, int length] )
bzwrite($bzo, $texte, strlen($texte));

$bzc = bzclose($bzo);
?>
```

Lecture

Gagner de la place est une chose et accéder de nouveau à des données en est une autre. Pourquoi gagner de la place s'il faut ensuite un grand nombre de lignes de code pour accéder à nouveau aux données ? Eh bien, c'est tout aussi simple de lire des données pour un fichier compressé en *.bz2* qu'en *.txt*.

Pour la lecture d'un fichier compressé en *.bz2*, la responsabilité en revient à la fonction `bzread()`.

- `chaine bzread(resource $bzo[, entier $nb_caract_max] )`

BZ_lecture.php

```
<?php
$file = "./fichiers/dynamisez_php.bz2";
$bzo = bzopen($file, "r") or die("Impossible d'ouvrir le fichier $file");

$decompresser_fichier = '';
while(!feof($bzo)) {
  $decompresser_fichier .= bzread($bzo, filesize($file));
}

bzclose($bzo);
echo "Le contenu du fichier $file est : <br />\n";
echo stripslashes($decompresser_fichier);
?>
```

Pour l'instant, les données affichées sont toujours compressées et leur lecture est incompréhensible pour un esprit normalement constitué. Il est donc nécessaire de décompresser les données afin de rendre la lecture compréhensible à tout un chacun.

Pour cela, il suffit juste de modifier la ligne :

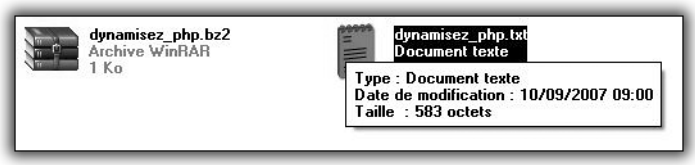
```
echo stripslashes($decompresser_fichier);
```

Et de procéder de la façon suivante :

```
echo stripslashes(bzdecompress($decompresser_fichier));
```

1. Maintenant, dans le même répertoire *fichiers*, créez un fichier *dynamisez_php.txt* et faites un copier/coller du texte original (non compressé) dans ce fichier texte. Nous insistons sur le fait qu'il s'agit d'un copier/coller pour être certain que le contenu soit identique afin de rendre crédible la démonstration. Vous vous retrouvez donc dans le répertoire *fichiers* avec *dynamisez_php.bz2* et *dynamisez_php.txt*.
2. Placez le curseur de la souris sur le fichier *.bz2* et regardez sa taille (en octets) puis faites de même sur le fichier *.txt*. Vous pouvez voir qu'avec si peu de contenu, il y a déjà une différence assez nette entre les deux fichiers, le fichier compressé étant bien entendu le plus léger. Imaginez maintenant la différence de taille par rapport à un fichier *.txt* de plusieurs Ko et ce pour un répertoire de plusieurs dizaines, voire centaines de fichiers. Vous visualisez le gain d'espace disque économisé sur le serveur ? Voici la différence que cela donne sur le poste suivant :

► Fig. 1.2 :
La taille du fichier compressé



► Fig. 1.3 :
La taille du fichier texte

Gestion des erreurs

bzerror/ bzerrno/bzerrstr

- `tableau bzerror(resource $bz)`
- `entier bzerrno(resource $bz)`
- `chaine bzerrstr(resource $bz)`

La fonction `bzerror()` retourne un tableau qui contient à la fois le numéro et le message d'erreur. Les noms des clés sont `errno` pour le numéro d'erreur et `errstr` pour le message d'erreur.

Les fonctions `bzerrno()` et `bzerrstr()` retournent respectivement le numéro et le message d'erreur sans avoir à utiliser la fonction `bzerror()` :

BZ_bzerror.php

```
<?php
$file = "./fichiers/dynamisez_php.bz2";
$bzo = bzopen($file, "r");

if(!bzwrite($bzo, "écriture impossible en mode lecture")){
  $bze = bzerror($bzo);

  echo '<pre>';
  print_r($bze);
  echo '</pre>';
}

bzclose($bzo);
?>
```

Vous pouvez récupérer les mêmes informations par les clés `errno` et `errstr` de la manière suivante :

BZ_errno_errstr.php

```
<?php
$file = "./fichiers/dynamisez_php.bz2";
$bzo = bzopen($file, "r");

if(!bzwrite($bzo, "écriture impossible en mode lecture")){
  $bze = bzerror($bzo);

  echo $bze['errno'] .'<br/>'. $bze['errstr'];
}

bzclose($bzo);
?>
```

La troisième méthode utilise les fonctions `bzerrno()` et `bzerror()` comme le montre le script *bzerrno_bzerrorstr.php* :

BZ_bzerrno_bzerrstr.php

```
<?php
$file = "./fichiers/dynamisez_php.bz2";
$bzo = bzopen($file, "r");

if(!bzwrite($bzo, "écriture impossible en mode lecture")){
  $bzerrno = bzerrno($bzo);
  $bzerrstr = bzerrstr($bzo);

  echo $bzerrno .'<br/>'. $bzerrstr;
}

bzclose($bzo);
?>
```

Vous pouvez vous apercevoir qu'il est très facile de gérer des fichiers compressés à l'aide de la bibliothèque BZIP2. Cependant, ce n'est pas la seule bibliothèque disponible pour PHP dont l'extension est programmée en C. Il en existe une autre qui contient plus de fonctions, et qui est par logique plus complète et plus complexe : il s'agit de ZIP. C'est l'extension la plus connue et la plus utilisée. Peut-être aussi la plus ancienne. Nous allons étudier cette extension et découvrir ce qu'elle contient de plus par rapport à BZIP2.

1.2 ZIP : créer des archives

Tout d'abord, un peu d'histoire. Le format de compression ZIP a été créé à la fin des années 80 par Phil Katz. Pendant des années, le format ZIP s'est imposé comme un standard de compression. Ce qui était normal, vu qu'il n'y avait pas de concurrence à l'époque. Puis, sont venus tellement de formats de compression qu'il est difficile de ne pas en oublier un seul. Créer un fichier ZIP par le biais de PHP est très simple.

La première différence entre BZIP2 et ZIP est que BZIP2 permet de compresser du contenu textuel, c'est-à-dire des données, alors que ZIP permet de créer une archive (un dossier compressé en quelque sorte) et d'y placer des fichiers, des répertoires. L'autre grande différence est que ZIP fonctionne en version objet.

Compression

Voici un exemple très simple de la création d'une archive dans laquelle deux fichiers créés précédemment avec BZIP2, *dynamisez_php.bz2* et *dynamisez_php.txt*, vont être insérés. Le programme est très court et, chose courante en PHP, il est très simple :

ZA_addFile.php

```
<?php
// On crée une instance de la classe ZipArchive
$zip = new ZipArchive();

// On crée une archive ZIP
// si elle n'existe pas, on la crée (ZIPARCHIVE::CREATE)
if($zip->open('./fichiers/exemple.zip',
              ZIPARCHIVE::CREATE) === TRUE){

  // On y insère le fichier dynamisez_php.txt
  $zip->addFile('./fichiers/dynamisez_php.txt',
                'dynamisez_php.txt');

  // ainsi que le fichier exemple.bz2
  // que l'on renomme php_dynamique.bz2
  $zip->addFile('./fichiers/dynamisez_php.bz2',
                'php_dynamique.bz2');

  // Et on ferme le fichier
  $zip->close();

  // Et voilà, c'est aussi simple que cela.
  echo 'ok';
```

```
} else {
  echo 'échec';
}
?>
```

Voici la syntaxe des fonctions utilisées :

- `mixte ZipArchive::open(chaîne nom_fichier[, entier drapeaux])`
- `bool ZipArchive::addFile(chaîne $nom_fichier[, chaîne $nom_dans_archive])`
- `bool ZipArchive::close(void)`

Explications

La première fonction utilisée est `open()`. Nous y reviendrons plus loin. La seconde fonction utilisée est `addFile()` qui va ajouter un fichier dans l'archive ZIP. Le premier paramètre, `$nom_fichier`, représente le nom du fichier existant, c'est-à-dire celui que l'on veut insérer dans l'archive. Le second paramètre, `$nom_local`, est optionnel. Il permet de renommer un fichier afin que son nom dans l'archive soit différent du nom du fichier récupéré en premier argument. Vous en avez un exemple lors de la deuxième utilisation de la fonction `addFile()` dans le script *creer_zip.php*.

Revenons maintenant à la fonction `open()`.Tout comme la fonction `fopen()` pour les fichiers texte, elle prend en premier argument le nom du fichier (archive ZIP) à ouvrir, ici *exemple.zip*. Le second argument est nouveau par rapport à la fonction `open()` des fichiers. Il s'agit de flags (drapeaux). En fait, ce sont des constantes qui possèdent une valeur numérique. Les constantes admises pour cette fonction sont les suivantes :

- `ZipArchive::CREATE` : crée l'archive si elle n'existe pas.
- `ZipArchive::EXCL` : génère une erreur si l'archive existe déjà.
- `ZipArchive::CHECKCONS` : effectue des analyses supplémentaires de consistances et émet une erreur si elles échouent.
- `ZipArchive::OVERWRITE` : crée l'archive si elle n'existe pas et l'écrase si elle existe déjà.

`ZipArchive::open()` retourne une valeur `mixed`. Il s'agit aussi de constantes dont une est booléenne (`TRUE` en cas de succès) et les autres sont de type `int` (les messages d'erreurs possibles en cas d'échec) :

- `ZipArchive::ER_EXISTS` : erreur générée si le fichier existe déjà.
- `ZipArchive::ER_INCONS` : archive ZIP inconsistante.
- `ZipArchive::ER_INVAL` : argument invalide.
- `ZipArchive::ER_MEMORY` : échec d'allocation mémoire.

- `ZipArchive::ER_NOENT` : erreur générée si le fichier n'existe pas.
- `ZipArchive::ER_NOZIP` : erreur générée s'il ne s'agit pas d'une archive ZIP.
- `ZipArchive::ER_OPEN` : erreur générée s'il est impossible d'ouvrir le fichier.
- `ZipArchive::ER_READ` : erreur de lecture.
- `ZipArchive::ER_SEEK` : erreur de pointeur.

Les erreurs

L'extension ZIP possède des constantes prédéfinies dont certaines sont destinées à gérer les erreurs. Le script nommé *retour_open.php* tente d'ouvrir une archive ZIP en générant volontairement une erreur. L'erreur générée est récupérée grâce à un switch-case qui prend en compte toutes les constantes de retour listées précédemment et permet de prendre connaissance du message d'erreur retourné lors de l'exécution du fichier PHP dans le navigateur. Le code donne donc :

ZA_retour_open.php

```
<?php
$zip = new ZipArchive();

$zo = $zip->open('./fichiers/exemple.zip', ZipArchive::EXCL);
if($zo !== true){
  switch($zo){
    case ZipArchive::ER_EXISTS:
      echo 'le fichier existe déjà';
      break;
    case ZipArchive::ER_INCONS:
      echo 'archive ZIP inconsistante';
      break;
    case ZipArchive::ER_INVAL:
      echo 'argument invalide';
      break;
    case ZipArchive::ER_MEMORY:
      echo 'échec allocation mémoire';
      break;
    case ZipArchive::ER_NOENT:
      echo 'fichier inexistant';
      break;
    case ZipArchive::ER_NOZIP:
      echo 'non reconnue comme archive ZIP';
      break;
    case ZipArchive::ER_OPEN:
      echo 'ouverture fichier impossible';
      break;
```

```
    case ZipArchive::ER_READ:
      echo 'argument de lecture';
      break;
    case ZipArchive::ER_SEEK:
      echo 'erreur de pointeur';
      break;
    default:
      echo 'erreur incompréhensible';
  }
}

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

Le flag `ZipArchive::EXCL` signifie que l'on veut ouvrir un fichier qui n'existe pas déjà, donc on veut en créer un nouveau. Ainsi, si le fichier que l'on essaie d'ouvrir existe, cela génère le message d'erreur suivant :

```
Le fichier existe déjà.
```

Ce qui correspond à la valeur de retour `ZipArchive::ER_EXISTS`.

Appelons maintenant un fichier qui n'existe pas et voyons ce que cela donne :

```
Ouverture fichier impossible.
```

Ce qui correspond à la valeur de retour `ZipArchive::ER_OPEN`.

L'erreur retournée n'est pas seulement due au flag mais aussi au premier argument.

Appelez, avec le premier argument, le fichier *exemple.bzip2* et placez le flag `ZipArchive::CREATE` ou `ZipArchive::CHECKCONS` en second argument. L'erreur retournée est `ZipArchive::ER_NOZIP`.

À présent, remplacez `CREATE` par `EXCL` et cette fois l'erreur retournée est `ZipArchive::ER_EXISTS`.

Décompression

La décompression d'une archive peut s'effectuer de plusieurs façons. La méthode décrite dans cette section est l'utilisation de la fonction `extracTo()`.

- `mixte ZipArchive::extractTo(chaîne $rep_dest[, mixte $entrees])`

`$rep_dest` est le répertoire dans lequel les fichiers extraits seront insérés. `$entrees` s'écrit sous forme de table si vous désirez extraire seulement quelques fichiers de l'archive. Ce

second paramètre étant optionnel, ne le renseignez pas si vous désirez extraire la totalité du contenu de l'archive. Le script *ZA_extractTo.php* extrait les fichiers *texte3.txt* et *texte4.txt* de l'archive et les place dans le répertoire *textes*. Si ce répertoire n'existe pas, il sera créé.

> ***Un exemple d'utilisation est donné dans la section*** **Fonctions utiles - Extraire des fichiers d'une archive.**

Fonctions utiles

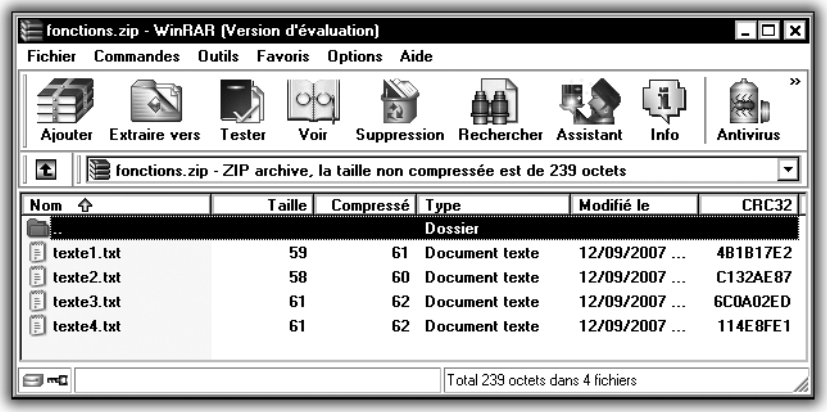
Ajouter des fichiers créés en temps réel

Vous avez vu comment insérer des fichiers existants dans une archive. Aussi, ZIP permet d'insérer des fichiers encore non existants et qui vont être créés en temps réel. Cela permet de mieux dynamiser PHP en gérant par exemple un système de fichiers de logs (gestion des erreurs et exceptions) qui seront automatiquement compressés dans une archive. Le script *ZA_addFromString.php* crée une archive ZIP qui contient plusieurs fichiers *.txt* :

ZA_addFromString.php

```
<?php
$oZA = new ZipArchive();

$open = $oZA->open('./fichiers/addFromString.zip', ZIPARCHIVE::CREATE);
if($open){

  echo "création de l'archive et insertion des fichiers texte
      grâce à la fonction addFromString()... ";
  $oZA->addFromString('texte1.txt', 'Le premier texte ajouté grâce à
            la fonction addFromString()');
  $oZA->addFromString('texte2.txt', 'Le deuxième texte ajouté
            grâce à la fonction addFromString()');
  $oZA->addFromString('texte3.txt', 'Le troisième texte ajouté
            grâce à la fonction addFromString()');
  $oZA->addFromString('texte4.txt', 'Le quatrième texte ajouté
            grâce à la fonction addFromString()');
  echo 'ok<br/>';

  $oZA->close();
} else {
  echo 'échec';
}

?>
```

Après avoir exécuté le fichier *addFromString.php*, une archive a été créée dans le répertoire courant. Elle contient quatre fichiers texte nommés *texte1.txt*, *texte2.txt*, *texte3.txt* et *texte4.txt*.



▶ Fig. 1.4 : Les quatre fichiers texte de l'archive ZIP

En étudiant le script précédent, il est très aisé de remarquer que la fonction qui permet de gérer du contenu en temps réel est `addFromString()`.

- `bool ZipArchive::addFromString(chaîne $nom_dans_archive, chaîne $contenu)`

Grâce à la fonction `addFromString()` le contenu (texte) passé en second paramètre (`$contenu`) sera intégré dans le fichier dont le nom est défini en premier paramètre (`$nom_dans_archive`) de la fonction.

Extraire des fichiers d'une archive

La fonction qui permet l'extraction de tout ou partie de l'archive est `extractTo()`.

- `mixte ZipArchive::extractTo(chaîne $destination[, mixte $entrees])`

Le script *ZA_extractTo.php* extrait les fichiers *texte1.txt* et *texte2.txt* dans un répertoire nommé *textes*. Si ce répertoire n'existe pas, il est créé. Si ce répertoire existe et qu'il contient des fichiers portant le même nom que ceux à extraire, alors les fichiers déjà existants seront écrasés sans demande de confirmation.

ZA_extractTo.php

```
<?php
$oZA = new ZipArchive;
```

```
if($oZA->open('./fichiers/addFromString.zip')){
  echo 'Extraction des fichiers texte2.txt et texte3.txt';
  $oZA->extractTo('./fichiers/textes/',
           array('texte2.txt', 'texte3.txt'));
  $oZA->close();

} else {
  echo 'échec';
}
?>
```

Renommer des fichiers contenus dans l'archive

Grâce à l'index

Il est possible d'agir sur les fichiers et répertoires même lorsque ceux-ci sont déjà intégrés dans l'archive. Il peut être ainsi nécessaire de renommer un fichier par le biais de son index. C'est la fonction `renameIndex()` qui s'en charge.

- `bool ZipArchive::renameIndex(entier $index, chaîne $nouveau_nom)`

La fonction `renameIndex()` va renommer le fichier qui a pour index `$index` défini en premier paramètre et lui donner le nom `$nouveau_nom` défini en second paramètre.

ZA_renameIndex.php

```
<?php
$oZA = new ZipArchive();

$open = $oZA->open('./fichiers/addFromString.zip',
              ZIPARCHIVE::CREATE);
if($open){
  // Renomme une entrée définie par son index
  echo "renommage d'un texte par le biais de son index :
              renameIndex()... ";
  $oZA->renameIndex(1, "rename_by_index.txt");
  echo 'ok<br/>';

  $oZA->close();
} else {
  echo 'échec';
}

?>
```

Une fois le fichier *renameIndex.php* exécuté dans le navigateur, il vous reste à ouvrir l'archive *addFromString.zip*. Maintenant, le fichier nommé *texte1.php* a été renommé et l'archive contient à la place un fichier nommé *rename_by_index.txt*. Renommez le fichier qui a pour nom *texte4.txt*.

Grâce au nom

Vous venez de voir à l'instant même comment renommer un fichier défini par son index. Vous pouvez effectuer la même action avec un fichier défini par son nom. Comme toujours, les fonctions PHP sont très simples à mémoriser, leurs noms étant suffisamment explicites. Donc, après avoir vu `renameIndex()`, nous allons prendre en considération `renameName()`.

- `bool ZipArchive::renameName(chaîne $nom_actuel, chaîne $nouveau_nom)`

La fonction `renameName()` prend en premier argument (`$nom_actuel`) le nom actuel du fichier cible contenu dans l'archive et en second argument (`$nouveau_nom`) le nouveau nom qui lui sera affecté.

ZA_renameName.php

```
<?php
$oZA = new ZipArchive();

$open = $oZA->open('./fichiers/addFromString.zip',
            ZIPARCHIVE::CREATE);
if($open){
  // Renomme une entrée définie par son nom
  echo "renommage d'un texte par le biais de son nom :
            renameName()... ";
  $oZA->renameName('texte4.txt', 'nouveau_nom_texte4.txt');
  echo 'ok<br/>';

  $oZA->close();
} else {
  echo 'échec';
}

?>
```

renameName.php renomme le fichier *texte4.txt* en *nouveau_nom_texte4.txt*.

Ajouter/Lire un commentaire à l'archive ZIP

- `chaîne ZipArchive::getArchiveComment()` : récupère (lit) le commentaire associé à l'archive.
- `mixte ZipArchive::setArchiveComment(chaîne $commentaire)` : ajoute (écrit) un commentaire associé à l'archive courante. Le commentaire est une chaîne de caractères `$commentaire` placée en paramètre de la fonction.

ZA_commentaire.php

```
<?php
$oZA = new ZipArchive();

$open = $oZA->open('./fichiers/addFromString.zip',
            ZIPARCHIVE::CREATE);
if($open){
  $readAC = $oZA->getArchiveComment();
  echo "lecture du commentaire de l'archive ZIP :
            getArchiveComment()...<br/>";
  echo '-> '. $readAC .'<br/>';

  echo "écriture d'un commentaire pour l'archive ZIP :
            setArchiveComment()...<br/>";
  $mix = $oZA->setArchiveComment('le commentaire de fonctions.zip');

  $readAC = $oZA->getArchiveComment();
  echo "lecture du commentaire de l'archive ZIP :
            getArchiveComment()...<br/>";
  echo '-> '. $readAC .'<br/>';
}

$oZA->close();
?>
```

Localiser une entrée

Vous avez vu précédemment que vous pouviez renommer un fichier en sélectionnant ledit fichier par son nom ou bien par son index. La présence d'un index signifie que les fichiers sont rangés dans un ordre. Vous pouvez localiser l'emplacement d'un fichier dans une archive avec la fonction `locateName()`.

- `mixte ZipArchive::locateName(chaîne $nom_fichier[, entier $drapeaux])`

La méthode `locateName()` va localiser l'emplacement du fichier cible défini dans la variable `$nom_fichier`. Le second paramètre, optionnel, est un drapeau auquel on affecte une constante prédéfinie parmi les suivantes :

- `FL_NOCASE`
- `FL_NODIR`

`FL_NOCASE` ignore la casse sur le nom. En clair, elle ne fera pas la différence entre les lettres majuscules et les lettres minuscules. `FL_NODIR` ignore le composant dossier.

ZA_locateName.php

```
<?php
$oZA = new ZipArchive();

$open = $oZA->open('./fichiers/addFromString.zip',
           ZIPARCHIVE::CREATE);
if($open){
  echo "localisation d'une entrée en utilisant son nom :
           locateName()...<br/>";
  $mix = $oZA->locateName('texte3.txt');
  echo $mix; // Affiche 2
}

$oZA->close();
?>
```

Vous avez pu étudier les principales fonctions de l'extension ZIP de PHP. Il en existe d'autres et elles sont assez nombreuses. Reportez-vous au manuel PHP (**http://php.net/**) pour en prendre connaissance. Il est temps de passer maintenant à ZLIB.

1.3 ZLIB : Quel est son secret ?

Jetons un œil rapide à cette bibliothèque. ZLIB a été créée par Jean-Loup Gailly pour la compression et Mark Adler en ce qui concerne la décompression. ZLIB fonctionne sous toutes les plates-formes (Windows, Linux et autres).

Fonctions ZLIB

Ouverture/Fermeture

Les syntaxes sont évidentes et ont été vues maintes fois dans les autres extensions qui gèrent les fichiers :

- `resource gzopen(chaîne $nom_fichier, chaîne $mode[, entier $use_include_path])` : ouvre un fichier compressé avec l'extension *.gzip* pour y écrire ou y lire des données.
- `bool gzclose(resource $gzo)` : `gzclose()` referme le fichier `$gzo` ouvert avec la méthode `gzopen()`.

GZ_creer.php

```
<?php
// Peut être utilisée pour lire un fichier qui n'est pas
// dans un format gzip ; dans ce cas, gzread() lira directement
//  le fichier sans décompression.
$find_in_include_path = 0;
$gzo = gzopen('./test.gz', "w", $find_in_include_path)
  or die('Ouverture du fichier échouée');
if(is_resource($gzopen)){
  echo '$gzopen est une ressource<br/>';
} else {
  echo "\$gzopen = ". $gzopen .'<br/>';
}

gzclose($gzopen);
?>
```

Écriture

Les noms de fonctions ZLIB sont souvent identiques aux noms de fonctions de BZIP2. La seule différence se trouve dans la première lettre (`b` pour BZIP2 et `g` pour ZLIB). De plus, elles remplissent souvent (toujours ?) la même action. Avec BZIP2, vous avez étudié `bzwrite()` pour écrire dans un fichier. Pour ZLIB, le nom sera `gzwrite()`. Comme expliqué à la phrase précédente, seule la première lettre change.

- `entier gzwrite(resource $gzo, chaîne $chaine [, entier $taille])`

La fonction `gzwrite()` va écrire dans la ressource (fichier *.gz*) définie dans le premier paramètre. Le troisième paramètre (`$taille`) est optionnel et va permettre d'imposer une limite de taille pour la chaîne à y inscrire.

Pour pouvoir écrire dans la ressource `$gzo`, il est nécessaire que celle-ci soit ouverte en mode Écriture :

GZ_ecriture.php

```
<?php
$chaine = "Bonjour tout le monde";

// Ouverture du fichier en mode écriture
$gzo = gzopen('./test.gz', "w")
  or die('Ouverture du fichier échouée');

// écrit le contenu de la chaîne string
// dans le fichier compressé zp.
gzwrite($gzo, $chaine);
unset($chaine);

gzclose($gzo);
?>
```

Lecture

Voici encore une fois un exemple très simple pour savoir comment lire dans un fichier compressé par ZLIB. La fonction qui s'y prête est `gzread()`.

- `chaîne gzread(resource $gzo, entier $taille)`

La fonction `gzread()` va lire les données situées dans la ressource `$gzo` en indiquant la taille maximale à lire grâce au deuxième paramètre `$taille` :

GZ_lecture.php

```
<?php
// Ouverture du fichier en mode de lecture
$gzopen = gzopen('./test.gz', "r")
  or die('Ouverture du fichier .gz échouée');

// lit jusqu'à length octets dans le fichier compressé gzip,
// représenté par zp. La lecture s'arrête lorsque
//length octets (décompressés) ont été lus, ou que la
// fin du fichier a été atteinte.
$gzread = gzread($gzopen, 1024);
echo $gzread;

gzclose($gzopen);
?>
```

Compresser/Décompresser

Tout aussi simple que l'écriture et la lecture, la compression et la décompression se font chacune en une seule ligne de code. Les fonctions concernées sont `gzcompress()` pour la compression et `gzuncompress()` pour la décompression.

- `chaîne gzcompress(chaîne $donnees[, entier $niveau_compression])`
- `chaîne gzuncompress(chaîne $donnees[, entier $taille])`

La fonction `gzcompress()` compresse la chaîne `$donnees` selon un taux de compression `$taux_compression` pouvant aller de 1 à 9. 1 est le plus rapide et 9 génère la meilleure compression. Pour sa part, `gzuncompress()` décompresse la chaîne `$donnees` compressée via `gzcompress()` jusqu'à avoir atteint la taille de `$taille` en octets.

GZ_compression.php

```
<?php
$gzopen = gzopen('./test.gz', "w")
  or die('Ouverture du fichier .gz échouée');

// string gzcompress ( string data [, int level] )
// compresse la chaîne donnée en utilisant le format de données ZLIB.
$chaine = "Bonjour tout le monde.";
echo 'avant la compression...<br/>$chaine = '. $chaine .'<br/><br/>';

$gzcompress = gzcompress($chaine);
echo 'après la compression...<br/>$chaine= '. $gzcompress .'<br/><br/>';

  // string gzuncompress ( string data [, int length] )
// décompresse une chaîne compressée.
$gzuncompress = gzuncompress($gzcompress);
echo 'après la décompression...<br/>$chaine= '. $gzuncompress
➥ .'<br/><br/>';

gzclose($gzopen);
?>
```

Jouer avec le curseur de position dans le fichier

Vous pouvez vous promener dans la chaîne de données grâce à la fonction `gzseek()` ou revenir au début du fichier compressé avec ZLIB au moyen de la fonction `gzrewind()`.

- `bool gzrewind(resource $gzo)`
- `entier gzseek(resource $gzo, entier $emplacement)`

La fonction `gzrewind()` replace le curseur de position au début du fichier défini par le paramètre `$gzo`. En fait, cela est équivalent à : `gzseek($gzopen, 0);`. Au contraire, `gzseek()` déplace le pointeur à un endroit précis (`$emplacement`) du fichier ZLIB défini par la ressource `$gzo`.

Positionnement n-1

Lors du choix de la position, il est indispensable de se rappeler que la position 1 dans le fichier possède la valeur `0` pour la fonction `gzseek()`. Pour rappel, c'est exactement la même chose pour `fseek()` en ce qui concerne le traitement des fichiers texte.

- `int gztell(resource $zp)` retourne la position du pointeur de position dans le fichier *.gz*.

Tout comme pour `gzseek()`, `gztell()` retournera la valeur `0` (zéro) lorsque le pointeur se trouvera à la première position dans le fichier *.gz*.

GZ_curseur.php

```
<?php
$file = 'test.gz';
$gzopen = gzopen($file, "r", 0);

// On lit le contenu du fichier gz en entier
$gzread = gzread($gzopen, filesize($file));
echo $gzread .'<br/>';

// On lit la position du pointeur de lecture
$gztell = gztell($gzopen);
echo $gztell .'<br/>';

// On revient au début du fichier
gzrewind($gzopen);
$gztell = gztell($gzopen);
echo $gztell .'<br/>';

// On déplace le curseur à la 5e position
// ATTENTION: la 1ere position a pour valeur 0
// donc la 5e position a pour valeur 4
$gzseek = gzseek($gzopen, 4);
$gztell = gztell($gzopen);
echo $gztell .'<br/>';
```

```
// On lit de la position actuelle jusqu'à la fin du fichier
$gzread = gzread($gzopen, filesize($file));
echo $gzread .'<br/>';

gzclose($gzopen);
?>
```

Lecture partielle dans le fichier

Lors de l'étude de la fonction `gzread()`, le contenu entier du fichier *.gz* a été lu. `gzread()`, grâce à son second argument, permet aussi de lire un nombre très précis de signes dans le fichier. Et une fonction très intéressante, nommée `gzpassthru()`, permet de connaître le contenu restant à lire.

- `int gzpassthru(resource $zp)`

GZ_finir_lecture.php

```
<?php

$gzopen = gzopen('./test.gz', "r")
   or die('Ouverture du fichier .gz échouée');

$gzread = gzread($gzopen, 2);
echo 'lecture de 2 caractères de la chaîne = '. $gzread .'<br/>';
// affiche: Bo

$nchar = gzpassthru($gzopen);

echo '<br/>'. $nchar;

gzclose($gzopen);
?>
```

Récupérer un fichier formaté

ZLIB permet de compresser des données. Il ne cherche aucunement à savoir ce que vous avez formaté. Donc, le fichier peut contenir du texte pur, du texte au format HTML ou XML ou autre encore. Voici un exemple avec une page Internet écrite en HTML.

Les fonctions qui vont être utilisées dans cet exemple sont les suivantes :

- `string gzgetc(resource $zp)` retourne une chaîne contenant un seul caractère. Cela revient à faire `gzread($gzopen, 1);`. L'avantage de `gzgetc()` est que cette fonction est plus rapide pour lire un seul caractère.

- `string gzgets(resource $zp, int $length)` retourne une chaîne d'une taille maximale n-1, où la valeur n est affectée à la variable `$length`.
- `string gzgetss(resource $zp, int $length [, string $balises_permises])` est identique à `gzgets()` avec en plus la suppression de toutes les balises HTML et PHP du contenu lu dans le fichier *.gz*. Les variables `$balises_permises` représentent les balises HTML que `gzgetss()` doit laisser dans le texte lu.

GZ_page_HTML.php

```
<?php
// La chaîne sur laquelle effectuer les manipulations
$chaine = "<html><head><title>Utilisation ZLIB</title></head>";
$chaine .= "<body>On étudie en ce moment <i>l'extension ";
echo "<b>ZLIB</b></i></body></html>";

// Ouverture du fichier test.gz en mode écriture
$gzopen = gzopen('./test.gz', "w")
    or die('Ouverture du fichier .gz échouée');

// On écrit la chaîne dans le fichier test.gz
gzwrite($gzopen, $chaine);

// on détruit la variable $chaine
unset($chaine);

gzclose($gzopen);

$file = 'test.gz';

// ouverture du fichier en mode lecture
$gzopen = gzopen($file, "r", 0);

// Lecture du contenu du fichier gz en entier
$gzread = gzread($gzopen, filesize($file));
echo 'lecture du contenu du fichier avec gzread()
            et htmlentities()...<br/>';
echo htmlentities($gzread) .'<br/><br/>';

// Retour du pointeur de lecture au début du fichier
gzrewind($gzopen);

// Déplacement du curseur à la 4e position
echo 'Déplacement du curseur à la 5e position,<br/>';
$gzseek = gzseek($gzopen, 3);
```

```
echo 'et on affiche le caractère qui y est positionné avec
           gzgetc()...<br/>';
$gzgetc = gzgetc($gzopen);
echo $gzgetc .'<br/><br/>';

// Retour du pointeur de lecture au début du fichier
gzrewind($gzopen);

echo 'Affichage du contenu du fichier avec gzgets()...<br/>';
$gzgets = gzgets($gzopen, filesize($file));
echo $gzgets .'<br/><br/>';

// Retour du pointeur de lecture au début du fichier
gzrewind($gzopen);

echo 'Affichage du contenu avec gzgetss()...<br/>';
echo 'Seule la balise HTML des caractères gras
           (<b>) <br/>';
$gzgetss = gzgetss($gzopen, filesize($file), "<b>");
echo $gzgetss;

gzclose($gzopen);
?>
```

Compression/Décompression : DEFLATE

DEFLATE est un algorithme de compression/décompression sur les données. Il obéit à la norme RFC1951. Autrement dit, DEFLATE va compresser des données et ces données compressées seront sauvegardées dans un fichier compressé à l'aide de ZLIB.

- `string gzdeflate(string $data [, int $level])` compresse les données.
- `string gzinflate(string $data [, int $taille])` décompresse les données. Le second argument, `$taille`, de la fonction `inflate()` indique le nombre de caractères à décoder.

GZ_deflate_inflate.php

```
<?php
$gzopen = gzopen('./test.gz', "w")
  or die('Ouverture du fichier .gz échouée');


$chaine = "Bonjour tout le monde.";
echo 'avant encodage...<br/>$chaine = '. $chaine .'<br/><br/>';
```

```
  // string gzdeflate ( string data [, int level] )
// compresse la chaîne donnée en utilisant
// le format de données DEFLATE.
$gzdeflate = gzdeflate($chaine);
echo 'après encodage...<br/>$chaine = '. $gzdeflate .'<br/><br/>';

  // string gzinflate ( string data [, int length] )
// décompresse une chaîne compressée avec la méthode gzdeflate().
$gzinflate = gzinflate($gzdeflate);
echo 'après décodage...<br/>$chaine = '. $gzinflate .'<br/><br/>';


gzclose($gzopen);
?>
```

1.4 Cas pratique : gestion des données et fichiers

Les programmeurs expérimentés sauvegardent souvent les erreurs générées par l'application dans des fichiers texte appelés fichiers de logs. Or, vous avez vu que sauvegarder des données textuelles dans un fichier compressé permet d'avoir un gain de place. Afin de comparer les trois méthodes, à savoir textes, fichiers compressés BZIP2 et archives ZIP, nous vous présentons en guise de cas pratique un exemple avec un formulaire de connexion. Si vous saisissez le bon pseudonyme (*login*) et le bon mot de passe (*password*), l'application vous inscrit un message de confirmation à l'écran particulièrement délicat. Par contre, si le pseudonyme ou le mot de passe ne valident pas votre identité, alors un message d'erreur sera sauvegardé afin que le webmaster puisse en prendre connaissance ultérieurement.

Afin de bien comparer les trois méthodes vues précédemment, nous allons effectuer les trois types de sauvegarde. L'objectif ici n'est pas de trouver le login et le mot de passe mais bien de se tromper une centaine de fois afin de comparer la différence de taille des fichiers/archives générés sur le disque dur et savoir quelle méthode est la plus économe en termes d'espace disque. C'est la raison pour laquelle nous vous avons caché le login et le mot de passe dans un autre chapitre du livre afin de vous laisser chercher des informations là où vous n'iriez pas habituellement.

cas_pratique1.php

```
<html>
<head>
<title>Gagnez de l'espace disque</title>
```

```
</head>
<body>
<?php
if(isset($_POST['login']) && isset($_POST['password'])){
  // Les versions non cryptées de $bonlogin et $bonpassword
  // se trouvent quelque part dans le livre
  // les deux sont mis ensemble et dans
  // un petit cadre. Regardez bien
  $bonlogin = '3491e6c6121c11030677673a743529c0880c4580';
  $bonpassword = 'deba12df7f2e9c6b4bb131f00ba4e86624d56baf';

  if(sha1($_POST['login']) == $bonlogin &&
     sha1($_POST['password']) == $bonpassword){
    echo 'Tu es trop fort!!!';
  } else {
    $time = time();
    echo $time;

    // On écrit dans bzip2
    $str = 'planté le '. $time;
    $bzo = bzopen('./bzip2/'. $time .'.bz2', "w");
    $bzw = bzwrite($bzo, $str, strlen($str));
    bzclose($bzo);

    // On écrit dans textes
    $pathname = './textes/';
    $filename = $time .'.txt';
    $fp = fopen($pathname . $filename, "w+");
    fwrite($fp, $str, strlen($str));

    // On écrit dans une archive ZIP
    $zip = new ZipArchive();

    if($zip->open('./erreurs_login.zip', ZipArchive::CREATE)){
      $zip->addFile($pathname . $filename, $filename);
      echo 'ok';
    } else {
      echo 'echec';
    }
    $zip->close();

    fclose($fp);

  }
} else {
  echo 'vous devez saisir un login et un mot de passe';
}
```

```
?>
<form action="#" method="POST">
login <input type="text" name="login" /><br/>
password <input type="password" name="password" /><br/>
<input type="submit" name="valider" value="Valider" />
</form>

</body>
</html>
```

Trompez-vous une bonne centaine de fois et comparez les différences de taille entre le répertoire *bzip2*, le répertoire texte et l'archive zippée. Attention, la différence est très nette.

Quant à la surprise, elle est de taille. Et le vainqueur est… le répertoire textes ! Le deuxième est le répertoire *bzip2* et le dernier l'archive ZIP. Pourquoi me demanderez-vous ? Simplement parce que la compression, que ce soit en ZIP ou en BZIP2, demande des opérations algorithmiques à effectuer et donc une plus grande taille dans le fichier. Ce qui signifie que pour un grand nombre de fichiers contenant très peu de données, les fichiers texte sont les plus légers puisqu'ils ne subissent aucune transformation. Voyons maintenant un cas pratique, le même à vrai dire, sauf que cette fois nous allons sauvegarder les messages d'erreurs dans un même fichier qui aura une validité d'une journée.

Au niveau métier, cette logique est d'ailleurs beaucoup plus facile à gérer pour le webmaster qui peut récupérer les erreurs au jour le jour. Voici donc le code qui, pour les trois méthodes, ajoute les messages d'erreurs en fin de fichier et ce, pendant une journée complète de 24 heures.

cas_pratique2.php

```
<html>
<head>
<title>Gagnez de la place</title>

</head>
<body>
<?php

if(isset($_POST['valider']) && isset($_POST['login']) &&
➥ isset($_POST['password'])){
  // L'important ici n'est pas de trouver le bon login
  // Mais justement de se tromper deux cents ou trois cents fois
```

```
// dans une journée et comparer la taille
$bonlogin = '3491e6c6121c11030677673a743529c0880c4580';
$bonpassword = 'deba12df7f2e9c6b4bb131f00ba4e86624d56baf';

if(sha1($_POST['login']) == $bonlogin && sha1($_POST['password']) ==
➥ $bonpassword){
  echo 'Tu es trop fort!!!';
} else {

  $str = "L'utilisateur a échoué lors de sa tentative de connexion le
  ➥ ". date("d/m/Y", time()) .'à'. date("H:i:s", time()) ."\n";
  $time = date("Y-m-d", time());
  echo $time .'<br/>';

  //////
  // On enregistre avec BZIP2
  $nom_fichier = './bzip2/'. $time .'.bz2';

  // Si le fichier du jour n'existe pas, il est créé,
  // sinon les données sont écrites à la suite des données existantes
  $bzo = bzopen($nom_fichier, "w");
  // On écrit dans bzip2
  $bzw = bzwrite($bzo, $str, strlen($str));
  bzclose($bzo);
  // Fin de BZIP2
  //////

  //////
  // On écrit dans le fichier texte quotidien
  //$pathname = './textes/';
  $nom_fichier = './textes/'. $time .'.txt';
  $fp = fopen($nom_fichier, "a+");
  fwrite($fp, $str, strlen($str));
  // Fin fichiers texte
  //////

  //////
  // On écrit dans une archive ZIP
  $zip = new ZipArchive();

  if($zip->open('./erreurs_login.zip', ZipArchive::CREATE)){
    $zip->addFile($nom_fichier, $nom_fichier);
    echo 'ouverture archive ZIP --> ok';
  } else {
    echo 'ouverture archive ZIP --> echec';
  }
  $zip->close();
```

```
    // Fin ZIP
    //////

    fclose($fp);

    echo '<br/><br/>échec de connexion.
         Allez, un peu de courage, vous y êtes presque.';
  }
} else {
    echo 'vous devez saisir un login et un mot de passe';
}

?>
<form action="#" method="POST">
login <input type="text" name="login" /><br/>
password <input type="password" name="password" /><br/>
<input type="submit" name="valider" value="Valider" />
</form>

</body>
</html>
```

Dans ce deuxième cas pratique, nous découvrons une limite de BZIP2. Il ne permet pas d'ajouter des données. Il écrase automatiquement le fichier BZIP existant pour en créer un nouveau alors que ZIP, récupérant toujours le fichier texte qui se met à jour à chaque tentative de connexion échouée, écrase simplement le fichier texte existant dans son archive. Il est donc très facile de faire évoluer le script de telle sorte que, au fur et à mesure du temps, c'est toujours la même archive ZIP, mais avec l'ajout de chaque fichier texte généré quotidiennement.

Ces deux cas pratiques peuvent donc être considérés comme des tests pour en venir au cas pratique définitif qui est *cas_pratique3.php*.

cas_pratique3.php

```
<html>
<head>
<title>Gagnez de la place</title>

</head>
<body>
<?php

if(isset($_POST['valider']) && isset($_POST['login']) &&
➥ isset($_POST['password'])){
```

```
// L'important ici n'est pas de trouver le bon login
// Mais justement de se tromper deux cents ou trois cents fois
// dans une journée et comparer la taille
$bonlogin = '3491e6c6121c11030677673a743529c0880c4580';
$bonpassword = 'deba12df7f2e9c6b4bb131f00ba4e86624d56baf';

if(sha1($_POST['login']) == $bonlogin && sha1($_POST['password']) ==
➥ $bonpassword){
  echo 'Tu es trop fort!!!';
} else {

  $str = "L'utilisateur a échoué lors de sa tentative de connexion le
  ➥ ". date("d/m/Y", time()) .'à'. date("H:i:s", time()) ."\n";

  //////
  // Ecriture dans le fichier texte quotidien

  // Définir la date du jour
  $aujourd_hui = date("Y-m-d", time());
  echo "aujourd'hui = ". $aujourd_hui .'<br/>';

  // Définir le fichier quotidien
  $nom_fichier = './textes/'. $aujourd_hui .'.txt';

  // S'il s'agit d'une nouvelle journée,
  // créer un nouveau fichier du jour
  if(!file_exists($nom_fichier)){

    // On définit la date de la veille
    $hier = date("Y-m-d", time() - 86400);
    echo 'hier = '. $hier .'<br/><br/>';

    // Récupérer le fichier de la veille
    $nom_fichier_hier = './textes/'. $hier .'.txt';
    // Renommer le fichier de la veille au fichier du jour
    rename($nom_fichier_hier, $nom_fichier);

    // Ecraser les données existantes de la veille
    $fp = fopen($nom_fichier, "w+");

  // Si c'est le fichier déjà créé pour la journée,
  // On ajoute les données à la suite du fichier
  } else {
    $fp = fopen($nom_fichier, "a+");
    fwrite($fp, $str, strlen($str));
  }
  // Fin fichiers texte
```

```
    // On écrit dans une archive ZIP
    $zip = new ZipArchive();

    if($zip->open('./erreurs_login.zip', ZipArchive::CREATE)){
      $zip->addFile($nom_fichier, $nom_fichier);
      echo 'ouverture archive ZIP --> ok';
    } else {
      echo 'ouverture archive ZIP --> echec';
    }
    $zip->close();
    // Fin ZIP
    //////

    fclose($fp);

    echo '<br/><br/>échec de connexion. Allez,
            un peu de courage, vous y êtes presque.';
  }
} else {
    echo 'vous devez saisir un login et un mot de passe';
}

?>
<form action="#" method="POST">
login <input type="text" name="login" /><br/>
password <input type="password" name="password" /><br/>
<input type="submit" name="valider" value="Valider" />
</form>

</body>
</html>
```

Avec ce troisième cas pratique, les seuls fichiers existants seront le fichier texte du jour ainsi qu'une seule archive ZIP qui contiendra toutes les données.

1.5 Check-list

Dans ce chapitre, nous avons vu :

- ✔ les extensions BZIP2, ZIP et ZLIB ;
- ✔ comment créer des fichiers de données compressées avec BZIP2 ;
- ✔ comment créer des archives avec ZIP ;

- ✔ la totalité des fonctions de ZLIB ;
- ✔ comment compresser avec chacune des extensions de ZLIB ;
- ✔ comment décompresser avec chacune des extensions de ZLIB ;
- ✔ la gestion de fichiers et dossiers compressés.

2

Utiliser la bureautique

Avez-vous déjà eu besoin d'exporter le contenu d'une base de données en un fichier Microsoft Excel ? Ou OpenOffice.org Calc ? Nous pouvons le faire facilement grâce à quelques extensions de PHP.

2.1 COM : créer des documents Microsoft Office

COM, c'est l'extension qui va nous permettre de créer des documents Microsoft Office. COM signifie *Component Object Model* et permet à du code écrit en n'importe quel langage d'interagir avec du code écrit dans un autre langage, à condition que ces deux langages respectent les conventions de nommage COM. Ce qui tombe plutôt bien, vu que PHP et Visual Basic les respectent… Nous allons donc utiliser COM afin que PHP puisse manœuvrer Visual Basic.

Même s'il est préférable de connaître Visual Basic, vous pourrez vous en sortir sans rien savoir de ce langage.

Découvrir COM

Tout d'abord, sachez que COM fait partie du cœur de PHP. Il n'y a donc rien à faire pour pouvoir l'utiliser, si ce n'est s'assurer que le langage avec lequel nous souhaitons faire communiquer PHP soit bien installé. Dans notre cas, cela se résume à s'assurer que Microsoft Word (si vous voulez générer un document Word) ou Excel (pour un document Excel) soit installé sur votre système.

Un fichier Microsoft Word

Nous allons commencer simplement, en écrivant une petite phrase dans un document Word. Dans un répertoire nommé *chap02*, créez le fichier *com_word.php* et remplissez-le comme ceci :

com_word.php

```
<?php

$repertoire = "D:\dynamisez PHP\chap02";

?>
```

Lorsque nous aurons besoin d'enregistrer notre fichier, nous utiliserons un chemin absolu pour indiquer le répertoire hôte.

Les quelques lignes que nous allons rajouter maintenant servent à instancier Microsoft Word :

```
// Démarrage de Word
$word = new COM("word.application")
  or die("Impossible de lancer Word");
```

En fait, l'objet COM que nous venons de créer et de mettre de côté dans la variable $word est une instance de l'application Microsoft Word. C'est donc à travers la variable $word que nous donnerons des instructions en Visual Basic. Et la première chose que nous allons faire, c'est de créer un document sur lequel s'appliqueront nos modifications :

```
// Creé un document vide
$word->Documents->Add();
```

Pour le moment, ne vous préoccupez pas de la syntaxe : nous sommes en train de faire du Visual Basic, nous y reviendrons ultérieurement. Voici maintenant la ligne qui va nous permettre d'inscrire une phrase dans notre document :

```
// écrit dans le document
$word->Selection->TypeText("Hello world !");
```

Nous décidons ensuite que le mot "world" soit en gras. Il va donc nous falloir sélectionner ce mot :

```
// déplacer le curseur de deux caractères
$word->Selection->MoveLeft(1, 2);
// Sélectionner 5 caractères
$word->Selection->MoveLeft(1, 5, 1);
```

Puis le faire passer en gras :

```
// Passer la sélection en gras
$word->Selection->Font->Bold=9999998;
```

Il ne nous reste plus qu'à enregistrer notre fichier, en spécifiant d'abord le bon chemin d'enregistrement :

```
// Sélectionner un répertoire
$word->ChangeFileOpenDirectory($repertoire);
// Enregistrer le fichier
$word->ActiveDocument->SaveAs("test.doc");
```

Et n'oublions pas de fermer notre instance de Microsoft Word et de libérer la mémoire :

```
//Fermeture de Word
$word->Quit();
// Libération des ressources
$word = null;
```

Vous pouvez maintenant lancer le fichier *com_word.php* depuis votre navigateur favori. Rien ne s'affichera à l'écran mais votre fichier a bien été sauvegardé à l'endroit convenu.

Voici le code complet du fichier *com_word.php* :

com_word.php

```
<?php

$repertoire = "D:\dynamisez PHP\chap02";

// Démarrage de Word
$word = new COM("word.application")
  or die("Impossible de lancer Word");

// Crée un document vide
$word->Documents->Add();

// écrit dans le document
$word->Selection->TypeText("Hello world !");

// déplacer le curseur de deux caractères
$word->Selection->MoveLeft(1, 2);
// Sélectionner 5 caractères
$word->Selection->MoveLeft(1, 5, 1);

// Passer la sélection en gras
$word->Selection->Font->Bold=9999998;

// Sélectionner un répertoire
$word->ChangeFileOpenDirectory($repertoire);
// Enregistrer le fichier
$word->ActiveDocument->SaveAs("test.doc");
//Fermeture de Word
$word->Quit();
// Libération des ressources
$word = null;
?>
```

La documentation de COM

Maintenant que le premier fichier Microsoft Word a été créé, attardons-nous un peu sur la partie Visual Basic de notre code. En effet, toutes les fonctions que nous avons utilisées dans le fichier *com_word.php* (`TypeText()`, `MoveLeft()`…) sont des fonctions Visual Basic, et non des méthodes de l'objet COM. C'est pour cela que la documentation de COM ne nous intéresse pas. Nous allons maintenant voir comment obtenir du code Visual Basic exécutant les actions que nous souhaitons, et comment le retranscrire au sein de notre code PHP.

Obtenir du code Visual Basic

Commençons par ouvrir Microsoft Word, exactement comme si nous allions rédiger un nouveau document.

Dans le menu **Outils**, sélectionnez le sous-menu **Macro**, puis cliquez sur **Nouvelle Macro…** ; une boîte de dialogue s'ouvre alors.

▶ Fig. 2.1 : Créer une nouvelle macro…

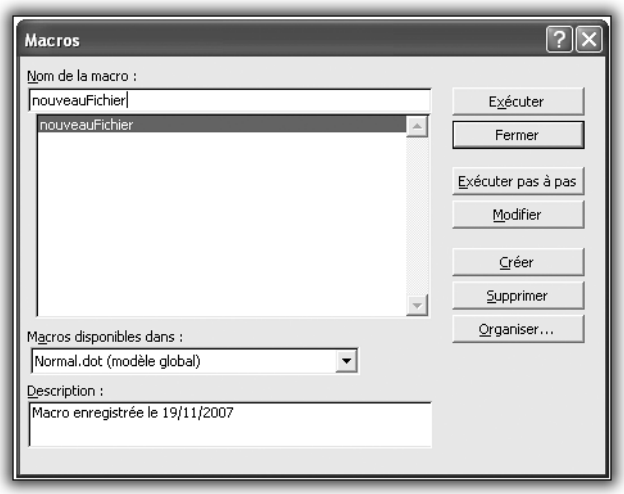
Donnez-lui un nom explicite, puis cliquez sur OK. Une nouvelle petite fenêtre apparaît à l'écran.

▶ Fig. 2.2 : Un enregistreur ?

Cette petite fenêtre ne présente que deux boutons dont les symboles sont éloquents : nous sommes visiblement en train d'enregistrer quelque chose, mais quoi ? Tout bonnement l'ensemble des modifications que nous allons maintenant faire à notre document.

Nous allons donc créer un nouveau document vide en cliquant sur **Nouveau**, dans le menu **Fichier**. Dans ce nouveau document, écrivons `"Hello world !"` puis sélectionnons le mot `"world"` à l'aide du clavier (tant que nous sommes en train d'enregistrer notre macro, les sélections de texte ne peuvent se faire à la souris). Mettons-le en gras, puis enregistrons notre document sur notre disque dur. Nous pouvons enfin arrêter l'enregistrement de la macro en cliquant sur le bouton carré du petit enregistreur.

Notre macro, c'est-à-dire la suite des instructions Visual Basic permettant de créer un nouveau document, d'écrire `"Hello world !"` à l'intérieur et bien d'autres choses, est donc enregistrée et visible quelque part. Pour la voir, sélectionnez le sous-menu **Macro** (dans le menu **Outils**) et cliquez sur **Macros…**



► Fig. 2.3 : Sélectionnez votre macro

Dans la boîte de dialogue qui apparaît, sélectionnez votre macro, puis cliquez sur le bouton **Modifier**. L'éditeur Visual Basic s'ouvre alors, affichant votre macro. Voici à quoi elle ressemble :

macro nouveauFichier

```
Sub nouveauFichier()
'
' nouveauFichier Macro
' Macro enregistrée le 16/11/2007
'
    Documents.Add Template:= _
        "C:\Documents and Settings\…\Normal.dot" _
        , NewTemplate:=False, DocumentType:=0
    Selection.TypeText Text:="Hello world !"
    Selection.MoveLeft Unit:=wdCharacter, Count:=2
    Selection.MoveLeft Unit:=wdCharacter, Count:=5, Extend:=wdExtend
    Selection.Font.Bold = wdToggle
    ChangeFileOpenDirectory "D:\dynamisez PHP\chap02"
    ActiveDocument.SaveAs FileName:="Helloworld.doc", FileFormat:= _
        wdFormatDocument, LockComments:=False, Password:="",
        ➥ AddToRecentFiles:= _
        True, WritePassword:="", ReadOnlyRecommended:=False,
        ➥ EmbedTrueTypeFonts:= _
        False, SaveNativePictureFormat:=False, SaveFormsData:=False,_
        SaveAsAOCELetter:=False
End Sub
```

Comprendre le code Visual Basic et l'interpréter en PHP à travers COM

Nous pouvons voir, dans la première ligne de ce code que l'on demande à un objet de l'application, `Documents`, d'ajouter (via, visiblement, une méthode `Add()`) quelque chose. La méthode `Add()` contient plusieurs arguments : `Template`, `NewTemplate` et `DocumentType`. Voici comment nous écrivons cette ligne, dans notre fichier *com_word .php* :

```
$word->Documents->Add();
```

Notre variable `$word` contient une instance de Microsoft Word. Nous cherchons à atteindre l'objet `Document` de l'application, c'est pourquoi il y a une flèche (->) entre notre variable et l'objet `Document`. Ce même objet `Document` dispose d'une méthode `Add()`, ainsi que nous le spécifie le code Visual Basic contenu par notre macro, qui nous permet d'ajouter un document sur lequel travailler. Comme vous pouvez le voir, nous ne passons, en PHP, aucun paramètre à la méthode `Add()`. Les noms de ces arguments, apparaissant dans le code Visual Basic, nous renseignent sur leur nature : ils ne servent qu'à décrire un document Word par défaut.

Néanmoins, avant d'enlever ses paramètres, nous nous sommes assurés, en faisant des tests avec et sans, que cela marchait. Lorsque vous découvrirez de nouvelles fonctions, ou de nouveaux objets Visual Basic, il vous faudra bien les tester au sein de votre code PHP !

Les fonctions en Visual Basic

Lorsque l'on appelle, en Visual Basic, une méthode ou une fonction, nous ne passons pas les paramètres à cette dernière en utilisant des parenthèses. On sépare (par un espace) le nom de la fonction et ses arguments, et on donne le nom de chaque paramètre. Ce sont ces spécificités syntaxiques qui nous permettent de savoir que l'on a affaire à une fonction et de la traiter comme telle depuis PHP.

Regardons la deuxième ligne du code Visual Basic :

```
Selection.TypeText Text:="Hello world !"
```

Elle nous informe de l'existence d'un objet `Selection` dans l'application Word ; et que cet objet dispose d'une méthode `TypeText()` qui ne prend qu'un seul argument nommé `Text`. De par le nom de la méthode, nous comprenons qu'il s'agit de l'instruction écrivant `"Hello world !"` dans notre document. Le paramètre à passer est donc la chaîne à écrire. Voici comment nous l'interprétons en PHP :

```
$word->Selection->TypeText("Hello world !");
```

Vous comprenez l'astuce ? Les différences de syntaxe qu'il y a entre PHP et Visual Basic ?

Les constantes : de Visual Basic vers PHP

L'étape d'après consiste à sélectionner le mot "world". Il va donc falloir déplacer le curseur, qui se trouve à la fin de la chaîne "Hello world !", de deux caractères vers la gauche, de manière à le positionner juste après le dernier caractère du mot "world". Voici l'instruction Visual Basic :

```
Selection.MoveLeft Unit:=wdCharacter, Count:=2
```

Comme nous le voyons, c'est encore l'objet Selection qui entre en jeu, par le biais d'une méthode MoveLeft(). Cette méthode nécessite deux paramètres : Unit et Count. Encore une fois, c'est le nom de ces paramètres et de ces méthodes qui va nous permettre d'en comprendre la nature.

Ainsi, la méthode MoveLeft() laisse clairement entendre, de par son nom, qu'elle va déplacer le curseur vers la gauche (et nous en profitons pour déduire qu'il doit exister une méthode MoveRight()). Le premier argument, Unit, demande clairement quelle va être l'unité étalon, et le second argument, Count, demande de combien d'unités nous allons nous déplacer.

Voici comment nous intégrons cela en PHP :

```
$word->Selection->MoveLeft(1, 2);
```

Le second paramètre, Count, étant un nombre, nous le faisons apparaître en tant que tel en PHP. Le premier paramètre, Unit, en revanche, n'est ni une chaîne de caractères (dans le code Visual Basic, le mot wdCharacter n'est pas entre guillemets), ni un nombre. Il s'agit d'une constante propre à Microsoft Word ; c'est pourquoi nous ne pouvons utiliser directement ce nom de constante dans notre code PHP, mais plutôt la valeur de cette constante. Mais alors, comment connaître le contenu de la constante wdCharacter ?

Pour cela, il va nous falloir faire un peu de Visual Basic. Dans l'éditeur Visual Basic, juste en dessous de notre précédente macro, écrivons le code Visual Basic suivant :

Macro ecritConstante

```
Sub ecritConstante()
'
    Selection.TypeText Text:="wdCharacter "
    Selection.TypeText Text:=wdCharacter
    Selection.TypeParagraph
```

```
    Selection.TypeText Text:="wdExtend "
    Selection.TypeText Text:=wdExtend
    Selection.TypeParagraph

    Selection.TypeText Text:="wdToggle "
    Selection.TypeText Text:=wdToggle
    Selection.TypeParagraph
End Sub
```

Cette macro va simplement écrire dans le document Word courant les noms des constantes que nous utilisons dans le fichier *com_word.php* suivis de leurs contenus. Fermez l'éditeur Visual Basic (inutile d'effectuer une sauvegarde). Vous pouvez maintenant lancer votre nouvelle macro dans Microsoft Word en cliquant sur **Macros…** dans le sous-menu **Macro** du menu **Outils**. Sélectionnez votre macro dans la boîte de dialogue qui apparaît, puis cliquez sur **Exécuter**. Dans votre document Word, il y a maintenant trois lignes, et la première vous renseigne sur le contenu de la constante `wdCharacter` : celle-ci vaut `1`.

Maintenant que nous avons déplacé notre curseur juste derrière le mot `"world"`, nous allons le sélectionner. Voici ce que nous dit le code Visual Basic :

```
Selection.MoveLeft Unit:=wdCharacter, Count:=5, Extend:=wdExtend
```

Nous allons encore une fois utiliser la méthode `MoveLeft()` de l'objet `Selection` mais, cette fois, un paramètre de plus a fait son apparition : `Extend`. Il est là pour signifier que nous allons étendre la sélection courante en même temps que nous déplaçons le curseur. Comme il s'agit d'une constante, nous exécutons notre macro `ecritConstante`, et nous apprenons qu'elle vaut `1`. Voici donc comment interpréter cette ligne de Visual Basic dans notre code PHP :

```
$word->Selection->MoveLeft(1, 5, 1);
```

Nous avons sélectionné notre mot. Nous allons le passer en gras. En Visual Basic, cela se dit :

```
Selection.Font.Bold = wdToggle
```

Cette fois, la syntaxe Visual Basic nous indique que nous allons manipuler l'objet `Font`, appartenant lui-même à l'objet `Selection`. Cette fois-ci, ce n'est pas un appel à une méthode que nous allons effectuer. Nous allons simplement changer la valeur de l'attribut `Bold`. Encore une fois nous allons utiliser une constante donc, encore une fois, nous allons lancer notre macro `ecritConstante` pour apprendre que cette constante vaut `9999998`. L'interprétation en PHP est la suivante :

```
$word->Selection->Font->Bold=9999998;
```

Sauvegarder le fichier et libérer les ressources

Il ne nous reste plus qu'a sauvegarder et à libérer les ressources. Voici comment opère le code Visual Basic pour la sauvegarde :

```
ChangeFileOpenDirectory " D:\dynamisez PHP\chap02"
    ActiveDocument.SaveAs FileName:="Helloworld.doc", FileFormat:= _
        wdFormatDocument, LockComments:=False, Password:="",
        ➥ AddToRecentFiles:= _
        True, WritePassword:="", ReadOnlyRecommended:=False,
        ➥ EmbedTrueTypeFonts:= _
        False, SaveNativePictureFormat:=False, SaveFormsData:=False, _
        SaveAsAOCELetter:=False
```

Tout d'abord, nous assignons un "répertoire courant" grâce à la méthode `ChangeFileDirectory()`. Cette méthode dispose de plusieurs arguments dont nous n'avons nul besoin. C'est pourquoi nous la traduirons ainsi au sein de notre fichier PHP :

```
$word->ChangeFileOpenDirectory($repertoire);
```

La seconde méthode, `SaveAs()`, est issue de l'objet `ActiveDocument`. Encore une fois, nous n'avons pas besoin de tous ses arguments :

```
$word->ActiveDocument->SaveAs("test.doc");
```

> **Ne maltraitez pas les arguments !**
>
> Dans cette démonstration, nous n'utilisons pas systématiquement tous les arguments d'une méthode ou fonction. Lorsque vous vous retrouverez face à des fonctions inattendues ou inconnues, prenez le temps de tester pour vérifier que tout marche bien, avant de supprimer des arguments a priori inutiles !

Notre fichier étant sauvegardé, nous devons encore fermer notre instance de Microsoft Word avant de libérer les ressources :

```
//Fermeture de Word
$word->Quit();
// Libération des ressources
$word = null;
```

Notre fichier est terminé, et nous avons appris à comprendre Visual Basic pour l'utiliser avec PHP via COM. Cependant, n'oubliez pas qu'il ne s'agit que d'astuces pour vous éviter de l'apprendre. Si d'aventure, vous êtes amené à générer des documents plus complexes, l'apprentissage de Visual Basic sera nécessaire…

Exporter des données de SQL vers Microsoft Excel

Avec les connaissances acquises depuis le début de ce chapitre, vous devriez y arriver sans trop d'inconvénients Mais nous allons quand même vous accompagner encore un peu dans l'utilisation de COM, ou plutôt dans l'utilisation de Visual Basic en PHP via COM.

Notre base de données

Ce n'est pas la peine de se compliquer la vie. Une seule table avec peu de données suffira. Nous allons donc créer une nouvelle base que nous appellerons "meteo" :

```
CREATE DATABASE 'meteo' ;
```

Au sein de cette base, créons la table "temperature" :

```
CREATE TABLE 'meteo'.'temperature' (
'id' INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
'mois' VARCHAR( 10 ) NOT NULL ,
'temperature' INT NOT NULL
)
```

Remplissons-la ainsi :

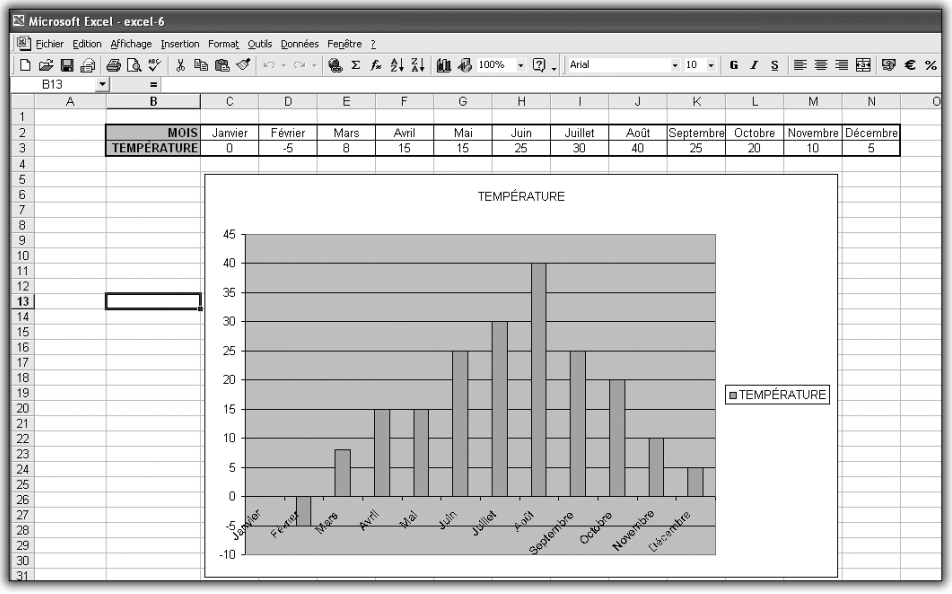
```
INSERT INTO 'meteo'.'temperature' (
'id' ,
'mois' ,
'temperature'
)
VALUES (
NULL , 'Janvier', '0'
), (
NULL , 'Février', '-5'
), (
NULL , 'Mars', '8'
), (
NULL , 'Avril', '15'
), (
NULL , 'Mai', '15'
), (
NULL , 'Juin', '25'
), (
NULL , 'Juillet', '30'
), (
NULL , 'Août', '40'
), (
```

```
NULL , 'Septembre', '25'
), (
NULL , 'Octobre', '20'
), (
NULL , 'Novembre', '10'
), (
NULL , 'Décembre', '5'
);
```

Définir le look final de notre fichier Microsoft Excel

Maintenant que nous avons des données stockées et prêtes à l'emploi, nous devons définir la manière par laquelle ces données seront organisées dans notre fichier Excel. Nous allons donc utiliser Microsoft Excel pour faire une maquette. Nous en profiterons pour enregistrer une nouvelle macro en même temps que nous réaliserons notre maquette.

Pour ce faire, nous utiliserons la même technique qu'avec Microsoft Word ! Lancez Microsoft Excel puis, immédiatement, sélectionnez l'option **Nouvelle Macro…** du sous-menu **Macro**, lui-même se trouvant dans le menu **Outils**. Donnez un nom à votre macro, et c'est parti. N'oubliez pas d'arrêter l'enregistreur, une fois que vous avez fini.



MOIS	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
TEMPÉRATURE	0	-5	8	15	15	25	30	40	25	20	10	5

► Fig. 2.4 : Un tableau et un graphique

Nous nous proposons, sur ce fichier Excel, de faire apparaître un tableau et un graphique.

Récupérer le code Visual Basic de la maquette

Une fois votre maquette terminée, récupérez dans l'éditeur Visual Basic le code de votre macro, et repérez quelles instructions permettent de sélectionner une cellule et d'écrire dedans.

Pour sélectionner une cellule, le code Visual Basic de notre macro nous propose ceci :

```
Range("B2").Select
```

Et pour écrire dans une cellule, le code Visual Basic nous propose cela :

```
ActiveCell.FormulaR1C1 = "MOIS"
```

Maintenant que nous le savons, nous allons écrire une macro qui mettra en évidence chacune des constantes Visual Basic que nous allons utiliser dans PHP :

```
Sub constante()
'
' constante Macro
' Macro enregistrée le 11/11/2007 par Redfish
'

'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "xlRight"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = xlRight

End Sub
```

Cette macro va écrire "`xlRight`" dans la cellule `A1`, et le contenu de la constante `xlRight` dans la cellule `A2`. Il nous faut encore compléter cette macro avec toutes les constantes que nous allons utiliser pour rédiger notre fichier PHP.

Commencer le fichier PHP

À présent, nous avons tout le matériel pour démarrer la rédaction de notre fichier PHP. Toujours dans votre répertoire *chap02*, créez un fichier *com_excel.php* et remplissez-le comme suit :

com_excel.php

```
<?php
$conn=mysql_connect('localhost','root','')
```

```
  or die('Problème lors de la connexion à MYSQL');
mysql_select_db('meteo',$conn)
  or die('Problème lors de la sélection de la base de données');
$query="SELECT * FROM 'temperature' ";
$res=mysql_query($query)
  or die('Problème lors de la réception des enregistrements');

$i = 1;
while($tableau = mysql_fetch_array($res)){
  $i++;
  echo $tableau['mois']." :: ".$tableau['temperature']."<br/>";
}
?>
```

Ce fichier se contente d'aller chercher les données dans la base puis de les afficher sur notre page web.

Ouvrir et fermer une instance de l'application Microsoft Excel

Il va nous falloir ouvrir une instance de Microsoft Excel. Copiez ces lignes au tout début de votre fichier *com_excel.php* :

```
$excel=new COM("Excel.application")
  or die("Impossible d'instancier l'application Excel.<br/>");
```

Lorsque nous aurons effectué toutes les opérations nécessaires, nous devrons fermer l'instance de Excel. Pour ce faire, copiez ces lignes à la toute fin de votre fichier :

```
$excel->Quit();
$excel = null;
```

Travailler sur des feuilles et des classeurs

Comme vous le savez sûrement, un fichier Excel est constitué d'un classeur, lui-même constitué d'une ou plusieurs feuilles. Si, lorsque vous avez enregistré votre macro, vous avez fait des opérations comme la création d'un nouveau fichier, d'une nouvelle feuille, etc., vous pourrez voir les instructions réalisant ces actions dans le code Visual Basic de votre macro et les utiliser dans PHP. Sinon, voici ce qu'il faut rajouter dans votre fichier *com_excel.php* :

```
$excel=new COM("Excel.application")
  or die("Impossible d'instancier l'application Excel.<br/>");

$excel->Workbooks->Add();//Ajout d'un classeur
```

```
$book=$excel->Workbooks(1);//$book contient le classeur actif
$sheet=$book->Worksheets(1);//$sheet contient la feuille active
$sheet->Name = "Température";//Attribution d'un nom à la feuille
```

En procédant de cette manière, le classeur (`$book`) et la feuille (`$sheet`) seront toujours à portée de main…

Sauvegarder le fichier

Comme nous allons tester de nombreuses fois notre fichier au fur et à mesure de sa création, autant écrire les instructions de sauvegarde et, pourquoi pas, rajouter un petit lien pointant vers le fichier Excel que *com_excel.php* aura créé. Modifiez votre code de la manière suivante :

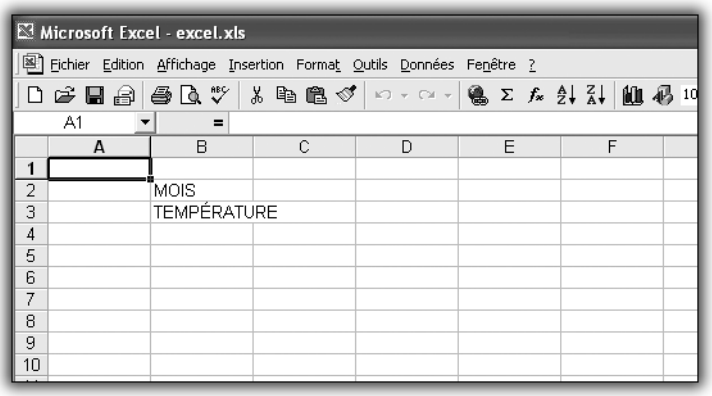
```
$fichier = "D:\dynamisez PHP\chap02\excel.xls";

$book->SaveAs($fichier);//Enregistrement du document
$sheet = null;
$book = null;
$excel->Workbooks->Close();//Fermeture du classeur
$excel->Quit();
$excel = null;
echo "Téléchargez votre <a href='excel.xls'>fichier</a>";
```

Nous en profitons pour ajouter des lignes libérant les ressources occupées par `$sheet` et `$book`.

Réaliser l'en-tête du tableau

Si vous testez votre travail dès maintenant, le fichier *excel.xls* se trouvera bien là où vous le souhaitiez. Maintenant, commençons à le remplir. L'en-tête de notre tableau, seul élément non dynamique, semble être un bon choix pour démarrer. Complétez donc *com_excel.php* ainsi :

```
$cell=$sheet->Range('B2'); // sélection de la cellule
$cell->FormulaR1C1="MOIS"; // écrire dans la cellule
$cell=$sheet->Range('B3');
$cell->FormulaR1C1="TEMPÉRATURE";

$i = 1;
while($tableau = mysql_fetch_array($res)){
  $i++;
  echo $tableau['mois']." :: ".$tableau['temperature']."<br/>";
}
```



► Fig. 2.5 :
C'est un bon début !

Vous pouvez tester votre travail. Et en profiter pour noter un léger souci : si par hasard le fichier *excel.xls* devait déjà exister, votre serveur moulinera sans générer le nouveau fichier. Voici donc comment éviter ce problème :

```
$fichier = "D:\dynamisez PHP\chap02\excel.xls";


if(file_exists("excel.xls"))
  unlink("excel.xls");

$book->SaveAs($fichier);//Enregistrement du document
```

Remplir le tableau avec les données de la base

Comme vous l'avez très justement deviné, c'est dans la boucle `while` de notre code que cela va se passer :

```
$i = ord("B");
while($tableau = mysql_fetch_array($res)){
  $i++;
  echo $tableau['mois']." :: ".$tableau['temperature']."<br/>";
  $cellSelected = chr($i)."2";
  $cell=$sheet->Range($cellSelected);
  $cell->FormulaR1C1=$tableau['mois'];
  $cellSelected = chr($i)."3";
  $cell=$sheet->Range($cellSelected);
  $cell->FormulaR1C1=$tableau['temperature'];
}
```

Comme l'adresse de chaque cellule est définie par un couple lettre/nombre, et que nous allongeons notre tableau horizontalement, il va nous falloir incrémenter la partie "lettre"

de ce système d'adressage. Pour cela, nous allons utiliser les fonctions `ord()` et `chr()`, qui permettent de connaître le code ASCII d'un caractère ou inversement, le caractère lié à un code ASCII.

Comme la toute première cellule que nous allons manipuler au sein de la boucle `while` se trouve dans la colonne *C*, nous allons initialiser notre variable `$i` à `ord("B")` (c'est-à-dire 66). La première instruction de la boucle est d'incrémenter `$i` : le reste de la boucle travaillera alors avec le bon contenu de variable.

Donner du style

Commençons par l'en-tête. Voici ce que nous indique le code Visual Basic :

```
Range("B2:B3").Select
Selection.ColumnWidth = 14.29
Selection.Font.Bold = True
```

Et voici son pendant PHP :

```
//Style de l'en-tête
// Sélectionner les cellules de l'en-tête
$cell = $sheet->Range("B2:B3");
//élargir la sélection
$cell->ColumnWidth = 14.29;
//passer le texte de la sélection en gras
$cell->Font->Bold = true;
//aligner le texte de la sélection à droite
$cell->HorizontalAlignment = -4152;
// mettre le fond de la sélection en gris
$cell->Interior->ColorIndex = 15;
```

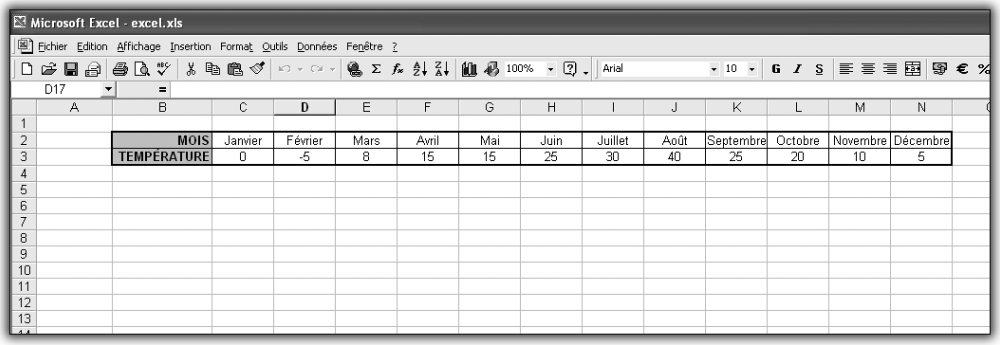
Ce code est, bien entendu, à écrire après que le tableau ait été rempli, soit juste après la boucle `while`.

Nous allons maintenant centrer le contenu de toutes les cellules de notre tableau (sauf celles de l'en-tête), et réduire un peu leur largeur :

```
//centrer les cellules du tableau
// Sélectionner les cellules à centrer
$cell = $sheet->Range('C2:'.chr($i).'3');
$cell->HorizontalAlignment = -4108;
// rétrécir la taille (largeur) de la sélection
$cell->ColumnWidth = 8.5;
```

Il ne vous reste plus qu'à faire apparaître les cadres :

```
// sélectionner l'ensemble du tableau
$cell = $sheet->range('B2:'.chr($i).'3');
//Appliquer une bordure autour de la sélection,
// sur les 4 côtés
for($a = 7 ; $a <= 10 ; $a++){
  // Sélectionner le bon côté de la cellule
  $bordure = $cell->Borders($a);
  $bordure->LineStyle = 1;
  $bordure->Weight = -4138;
}
// Mettre des bordures à l'intérieur de la sélection
for($a = 11 ; $a <= 12 ; $a++){
  // Sélectionner le bon côté de la cellule
  $bordure = $cell->Borders($a);
  $bordure->LineStyle = 1;
  $bordure->Weight = 2;
}
$bordure = null ;
```



► Fig. 2.6 : Un magnifique tableau

Ajouter un graphique

Eh oui ! Que serait Microsoft Excel sans les graphiques ! Regardons tout de suite dans notre macro comment Visual Basic s'y prend :

```
Charts.Add
   ActiveChart.ChartType = xlColumnClustered
   ActiveChart.SetSourceData
Source:=Sheets("Température ").Range("B2:N3"), PlotBy:= _
       xlRows
   ActiveChart.Location Where:=xlLocationAsObject, Name:="Température"
   ActiveSheet.Shapes("Graphique 1").IncrementLeft -89.25
   ActiveSheet.Shapes("Graphique 1").IncrementTop -113.25
```

```
   ActiveSheet.Shapes("Graphique 1").ScaleWidth 1.14, msoFalse, _
       msoScaleFromTopLeft
```

Et voici comment nous l'interprétons avec PHP :

```
//Ajouter un graphique
$graphique = $book->Charts->Add();
$graphique->ChartType = 51; // histogramme
//Sélection des sources du graphique :
$sourceGraphique = $sheet->range('B2:'.chr($i).'3');
$graphique->SetSourceData($sourceGraphique, 1);
$sourceGraphique = null;
// Définir le graphique comme faisant partie
// de la feuille "Température"
$graphique->Location (2, "Température");
// déplacer et dimensionner le graphique
$forme = $sheet->Shapes("Graphique 1");
$forme->IncrementLeft(-89.25);
$forme->IncrementTop(-113.25);
$forme->ScaleWidth(1.14, 0, 0);
$forme = null;
$graphique = null ;
```

Votre fichier Excel est d'ores et déjà terminé, avec son tableau et son graphique ; il est prêt à l'emploi.

Pensez à libérer la mémoire

Lorsque l'on utilise COM pour créer des documents Microsoft Office, les variables que nous créons pour cela – telles que `$excel` ou `$graphique` – sont très gourmandes en mémoire. N'oubliez donc pas de la libérer dès que vous n'en avez plus besoin en assignant la valeur `null` ou en utilisant la fonction `unset()`.

Terminons ce passage sur COM par le code complet du fichier *com_excel.php* :

com_excel.php

```
<?php
$excel=new COM("Excel.application")
  or die("Impossible d'instancier l'application Excel.<br/>");

$excel->Workbooks->Add();//Ajout d'un classeur
$book=$excel->Workbooks(1);//$book contient le classeur actif
$sheet=$book->Worksheets(1);//$sheet contient la feuille active
$sheet->Name = "Température";//Attribution d'un nom à la feuille
```

```
$conn=mysql_connect('localhost','root','')
  or die('Problème lors de la connexion à MYSQL');
mysql_select_db('meteo',$conn)
  or die('Problème lors de la sélection de la base de données');
$query="SELECT * FROM 'temperature' ";
$res=mysql_query($query)
  or die('Problème lors de la réception des enregistrements');

$cell=$sheet->Range('B2'); // sélection de la cellule
$cell->FormulaR1C1="MOIS"; // écrire dans la cellule
$cell=$sheet->Range('B3');
$cell->FormulaR1C1="TEMPÉRATURE";

$i = ord("B");
while($tableau = mysql_fetch_array($res)){
  $i++;
  echo $tableau['mois']." :: ".$tableau['temperature']."<br/>";
  $cellSelected = chr($i)."2";
  $cell=$sheet->Range($cellSelected);
  $cell->FormulaR1C1=$tableau['mois'];
  $cellSelected = chr($i)."3";
  $cell=$sheet->Range($cellSelected);
  $cell->FormulaR1C1=$tableau['temperature'];
}

//Style de l'en-tête
// Sélectionner les cellules de l'en-tête
$cell = $sheet->Range("B2:B3");
//élargir la sélection
$cell->ColumnWidth = 14.29;
//passer le texte de la sélection en gras
$cell->Font->Bold = true;
//aligner le texte de la sélection à droite
$cell->HorizontalAlignment = -4152;
// mettre le fond de la sélection en gris
$cell->Interior->ColorIndex = 15;

//centrer les cellules du tableau
// Sélectionner les cellules à centrer
$cell = $sheet->Range('C2:'.chr($i).'3');
$cell->HorizontalAlignment = -4108;
// rétrécir la taille (largeur) de la sélection
$cell->ColumnWidth = 8.5;

// sélectionner l'ensemble du tableau
$cell = $sheet->range('B2:'.chr($i).'3');
//Appliquer une bordure autour de la sélection,
```

```
// sur les 4 côtés
for($a = 7 ; $a <= 10 ; $a++){
  // Sélectionner le bon côté de la cellule
  $bordure = $cell->Borders($a);
  $bordure->LineStyle = 1;
  $bordure->Weight = -4138;
}
// Mettre des bordures à l'intérieur de la sélection
for($a = 11 ; $a <= 12 ; $a++){
  // Sélectionner le bon côté de la cellule
  $bordure = $cell->Borders($a);
  $bordure->LineStyle = 1;
  $bordure->Weight = 2;
}
$bordure = null;

//Ajouter un graphique
$graphique = $book->Charts->Add();
$graphique->ChartType = 51; // histogramme
//Sélection des sources du graphique :
$sourceGraphique = $sheet->range('B2:'.chr($i).'3');
$graphique->SetSourceData($sourceGraphique, 1);
$sourceGraphique = null;
// Définir le graphique comme faisant partie
// de la feuille "Température"
$graphique->Location (2, "Température");
// déplacer et dimensionner le graphique
$forme = $sheet->Shapes("Graphique 1");
$forme->IncrementLeft(-89.25);
$forme->IncrementTop(-113.25);
$forme->ScaleWidth(1.14, 0, 0);
$forme = null;
$graphique = null;

$fichier = "D:\dynamisez PHP\chap02\excel.xls";

if(file_exists("excel.xls"))
  unlink("excel.xls");

$book->SaveAs($fichier);//Enregistrement du document
$sheet = null;
$book = null;
$excel->Workbooks->Close();//Fermeture du classeur
$excel->Quit();
$excel = null;
echo "Téléchargez votre <a href='excel.xls'>fichier</a>";

?>
```

2.2 DOM : de XML à OpenOffice.org (OOo)

Nous allons maintenant voir comment générer des documents OpenOffice.org. Pour cela, nous allons utiliser l'extension DOM, qui permet de créer et manipuler des documents XML avec PHP, et l'extension ZIP. Mais quel rapport y a-t-il entre ces extensions et les fichiers OOo ? Eh bien, un fichier OOo, c'est tout simplement un ensemble de fichiers XML compressés en une archive…

DOM et XML

DOM (*Document Object Model*) est une extension de PHP qui nous permet de manipuler des fichiers XML. Le but de ce passage n'est pas d'apprendre à utiliser DOM ou XML, mais comme nous en aurons besoin pour arranger quelques fichiers, autant faire le tour de quelques fonctions importantes.

XML et PHP

DOM ne fonctionne, pour le moment, qu'avec PHP5. Si vous ne disposez que de PHP4, vous pouvez utiliser en lieu et place l'extension DOM XML.

Éventuellement, vous pouvez aussi utiliser l'extension SimpleXML pour manipuler des fichiers XML. Comme son nom l'indique, SimpleXML est très simple d'utilisation mais, en contrepartie, ne permet pas autant de choses que DOM.

Commençons donc, dans notre répertoire *chap02*, par créer le fichier *dom_test.xml* :

dom_test.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animaux>
  <famille nom="oiseau">
    <poule>
      La poule domestique (Gallus gallus domesticus)
      est un oiseau de l'ordre des galliformes.
    </poule>
    <pigeon>
      Les pigeons (genre Columba) sont des oiseaux de
      la famille des Columbidae.
    </pigeon>
  </famille>
<famille nom="poisson">
    <hareng>
      Le hareng (Clupea harengus) est un poisson vivant
```

```
      en grands bancs.
    </hareng>
  </famille>
</animaux>
```

Ouvrir un fichier XML et récupérer le nœud racine

Toujours dans le répertoire *chap02*, créez un fichier nommé *dom_test.php* et remplissez-le comme suit :

dom_test.php

```
<?php
$dom = new DomDocument();
$dom->load('dom_test.xml');

// Récupérer la racine
$animaux = $dom->firstChild;
echo "<b>noeud racine : </b>".$animaux->tagName;
echo "<br/>";
?>
```

L'objet `DomDocument` est l'objet de base. Il représente un document XML dans son intégralité. C'est d'ailleurs à partir de cet objet que nous chargeons notre fichier XML, via la méthode `DomDocument::load()`.

Pour récupérer le nœud racine de notre document, nous allons utiliser la propriété `firstChild` de l'objet `DomDocument`. Cette propriété renvoie le premier nœud enfant sous la forme d'un objet `DomNode`.

Dans notre exemple, la variable `$animaux` contient l'intégralité du nœud `<animaux>`, enfants compris.

Cet objet `DomNode` dispose d'une propriété `tagName` qui renvoie, sous la forme d'une chaîne de caractères, le nom de notre balise.

Obtenir une liste de nœuds grâce au nom des balises

L'extension DOM nous permet de faire une recherche de nœuds dans l'intégralité d'un document, et ce à partir d'un nom de balise. Complétez donc le fichier *dom_test.php* par le code suivant :

```
// Obtenir une liste de noeuds par leur nom
$listeFamille = $dom->getElementsByTagName('famille');
```

Nous utilisons la variable `$dom`, c'est-à-dire l'objet `DomDocument` contenant l'intégralité de notre document XML, afin de faire une recherche dans l'ensemble du document, sans s'occuper de la hiérarchie des éléments.

La méthode `DomDocument::getElementsByTagName()` recherche dans tout le document les nœuds dont le nom de balise est `"famille"`. Elle renvoie le résultat sous la forme d'un objet `DomNodeList`.

> ⚠ **Une liste n'est pas un tableau**
>
> L'objet `DomNodeList` n'est pas un tableau. Il est hors de question d'accéder au contenu de cette objet par une syntaxe à crochets, donc ! Pour ce faire, utilisez plutôt la méthode `DomNodeList::index()`…

Vous souhaitez connaître le nombre de résultats renvoyés par la recherche ?

```
// Savoir combien il y a de noeuds dans une liste
echo "noeud racine a <b>".$listeFamille->length."</b> noeuds enfants";
```

Il nous suffit d'utiliser la propriété `length` de l'objet `DomNodeList`.

Nous allons maintenant passer en revue les nœuds contenus dans `$listeFamille` afin d'y appliquer un traitement. Nous utilisons à cet effet la structure `foreach`, bien que d'autres solutions soient possibles :

```
// Passer en revue les noeuds d'une liste
echo "<ol>";
foreach($listeFamille as $famille){
// obtenir la valeur d'un attribut
  echo "<li><u>".$famille->getAttribute('nom')."</u><br/>";
}
echo "</ol>";
```

▶ Fig. 2.7 : Un bon début

Vous pouvez noter l'usage de la méthode `DomNode::getAttribute()` qui nous permet de récupérer, sous la forme d'une chaîne de caractères, le contenu d'un des attributs du nœud, en l'occurrence, le nom de la famille.

Passer en revue les enfants d'un nœud

Nous allons maintenant vérifier la présence (ou non) d'éventuels enfants au sein de chaque famille, puis les lister. Modifions notre code ainsi :

```
foreach($listeFamille as $famille){
  // obtenir la valeur d'un attribut
  echo "<li><u>".$famille->getAttribute('nom')."</u><br/>";

  // Vérifier si chaque famille dispose d'enfants
  if(!$famille->hasChildNodes()){
    echo "Cette famille n'a pas d'enfants<br/>";
  } else {
    // futurs traitements
  }
}
```

La méthode `DomNode::hasChildNodes()`, comme son nom l'indique, renvoie un booléen permettant de savoir si notre nœud dispose d'enfants.

Continuons sur notre lancée et appliquons un traitement aux éventuels enfants de chaque famille. Complétez le bloc `else` de la manière suivante :

```
else {

  // récupérer le premier noeud enfant
  $animal = $famille->firstChild;

    //Passer en revue les noeuds d'une liste
    //sans en connaître les noms
    echo "<ul>";
    while($animal){

      // vérifie qu'il s'agit bien d'un DOMElement
      if($animal->nodeType == 1){
        echo "<li><b>".$animal->tagName."</b><br/>";
        echo utf8_decode($animal->firstChild->nodeValue);
      }

      // avancer au noeud suivant
      $animal = $animal->nextSibling;
    }
```

```
    echo "</ul>";
  }
```

La première chose à faire est de récupérer le premier enfant d'une famille. À ce titre, nous utilisons la propriété `firstChild` de l'objet `DomNode`.

Méthodes homonymes

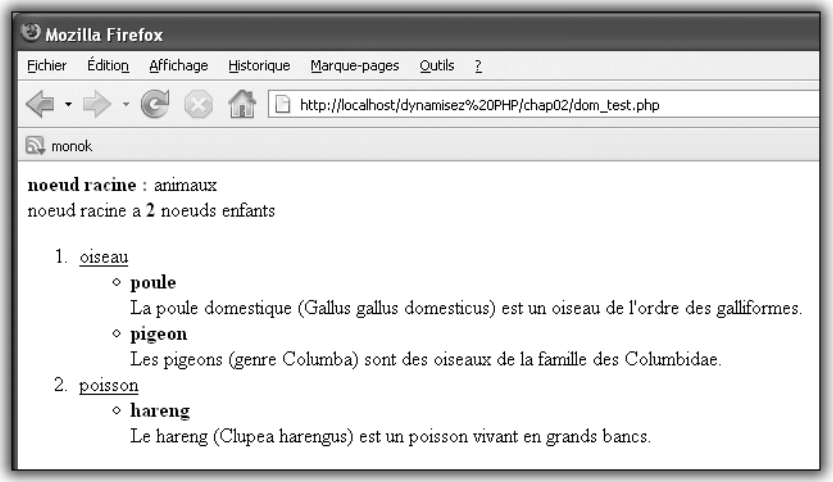
Nous avons déjà utilisé la propriété `firstChild`, mais avec l'objet `DomDocument`. Faites attention de bien savoir quel objet vous êtes en train de manipuler, car si les objets `DomDocument` et `DomNode` disposent de propriétés et de méthodes homonymes, elles ne renvoient pas forcément les mêmes choses !

Ensuite, nous vérifions le type du nœud auquel nous avons affaire, en utilisant la propriété `nodeType` de l'objet `DomNode`. En effet, nous ne cherchons à avoir affaire qu'avec des nœuds éléments, quand un nœud peut être un attribut ou du texte.

Si notre nœud est bien du type voulu, nous affichons son nom de balise qui, dans notre exemple, représente aussi le nom de l'animal, grâce à la propriété `tagName`.

Nous allons aussi afficher le petit texte contenu par ce nœud. Pour cela, nous devons d'abord atteindre le premier enfant de notre nœud. En effet, le texte est considéré comme étant un nœud texte. Ensuite, la propriété `nodeValue` renvoie le contenu de notre nœud texte sous forme de chaîne de caractères.

Enfin, nous passons d'un nœud à l'autre grâce à la propriété `nextSibling`. Cette dernière renvoie le nœud suivant, ou `null` s'il n'y en a pas.



► Fig. 2.8 : Tout est affiché.

Voici le code complet du fichier *dom_test.php* :

dom_test.php

```
<?php
$dom = new DomDocument();
$dom->load('dom_test.xml');

// Récupérer la racine
$animaux = $dom->firstChild;
echo "<b>noeud racine : </b>".$animaux->tagName;
echo "<br/>";

// Obtenir une liste de noeuds par leur nom
$listeFamille = $dom->getElementsByTagName('famille');

// Savoir combien il y a de noeuds dans une liste
echo "noeud racine a <b>".$listeFamille->length."</b> noeuds enfants";
echo "<br/>";

// Passer en revue les noeuds d'une liste
echo "<ol>";
foreach($listeFamille as $famille){
  // obtenir la valeur d'un attribut
  echo "<li><u>".$famille->getAttribute('nom')."</u><br/>";

  // Vérifier si chaque famille dispose d'enfants
  if(!$famille->hasChildNodes()){
    echo "Cette famille n'a pas d'enfants<br/>";
  } else {

    // récupérer le premier noeud enfant
    $animal = $famille->firstChild;

    //Passer en revue les noeuds d'une liste
    //sans en connaître les noms
    echo "<ul>";
    while($animal){

      // vérifie qu'il s'agit bien d'un DOMElement
      if($animal->nodeType == 1){
        echo "<li><b>".$animal->tagName."</b><br/>";
        echo utf8_decode($animal->firstChild->nodeValue);
      }

      // avancer au noeud suivant
      $animal = $animal->nextSibling;
    }
    echo "</ul>";
```

```
  }
}
echo "</ol>";

?>
```

Ajouter un nœud

Avec DOM, lorsque vous souhaitez ajouter un nœud à un document XML, il vous faut d'abord le créer. Démarrons un nouveau fichier PHP que nous appellerons *dom_ajouter .php*, et remplissons-le de la manière suivante :

dom_ajouter.php

```
<?php
$dom = new DomDocument();
$dom->load('dom_test.xml');

// créer le noeud sans l'ajouter à l'arbre XML
$nouvelleFamille = $dom->createElement("famille");

// créer un attribut "nom" à ce noeud
// et lui attribuer une valeur
$nouvelleFamille->setAttribute("nom", "reptile")
?>
```

L'objet `DomDocument` dispose de la méthode `DomDocument::createElement()`, qui permet de créer, en mémoire seulement, un nouveau nœud. Dans notre exemple, nous créons un nouveau nœud `"famille"`.

La méthode `DomElement::setAttribute()` ajoute un attribut à l'élément. Le premier paramètre de cette méthode indique le nom de l'attribut, et le second indique la valeur de cet attribut.

Le nœud que nous venons de créer ressemble à cela :

```
<famille nom="reptile" />
```

Pour le moment, notre nœud ne dispose pas d'enfants. Corrigeons cela. Rajoutez, à la fin de *dom_ajouter.php*, le code suivant :

```
//créer le nouveau noeud "animal"
$nouvelAnimal = $dom->createElement("serpent");

$desc = "Les serpents (sous-ordre des Serpentes) " .
  "sont des reptiles au corps cylindrique " .
```

```
  "et allonge, dépourvus de membres apparents." .
  " Ils partagent cette derniere caracteristique" .
  " avec un groupe de vertebres " .
  "tetrapodes : les gymnophiones, qui appartiennent" .
  " au groupe des lissamphibiens.";

//créer le DOMText de description
$description = $dom->createTextNode($desc);

// ajouter le noeud texte au noeud $nouvelAnimal
$nouvelAnimal->appendChild($description);

// ajouter le noeud $nouvelAnimal
// au noeud $nouvelleFamille
$nouvelleFamille->appendChild($nouvelAnimal);
```

Nous faisons appel à deux nouvelles méthodes. Tout d'abord la méthode `DomDocument::createTextNode()` qui, comme son nom l'indique, va créer un nouveau nœud de type texte. Ce nouveau nœud n'est encore relié à rien, et surtout pas à notre document.

La seconde méthode, `DomElement::appendChild()`, permet enfin de lier un nœud à un autre. Dans notre exemple, le nœud `$nouvelAnimal` devient un enfant du nœud `$nouvelleFamille`. Voici à quoi ressemble maintenant le nœud `$nouvelleFamille` :

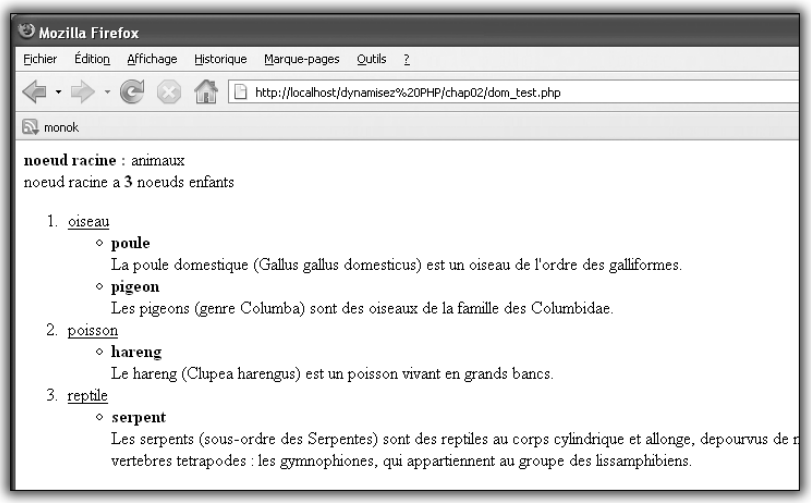
```
<famille nom="reptile">
  <serpent>
    Les serpents (sous-ordre des Serpentes)…
    …au groupe des lissamphibiens.
  </serpent>
</famille>
```

Passons maintenant à la dernière étape : lier notre nouveau nœud à notre document. Pour ce faire, nous allons utiliser encore une fois la méthode `DomElement::appendChild()`, mais nous allons l'appliquer depuis le nœud de notre document que nous souhaitons voir compléter. Ainsi, dans notre exemple, nous créons un nœud `"famille"` qui doit s'ajouter à la suite des autres nœuds `"famille"` de notre document. Ces nœuds sont des enfants du nœud racine, `"animaux"`. Nous allons donc commencer par isoler le nœud racine, puis lui ajouter un nouvel enfant. Voici donc les quelques lignes à rajouter à *dom_ajouter.php* :

```
// Récupérer la racine
$animaux = $dom->firstChild;

// ajouter le noeud $nouvelleFamille
// au noeud $animaux
$animaux->appendChild($nouvelleFamille);
```

```
// on n'oublie pas de sauvegarder notre document modifié :
$dom->save('dom_test.xml');
```

Et voilà ! N'oubliez pas d'utiliser la méthode `DomDocument::save()` pour sauvegarder vos modifications dans le fichier *dom_test.xml*.



► Fig. 2.9 : Un nœud de plus

Vous pourrez voir les modifications en ouvrant le fichier *dom_test.php* avec votre navigateur préféré.

Effacer un nœud

Nous venons de créer un nouveau nœud `"famille"`. À présent, nous allons l'effacer. Pour ce faire, créez un nouveau fichier, *dom_effacer.php*, et remplissez-le comme suit :

dom_effacer.php

```
<?php
//ouvrir le fichier content.xml
$dom = new DomDocument();
$dom->load('dom_test.xml');
$animaux = $dom->firstChild;

// obtenir la liste des noeuds "famille"
$listeFamille = $dom->getElementsByTagName('famille');

// isoler le noeud "reptile"
$reptile = null;
```

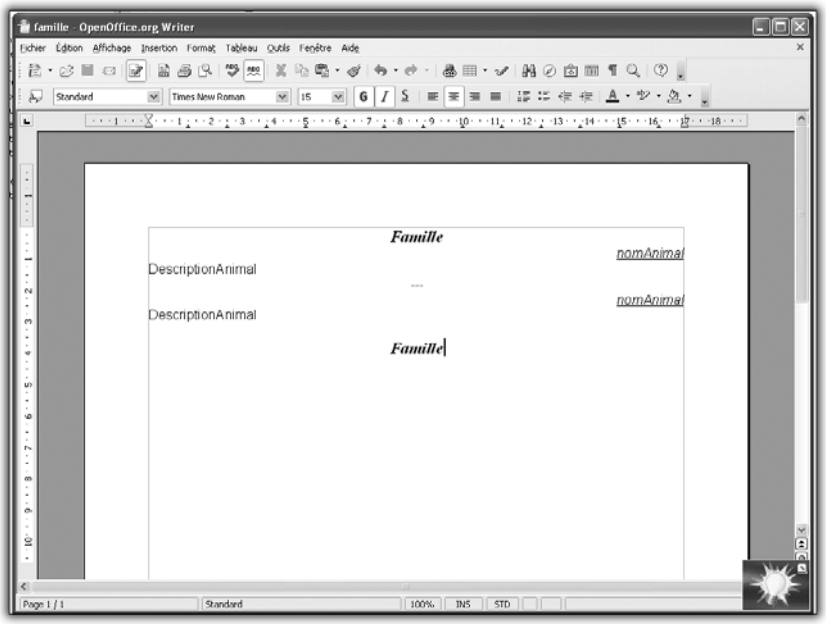
```
foreach($listeFamille as $famille){
  if($famille->getAttribute('nom') == 'reptile'){
    $reptile = $famille;
    break;
  }
}

// effacer le noeud $reptile
$animaux->removeChild($reptile);

$dom->save('dom_test.xml');
?>
```

Il n'y a rien de bien compliqué : on commence par isoler dans une variable le nœud que l'on souhaite supprimer, ainsi que son nœud parent. Ensuite, grâce à la méthode `DomNode::removeChild()`, on le supprime purement et simplement.

XML et OOo

Le fichier OOo Writer que nous allons créer ensemble va présenter les informations contenues dans le fichier *dom_test.xml*. Comme nous l'avons déjà fait pour créer des documents Microsoft Office, nous allons partir d'une maquette.

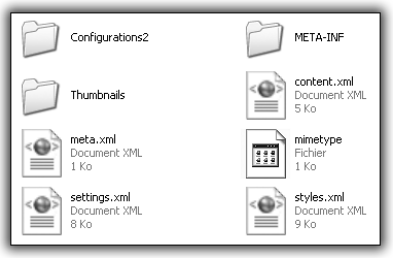
Ouvrez OOo Writer et faites une maquette.



► Fig. 2.10 : Notre maquette OOo Writer

Enregistrez votre maquette sous le nom *famille.odt*, dans votre répertoire *chap02*.

Vous allez maintenant ouvrir ce fichier avec un logiciel de type Winrar ou 7-Zip, et en extraire le contenu.



▶ Fig. 2.11 :
Le contenu de l'archive famille.odt

Comme vous pouvez le constater, nous n'avons ici que des documents XML. Rassurez-vous, nous n'allons pas tous les modifier. Nous nous contenterons de manipuler le fichier *content.xml*. Vous pouvez donc copier ce fichier dans votre répertoire *chap02*, et effacer tous les autres.

Ouvrez maintenant *content.xml*. Voici comment se présente son architecture :

content.xml

```
<office:document-content>

  <office:scripts/>

  <office:font-face-decls>
  </office:font-face-decls>

  <office:automatic-styles>
  </office:automatic-styles>

  <office:body>

    <office:text>

      <office:forms/>

      <text:sequence-decls>
      </text:sequence-decls>

      <text:p></text:p>
      <text:p></text:p>
      <text:p></text:p>

    </office:text>
```

```
  </office:body>

</office:document-content>
```

L'espace de noms

Dans ce fichier, la première chose que l'on remarque, ce sont les noms des balises.

En effet, les noms de balises sont composés de deux mots séparés par le signe ":". Le premier mot représente ce que l'on appelle l'espace de noms. Ce mot lie chacune des balises avec une définition *xmlns* (toutes ces définitions sont écrites sous la forme d'attributs du nœud racine).

Le second mot représente tout simplement le nom de la balise.

C'est une spécificité qu'il nous faudra prendre en compte lorsque nous modifierons ce fichier. Ainsi, lorsque nous souhaitons créer un nouveau nœud, nous n'utiliserons pas la méthode `DomDocument::createElement()`, mais la méthode `DomDocument::createElementNS()`. Voici un exemple :

```
$nouveauNoeudNS = $dom->createElementNS('text', 'p');
//crée un noeud
//<text:p />
```

Le premier argument de la méthode indique l'espace de noms, et le second indique le nom.

Créer un fichier OOo Writer avec DOM et PHP

Toujours dans notre fichier *content.xml*, nous allons tenter de repérer où a été placé le contenu. Vous l'avez trouvé ?

```
<text:p text:style-name="P1">Famille</text:p>
<text:p text:style-name="P3">nomAnimal</text:p>
<text:p text:style-name="P2">DescriptionAnimal</text:p>
<text:p text:style-name="P4">---</text:p>
<text:p text:style-name="P3">nomAnimal</text:p>
<text:p text:style-name="P2">DescriptionAnimal</text:p>
<text:p text:style-name="Standard"/>
<text:p text:style-name="P1">Famille</text:p>
```

Il s'agit des balises "p" appartenant à l'espace de noms "text". Notez que chacune des ces balises contient un attribut "text :style-name". Cet attribut indique le style du texte, c'est-à-dire sa police de caractères, son alignement, etc.

Nous allons, dans un nouveau fichier PHP, commencer par effacer toutes ces balises, puis les remplacer par de nouvelles, contenant les informations de *dom_test.xml*.

Effacer les vieilles balises

Toujours dans votre répertoire *chap02*, créez le fichier *ooo_writer.php*, et remplissez-le de la manière suivante :

ooo_writer.php

```
<?php
//ouvrir le fichier content.xml
$dom = new DomDocument();
$dom->load('content.xml');

$racine = $dom->firstChild;

// l'espace de nom "text"
$ns = "urn:oasis:names:tc:opendocument:xmlns:text:1.0";

// récupérer la liste des noeuds appartenant au
// name systeme $ns et dont le nom du tag est "p"
$listeTextp = $dom->getElementsByTagNameNS($ns, 'p');

// récupérer la liste des noeuds dont
// le nom du tag est "text" (<office:text>)
$li = $dom->getElementsByTagName('text');
$off = null; // contiendra le seul noeud de ce type
foreach($li as $l){
  $off = $l;
}

// effacer les anciens noeuds <text:p>
while($listeTextp->length > 0){
  $off->removeChild($listeTextp->item(0));
}
?>
```

Vous remarquez que nous créons une variable `$ns` contenant une drôle d'adresse… Il s'agit de l'espace de nom `"text"`, que nous avons récupéré dans notre fichier *content.xml* (l'espace de nom est indiqué sous la forme d'un attribut dans le nœud racine).

Comme vous le voyez, il n'y a rien de bien compliqué, si ce n'est que nous utilisons la méthode `DomDocument::getElementsByTagNameNS()`, au lieu de la méthode `DomDocument::getElementsByTagName()`, qui fonctionne exactement de la même manière que la méthode `DomDocument::createElementNS()`. Vous pouvez néanmoins retrouver vos nœuds sans faire de recherche incluant l'espace de noms ; seulement avec le nom de balise. C'est d'ailleurs ce que nous faisons pour retrouver la balise `<office:text>` (mais nous vous le déconseillons).

Ajouter les nouvelles balises

Cela ne devrait pas vous sembler si difficile. Voici comment procéder (rajoutez ce code à la fin de *ooo_writer.php*) :

```
// ouvrir le fichier dom_test.xml
$domSource = new DomDocument();
$domSource->load('dom_test.xml');

//récupérer la liste des familles
$listeFamille = $domSource->getElementsByTagName('famille');

//chaque famille doit être traitée :
foreach($listeFamille as $l){

  // ajouter un noeud pour le "titre" de cette famille
  // dans le document content.xml
  $titreFamille = $l->getAttribute('nom');
  $nouveauNoeud = $dom->createElementNS($ns, 'p', $titreFamille);
  // en se référant à content.xml, on attribue le style P1
  $nouveauNoeud->setAttributeNS($ns, 'style-name', 'P1');
  $off->appendChild($nouveauNoeud);

  // récupérer le premier noeud enfant de cette famille
  $animal = $l->firstChild;

  // traiter tous les noeuds de cette famille
  $noeudAnimal = null;
  while($animal){

    // vérifie qu'il s'agit bien d'un DOMElement
    if($animal->nodeType == 1){

      // Créer le noeud pour le nom de l'animal
      $nomAnimal = $animal->tagName;
      $noeudAnimal = $dom->createElementNS($ns, 'p', $nomAnimal);
      // en se référant à content.xml, on attribue le style P3
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P3');
      $off->appendChild($noeudAnimal);

      // Créer le noeud pour la description
      $description = $animal->firstChild->nodeValue;
      $noeudAnimal = $dom->createElementNS($ns, 'p', $description);
      // en se référant à content.xml, on attribue le style P2
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P2');
      $off->appendChild($noeudAnimal);

      // Créer le noeud "---" entre chaque animal
      $noeudAnimal = $dom->createElementNS($ns, 'p', '---');
      // en se référant à content.xml, on attribue le style P4
```

```
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P4');
      $off->appendChild($noeudAnimal);
    }
    $animal = $animal->nextSibling;
  }
  if($noeudAnimal != null){
    $off->removeChild($noeudAnimal);
  }
  // ajouter un noeud pour sauter une ligne
  $nouveauNoeud = $dom->createElementNS($ns, 'p');
  $nouveauNoeud->setAttributeNS($ns, 'style-name', 'Standard');
  $off->appendChild($nouveauNoeud);
}
$dom->save('content.xml');
```

Il n'y a rien de sorcier, n'est-ce pas ? Nous attirons tout de même votre attention sur un point :

```
$noeudAnimal = $dom->createElementNS($ns, 'p', $nomAnimal);
```

Un troisième argument a été ajouté à la méthode. En fait, cela revient à faire :

```
$noeudAnimal = $dom->createElementNS($ns, 'p');
$noeudTexteAnimal = $dom->createTextNode($nomAnimal);
$noeudAnimal->appendChild($noeudTexteAnimal);
```

Ajouter le fichier content.xml à l'archive famille.odt

Maintenant que nous avons refait le fichier *content.xml* à notre convenance, il ne nous reste plus qu'une seule chose à faire : mettre ce fichier dans l'archive *famille.odt*, qui n'est autre que notre fichier OOo.

Il vous faut ajouter ce code à la fin de votre fichier *ooo_writer.php* :

```
//Sauvegarder en un fichier OOo :
$odt = new ZipArchive;

if ($odt->open('famille.odt', ZIPARCHIVE ::CREATE) === TRUE){
  // Ajouter (remplacer) le fichier content.xml
  $odt->addFile('content.xml', 'content.xml');
  $odt->close();
  echo "ok";

}
```

***Pour plus de détails sur la compression, n'hésitez pas à vous référer au chapitre** Compresser c'est gagner…*

► Fig. 2.12 : Et voici le résultat final

Pour conclure ce chapitre, voici le code complet de *ooo_writer.php* :

ooo_writer.php

```
<?php
//ouvrir le fichier content.xml
$dom = new DomDocument();
$dom->load('content.xml');

$racine = $dom->firstChild;

// l'espace de nom "text"
$ns = "urn:oasis:names:tc:opendocument:xmlns:text:1.0";

// récupérer la liste des noeuds appartenant au
// name systeme $ns et dont le nom du tag est "p"
$listeTextp = $dom->getElementsByTagNameNS($ns, 'p');

// récupérer la liste des noeuds dont
// le nom du tag est "text" (<office:text>)
```

```
$li = $dom->getElementsByTagName('text');
$off = null; // contiendra le seul noeud de ce type
foreach($li as $l){
  $off = $l;
}

// effacer les anciens noeuds <text:p>
while($listeTextp->length > 0){
  $off->removeChild($listeTextp->item(0));
}

// on peut commencer à créer de nouveaux noeuds

// ouvrir le fichier dom_test.xml
$domSource = new DomDocument();
$domSource->load('dom_test.xml');

//récupérer la liste des familles
$listeFamille = $domSource->getElementsByTagName('famille');

//chaque famille doit être traitée :
foreach($listeFamille as $l){

  // ajouter un noeud pour le "titre" de cette famille
  // dans le document content.xml
  $titreFamille = $l->getAttribute('nom');
  $nouveauNoeud = $dom->createElementNS($ns, 'p', $titreFamille);
  $nouveauNoeud->setAttributeNS($ns, 'style-name', 'P1');
  $off->appendChild($nouveauNoeud);

  // récupérer le premier noeud enfant de cette famille
  $animal = $l->firstChild;

  // traiter tous les noeuds de cette famille
  $noeudAnimal = null;
  while($animal){

    // vérifie qu'il s'agit bien d'un DOMElement
    if($animal->nodeType == 1){

      // Créer le noeud pour le nom de l'animal
      $nomAnimal = $animal->tagName;
      $noeudAnimal = $dom->createElementNS($ns, 'p', $nomAnimal);
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P3');
      $off->appendChild($noeudAnimal);

      // Créer le noeud pour la description
      $description = $animal->firstChild->nodeValue;
      $noeudAnimal = $dom->createElementNS($ns, 'p', $description);
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P2');
```

```
      $off->appendChild($noeudAnimal);

      // Créer le noeud "---" entre chaque animal
      $noeudAnimal = $dom->createElementNS($ns, 'p', '---');
      $noeudAnimal->setAttributeNS($ns, 'style-name', 'P4');
      $off->appendChild($noeudAnimal);
    }

    $animal = $animal->nextSibling;
  }

  if($noeudAnimal != null){
    $off->removeChild($noeudAnimal);
  }

  // ajouter un noeud pour sauter une ligne
  $nouveauNoeud = $dom->createElementNS($ns, 'p');
  $nouveauNoeud->setAttributeNS($ns, 'style-name', 'Standard');
  $off->appendChild($nouveauNoeud);

}

$dom->save('content.xml');

//Sauvegarder en un fichier OOo :
$odt = new ZipArchive;

if ($odt->open('famille.odt', ZIPARCHIVE ::CREATE) === TRUE){

  // Ajouter (remplacer) le fichier content.xml
  $odt->addFile('content.xml', 'content.xml');
  $odt->close();
  echo "ok";

}
?>
```

2.3 Check-list

Voici la fin de ce long chapitre consacré à la bureautique. Nous avons vu comment :

- ✔ créer un fichier Microsoft Office en utilisant COM ;
- ✔ créer une maquette et sa macro ;
- ✔ comprendre le code Visual Basic d'une macro et l'interpréter dans PHP ;
- ✔ utiliser DOM ;
- ✔ créer un document OpenOffice.org.

3

Manipuler les images

PHP ne se limite pas à la génération de pages HTML. Dans ce chapitre, nous allons voir comment il est possible de générer dynamiquement des images dans les formats les plus courants sur le Web, d'en créer à partir d'images déjà existantes, ou d'en créer à partir de rien…

3.1 GD2 : En toute simplicité

Nous commençons notre exploration du monde de l'image façon PHP par GD2, une extension qui vous permettra de réaliser simplement et rapidement la plupart des tâches à effectuer sur des images, comme les réduire ou y écrire une adresse URL.

Créer une image dynamiquement

Afin d'aborder simplement GD2, nous allons commencer par voir comment on appelle une image déjà existante. Créez dans votre éditeur favori un nouveau fichier : *dynamisez_php/chap03/gd.php*. À côté de ce fichier, créez un dossier *images*. Placez une image dans ce dossier puis, dans votre fichier *gd.php*, entrez le code suivant :

gd.php

```
<?php

header("Content-type: image/jpeg");

$im = @imagecreatefromjpeg("images/ciel2.jpg")
  or die("impossible de créer un flux GD2");
// remplacez "ciel2.jpg" par le nom de votre image
imagejpeg($im);

imagedestroy($im);

?>
```

Lorsque vous ouvrez la page *gd.php* depuis votre navigateur préféré, votre image apparaît à l'écran. Vous pouvez l'enregistrer sur votre disque dur, comme n'importe quelle image affichée dans une page web. Vous pouvez aussi faire appel à cette image depuis du code HTML. Créez, dans le même répertoire que votre fichier *gd.php*, un fichier *html_gd.html* contenant le code suivant :

html_gd.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
➥ "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type"
     content="text/html; charset=iso-8859-1" />
      <title>Image GD2</title>
  </head>
  <body>
    <img alt="image dynamique avec GD2" src="gd.php" />
```

```
  </body>
</html>
```

Lancez le fichier *html_gd.html* avec votre navigateur : l'image apparaît comme telle, alors que l'attribut `src` de la balise `img` pointe vers un fichier PHP.

Observons notre code PHP instruction par instruction, afin de découvrir les rouages de base de GD2 :

```
header("Content-type: image/jpeg");
```

Cette première instruction sert à renseigner l'en-tête de notre fichier *gd.php* une fois qu'il aura été généré. C'est grâce à cet en-tête que votre navigateur interprètera le fichier généré comme étant une image. Il est nécessaire de placer cette instruction avant toute autre instruction d'écriture.

Les en-têtes Content-type compatibles avec GD2

Avec GD2, vous pouvez générer autre chose que des images JPEG. Remplacez donc l'argument de la fonction `header()` en fonction du type d'image que vous souhaitez obtenir :

```
header("Content-type: image/jpeg"); // en-tête JPEG
header("Content-type: image/gif"); // en-tête GIF
header("Content-type: image/png"); // en-tête PNG
header("Content-type: image/wbmp"); // en-tête WBMP (attention : la
➥ plupart des navigateurs ne reconnaissent pas les types WBMP)
```

Une fois l'en-tête correctement placé, nous chargeons notre image en mémoire :

```
$im = imagecreatefromjpeg("images/ciel2.jpg");
```

La fonction `imagecreatefromjpg()` va chercher l'image JPEG de notre choix. Cette fonction renvoie un objet de type `image` que nous stockons dans une variable pour une utilisation ultérieure.

Les autres types d'images

L'extension GD2 permet de charger d'autres types que le JPEG. Utilisez l'une de ces fonctions selon vos besoins :

```
// créer la ressource depuis une image GIF. :
$im = imagecreatefromgif("images/ciel2.gif");
// créer la ressource depuis une image PNG :
$im = imagecreatefrompng("images/ciel2.png
// créer la ressource depuis une image WBMP :
$im = imagecreatefromwbmp("images/ciel2.wbmp");
```

Maintenant que notre image est chargée, il nous faut l'afficher :

```
imagejpeg($im) ;
```

Il s'agit d'une instruction d'écriture, c'est-à-dire que c'est ce qui va être écrit dans le fichier généré par PHP. Il faut donc impérativement que vous ayez spécifié l'en-tête de votre image (fonction `header()`) avant cette instruction.

Les autres types d'images

Comme nous pouvons charger différents types d'images, GD2 nous permet aussi de créer des images de différents types : : :

```
imagejpeg($im) // créer une image JPEG
imagepng($im) // créer une image PNG
imagegif($im) // créer une image GIF
imagewbmp($im) // créer une image WBMP
```

La toute dernière instruction concerne la destruction de l'image `$im` :

```
imagedestroy($im) ;
```

Cette instruction permet de libérer la mémoire utilisée par notre image. En contrepartie, nous ne pourrons plus la modifier.

► Fig. 3.1 : L'image ciel2.jpg telle qu'affichée avec le fichier html_gd.html

Convertir une image

Voici comment convertir ses images le plus simplement du monde grâce à GD2. Modifiez votre fichier *gd.php* comme suit :

gd.php

```
<?php

// nous allons renvoyer une image GIF :
header("Content-type: image/gif");
// nous créons notre ressource à partir d'une image JPEG :
$im = @imagecreatefromjpeg("images/ciel2.jpg")
  or die("impossible de créer un flux GD2");
// nous créons une image GIF :
imagegif($im);
imagedestroy($im);

?>
```

La fonction `imagegif()` convertit automatiquement notre image de type JPEG en une image de type GIF. Il en va naturellement de même pour les autres fonctions. Quel que soit le type de notre image, `imagejpeg()` créera toujours une image JPEG, `imagepng()` une image PNG, etc.

La seule règle à respecter est l'adéquation entre la fonction `header()` et la fonction d'écriture.

Sauvegarder une image créée dynamiquement

La sauvegarde d'images se fait, comme la conversion, grâce aux fonctions d'écriture (`imagejpeg()`, `imagegif()`, `imagepng()`…). Nous allons modifier à nouveau notre fichier *gd.php* :

gd.php

```
<?php

header("Content-type: image/jpeg");
$im = @imagecreatefromjpeg("images/ciel2.jpg")
  or die("impossible de créer un flux GD2");
imagejpeg($im, "images/ciel2_copie.jpg");
imagedestroy($im);

?>
```

La seule différence est l'ajout d'un argument à la fonction `imagejpeg()` (ou à n'importe quelle autre fonction d'écriture) : une chaîne de caractère indique l'emplacement et le nom de l'image à sauvegarder.

Lorsque nous lançons dans notre navigateur le fichier *html_gd.html*, rien ne s'affiche si ce n'est le contenu de l'attribut `alt` de notre balise `img`. Par contre, le fichier *ciel2_copie.jpg* a bien été créé dans le répertoire *images*. En effet, lorsqu'elles ne disposent que d'un seul argument, les fonctions d'écriture écrivent dans le fichier généré par PHP (dans notre cas, l'image est écrite dans *gd.php*). Si ces fonctions d'écriture disposent de deux arguments, le flux d'écriture n'est pas dirigé vers le fichier généré, mais vers le fichier image à créer.

Si nous voulons que notre image soit créée à la fois sur le disque dur et à la fois dans le fichier généré, nous avons besoin d'écrire deux flux dans notre code ; soit deux fonctions d'écriture :

gd.php

```
<?php

header("Content-type: image/jpeg");
$im = @imagecreatefromjpeg("images/ciel2.jpg")
  or die("impossible de créer un flux GD2");

// créer l'image dans le fichier généré
imagejpeg($im);

// créer l'image sur le disque dur
imagejpeg($im, "images/ciel2_copie.jpg");

imagedestroy($im);

?>
```

Bien entendu, il est possible de créer l'image du fichier généré dans un certain type, et de la sauvegarder sur le disque dans un autre.

3.2 Effectuer des transformations d'images

Vous voulez pouvoir redimensionner une image ? La faire tourner ? Créer une mosaïque ? Voici comment procéder en toute simplicité.

Redimensionner une image : création de vignettes

Nous allons réaliser ensemble un script PHP qui réalise une copie d'une image en réduisant sa taille. Cela peut être très utile si l'on veut programmer une galerie avec un système de vignettes.

Commençons donc par créer un nouveau fichier (dans le répertoire *chap03*) que l'on va nommer *gd_vignette.php* et qui va contenir le code suivant :

gd_vignette.php

```
<?php
function makeVignette($grandeImage, $vignette,
                      $vignetteLargeurMax, $vignetteHauteurMax){

}

makeVignette("images/ciel2.jpg", "vignettes/ciel2.jpg", 50, 50);
?>
```

Comme vous pouvez le voir, nous allons programmer cela sous la forme d'une fonction qui sera réutilisable à tout moment. Cette fonction dispose de quatre arguments : le premier indique là où se trouve l'image originale, et le deuxième où sauver notre vignette. Les troisième et quatrième arguments fixent la largeur et la hauteur maximales de la vignette. Dans notre exemple, les images originales seront stockées dans un répertoire *images*, et les vignettes dans un répertoire *vignettes*.

Nous pouvons d'ores et déjà récupérer les dimensions de l'image originale et créer l'objet image :

gd_vignette.php

```
<?php
function makeVignette($grandeImage, $vignette,
                      $vignetteLargeurMax, $vignetteHauteurMax){

  $imOriginale = @imagecreatefromjpeg($grandeImage)
    or die("impossible de créer un flux d'image GD2.");

  $infoOriginale = getimagesize($grandeImage);

}

makeVignette("images/ciel2.jpg", "vignettes/ciel2.jpg", 50, 50);
?>
```

La fonction `getimagesize()` permet de récupérer des informations sur une image et de les stocker dans un tableau. L'indice 0 de ce tableau contient la largeur de l'image, et l'indice 1 contient la hauteur de l'image.

Ces informations vont nous permettre de déterminer précisément quelles vont être les dimensions de notre vignette, de manière à ne pas déformer l'image :

gd_vignette.php

```
<?php
function makeVignette($grandeImage,
                      $vignette,
                      $vignetteLargeurMax,
                      $vignetteHauteurMax){

$imOriginale = @imagecreatefromjpeg($grandeImage)
  or die("impossible de créer un flux d'image GD2.");

$infoOriginale = getimagesize($grandeImage);

$largeurVignette;
$hauteurVignette;

if($infoOriginale[0] >= $infoOriginale[1]){
  $largeurVignette = $vignetteLargeurMax;
  $hauteurVignette = ($vignetteLargeurMax/$infoOriginale[0])
                     *$infoOriginale[1];
} else {
  $hauteurVignette = $vignetteHauteurMax;
  $largeurVignette = ($vignetteHauteurMax/$infoOriginale[1])
                     *$infoOriginale[0];
}

$im = @imagecreatetruecolor($largeurVignette,
                            $hauteurVignette)
  or die("impossible de créer un flux d'image GD2.");

}

makeVignette("images/ciel2.jpg", "vignettes/ciel2.jpg", 50, 50);
?>
```

La fonction `imagecreatetruecolor()` permet de créer un objet image qui contient une image vide ayant le premier argument (`$largeurVignette`, dans notre exemple) pour largeur, et le second argument (`$hauteurVignette`, dans notre exemple) pour hauteur.

Il ne nous reste plus qu'à créer la vignette sur le disque :

gd_vignette.php

```
<?php
function makeVignette($grandeImage,
                      $vignette,
                      $vignetteLargeurMax,
                      $vignetteHauteurMax){

  $imOriginale = @imagecreatefromjpeg($grandeImage)
    or die("impossible de créer un flux d'image GD2 vignette.");

  $infoOriginale = getimagesize($grandeImage);

  $largeurVignette;
  $hauteurVignette;

  if($infoOriginale[0] >= $infoOriginale[1]){
    $largeurVignette = $vignetteLargeurMax;
    $hauteurVignette = ($vignetteLargeurMax/$infoOriginale[0])
                       *$infoOriginale[1];
  } else {
    $hauteurVignette = $vignetteHauteurMax;
    $largeurVignette = ($vignetteHauteurMax/$infoOriginale[1])
                       *$infoOriginale[0];
  }

  $imVignette = @imagecreatetruecolor ($largeurVignette,
                                       $hauteurVignette)
    or die("impossible de créer un flux d'image GD2.");

  if(!imagecopyresized($imVignette,
                       $imOriginale,
                       0, 0, 0, 0,
                       $largeurVignette,
                       $hauteurVignette,
                       $infoOriginale[0],
                       $infoOriginale[1])){
    return false;
  }

  imagejpeg($imVignette, $vignette);

  return true;
}
```

```
if(!makeVignette("images/ciel2.jpg",
                 "vignettes/ciel2.jpg",
                 50, 50)){
  echo "Erreur : la vignette n'a pas été créée" ;
} else {
  echo "la vignette a bien été créée" ;
}
?>
```

Notre fonction `makeVignette()` est à présent opérationnelle : la fonction `imagecopyresized()`redimensionne l'image contenue par l'objet image `$imOriginale` et la copie dans l'objet image `$imVignette`. Enfin, la fonction `imagejpeg()`sauvegarde notre vignette sur le disque.

Voici la définition de la fonction `imagecopyresized()` :

imagecopyresized()

Redimensionne l'image `src_image` et la copie dans l'image `dst_image`. Vous pouvez choisir de ne copier et/ou redimensionner qu'une partie de l'image, et la positionner n'importe où dans l'image de destination.

Syntaxe	`bool imagecopyresized (resource` ***dst_image***`, resource` ***src_image***`, int` ***dst_x***`, int` ***dst_y***`, int` ***src_x***`, int` ***src_y***`, int` ***dst_w***`, int` ***dst_h***`, int` ***src_w***`, int` ***src_h***`)`
dst_im	L'image de destination.
src_im	L'image source.
dst_x	Abscisse du coin supérieur gauche de l'image copiée par rapport à l'image de destination.
dst_y	Idem, mais avec l'ordonnée.
src_x	Désigne, dans l'image source, l'abscisse du point représentant le coin supérieur gauche de l'image à copier : nous ne sommes pas forcés de copier l'intégralité de l'image source.
src_y	Idem, mais avec l'ordonnée.
dst_w	La largeur de notre copie, sur son image de destination.
dst_h	La hauteur de notre copie, sur son image de destination.
src_w	La largeur du morceau d'image que nous souhaitons copier, sur l'image source.

src_h Idem, mais avec la hauteur.

Notre fonction `makeVignette()` est certes opérationnelle, mais elle est limitée. Nous ne pouvons l'appliquer que sur des images de type JPEG. Afin de passer outre, nous allons devoir déterminer quel est le type de l'image originale avant de créer l'objet image qui s'y rapporte. Puis, nous allons faire en sorte que le type de la vignette soit donné par son nom (argument `$vignette` de la fonction).

Il s'avère que la fonction `getimagesize()`, que nous utilisons déjà dans notre code, renvoie aussi le type de l'image dans le tableau, index 2, sous la forme d'un entier. Nous allons donc utiliser cette possibilité dans notre code :

gd_vignette.php

```
<?php
function makeVignette($grandeImage, $vignette, $vignetteLargeurMax,
                      $vignetteHauteurMax){


  $infoOriginale = getimagesize($grandeImage);

  $imageOriginale;

  switch($infoOriginale[2]){
    case 1 :
      $imOriginale = @imagecreatefromgif($grandeImage)
        or die("impossible de créer un flux d'image GD2 vignette.");
      break;
    case 2 :
      $imOriginale = @imagecreatefromjpeg($grandeImage)
        or die("impossible de créer un flux d'image GD2 vignette.");
      break;
    case 3 :
      $imOriginale = @imagecreatefrompng($grandeImage)
        or die("impossible de créer un flux d'image GD2 vignette.");
      break;
    case 15 :
      $imOriginale = @imagecreatefromwbmp($grandeImage)
        or die("impossible de créer un flux d'image GD2 vignette.");
      break;
    default :
      return false;
  }

  $largeurVignette;
  $hauteurVignette;
```

```
  if($infoOriginale[0] >= $infoOriginale[1]){
    $largeurVignette = $vignetteLargeurMax;
    $hauteurVignette = ($vignetteLargeurMax/$infoOriginale[0])
                       *$infoOriginale[1];
  } else {
    $hauteurVignette = $vignetteHauteurMax;
    $largeurVignette = ($vignetteHauteurMax/$infoOriginale[1])
                       *$infoOriginale[0];
  }

  $imVignette = @imagecreatetruecolor($largeurVignette,
  ➥ $hauteurVignette)
    or die("impossible de créer un flux d'image GD2.");

  if(!imagecopyresized($imVignette, $imOriginale, 0, 0, 0, 0,
                       $largeurVignette, $hauteurVignette,
                       $infoOriginale[0], $infoOriginale[1])){
    return false;
  }

  switch(strrchr($vignette, ".")){
    case ".jpg" :
      imagejpeg($imVignette, $vignette);
      break;
    case ".jpeg" :
      imagejpeg($imVignette, $vignette);
      break;
    case ".gif" :
      imagegif($imVignette, $vignette);
      break;
    case ".png" :
      imagepng($imVignette, $vignette);
      break;
    case ".wbmp" :
      imagewbmp($imVignette, $vignette);
      break;
    default :
      return false;
  }

  return true;
}

if(!makeVignette("images/ciel2.jpg",
                 "vignettes/ciel2.jpg",
                 50, 50)){
```

```
  echo "Erreur : la vignette n'a pas été créée";
} else {
  echo "la vignette a bien été créée<br/>";
  echo "<img alt='grosse image' src='images/ciel2.jpg' />";
  echo "<img alt='petite image' src='vignettes/ciel2.jpg' />";
}
?>
```

► Fig. 3.2 : L'image d'origine et, juste à côté, son alter ego "vignette"

Voici les différentes valeurs que peut prendre l'index 2 du tableau renvoyé par la fonction `getimagesize()`:

- 1 = GIF
- 2 = JPEG
- 3 = PNG
- 4 = SWF
- 5 = PSD
- 6 = BMP
- 7 = TIFF (*Intel byte order*)

- 8 = TIFF (*Motorola byte order*)
- 9 = JPC
- 10 = JP2
- 11 = JPX
- 12 = JB2
- 13 = SWC
- 14 = IFF
- 15 = WBMP
- 16 = XBM

Écrire et superposer une image sur une autre

Vous avez besoin d'incruster un logo sur une image, puis d'y écrire quelque chose, tout cela grâce à PHP et GD2 ? Rien de plus simple. Voyons d'abord comment incruster un logo (ou quoi que ce soit) sur une image.

Nous allons pour ce faire créer une nouvelle fonction. Toujours dans le répertoire *chap03*, créez un fichier *gd_incrustation.php*, ainsi qu'un répertoire *incrustations*.

Ouvrez *gd_incrustation.php* avec votre éditeur favori et entrez le code suivant :

gd_incrustation.php

```
<?php
function incrusteImage($image,
                       $imageFini,
                       $logoLargeurMax,
                       $logoHauteurMax){

}
?>
```

Cette fonction prend en compte quatre arguments : le premier indiquera l'adresse de l'image originale, le deuxième l'endroit où nous voulons stocker l'image une fois traitée, le quatrième et le cinquième indiqueront la taille maximale du logo sur l'image. Comme nous considérons qu'il n'y a qu'un seul logo, l'adresse de celui-ci sera codé directement dans la fonction.

Commençons par créer les images dont nous aurons besoin :

```
$infoImage = getimagesize($image);
```

```
$imOriginale = @imagecreatefromjpeg($image)
  or die("impossible de créer un flux d'image GD2");

// utilisez l'image de votre choix :
$adresseLogo = "images/redfish.gif";

$imLogo = @imagecreatefromgif($adresseLogo)
  or die("impossible de créer un flux d'image GD2.");

$infoLogo = getimagesize($adresseLogo);
```

Vous remarquerez que nous récupérons des informations sur ces images grâce à la fonction `getimagesize()`. Cela nous permet, dans un premier temps, de gérer le redimensionnement du logo grâce au petit algorithme que nous avions vu lorsque nous réduisions des images en vignettes. Ajoutons donc le code suivant à notre fonction :

```
$largeurLogo;
$hauteurLogo;

if($infoLogo[0] >= $infoLogo[1]){
  $largeurLogo = $logoLargeurMax;
  $hauteurLogo = ($logoLargeurMax/$infoLogo[0])*$infoLogo[1];
} else {
  $hauteurLogo = $logoHauteurMax;
  $largeurLogo = ($logoHauteurMax/$infoLogo[1])*$infoLogo[0];
}
```

Il ne nous reste plus qu'à réaliser l'incrustation en elle-même :

```
if(!imagecopyresized($imOriginale,
                     $imLogo,
                     10, 10, 0, 0,
                     $largeurLogo,
                     $hauteurLogo,
                     $infoLogo[0],
                     $infoLogo[1])){
  return false;
}

imagegif($imOriginale, $imageFini);

return true;
```

Nous utilisons encore une fois la fonction `imagecopyresized()`. Vu le nombre d'arguments de cette fonction, il est possible d'opérer toutes sortes d'agrandissements et de copies.

Nous souhaitons maintenant écrire une petite phrase (par exemple l'adresse d'un site web) sur notre image, en même temps que nous y incrustons le logo. Nous allons donc continuer à ajouter du code dans notre fonction, après l'instruction pour l'incrustation et avant l'instruction d'écriture sur le disque (fonction `imagegif()`) :

```
// la couleur du texte :
$couleur = imagecolorallocate($imOriginale,
                              220, 30, 0);
// la phrase à écrire :
$phrase = "Une magnifique petite phrase !";
// la position en x de la phrase écrite :
// la position en Y de la phrase écrite :
$positionX = 10;
$positionY = $infoImage[1]-20;
$tailleDuTexte = 5 ;
imagestring($imOriginale,
            $tailleDuTexte,
            $positionX,
            $positionY,
            $phrase,
            $couleur);
```

La fonction `imagecolorallocate()` nous permet de définir la couleur que nous utiliserons pour notre phrase. Les trois derniers arguments de cette fonction représentent les trois composantes (rouge, vert et bleu) de la couleur.

Enfin, la fonction `imagestring()` écrit notre phrase sur notre image. L'argument `$tailleDuTexte` doit être compris entre 1 et 5.

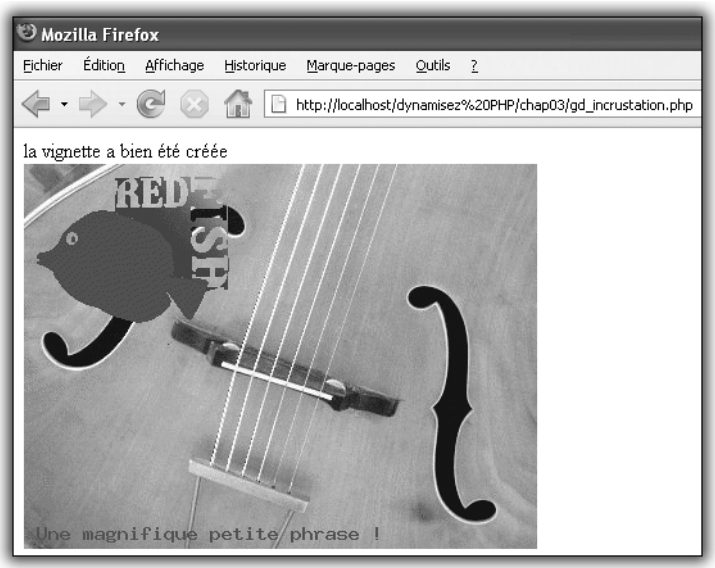
Voici à présent le code complet du fichier *gd_incrustation.php* :

gd_incrustation.php

```
<?php
function incrusteImage($image,
                       $imageFini,
                       $logoLargeurMax,
                       $logoHauteurMax){

  $infoImage = getimagesize($image);

  $imOriginale = @imagecreatefromjpeg($image)
    or die("impossible de créer un flux
            d'image GD2 vignette. jpg");
```

```
  $adresseLogo = "images/redfish.gif";

  $imLogo = @imagecreatefromgif($adresseLogo)
    or die("impossible de créer un flux
            d'image GD2.");

  $infoLogo = getimagesize($adresseLogo);

  $largeurLogo;
  $hauteurLogo;

  if($infoLogo[0] >= $infoLogo[1]){
    $largeurLogo = $logoLargeurMax;
    $hauteurLogo = ($logoLargeurMax/$infoLogo[0])
                   *$infoLogo[1];
  } else {
    $hauteurLogo = $logoHauteurMax;
    $largeurLogo = ($logoHauteurMax/$infoLogo[1])
                   *$infoLogo[0];
  }

  if(!imagecopyresized($imOriginale, $imLogo, 10, 10, 0, 0,
                       $largeurLogo, $hauteurLogo, $infoLogo[0],
                       $infoLogo[1])){
    return false;
  }

  // la couleur du texte :
  $couleur = imagecolorallocate($imOriginale, 220, 30, 0);
  // la phrase à écrire :
  $phrase = "Une magnifique petite phrase !";
  // la position en x de la phrase écrite :
  $positionX = 10;
  // la position en Y de la phrase écrite :
  $positionY = $infoImage[1]-20;
  $tailleDuTexte = 5;
  imagestring($imOriginale, $tailleDuTexte, $positionX,
              $positionY, $phrase, $couleur);

  imagegif($imOriginale, $imageFini);

  return true;
}

if(!incrusteImage("images/guitare.jpg", "incrustations/guitare.gif",
                  150, 150)){
  echo "Erreur : la vignette n'a pas été créée";
```

```
} else {
  echo "la vignette a bien été créée</br>";
  echo "<img alt=incrustation' src= incrustations/guitare.gif' />";
}
?>
```



► Fig. 3.3 : Notre incrustation

Dessiner sur une image

GD2 est fourni avec tout ce qu'il faut pour dessiner. Imaginons que vous ayez besoin, par exemple, de générer un graphique sous forme d'image. Nous allons donc créer pas à pas un script PHP qui construira un histogramme représentant un nombre indéfini d'enregistrements.

Commencez par créer, dans votre répertoire *chap03*, un fichier appelé *graphique_data .php* :

graphique_data.php

```
<?php
$graphique_data = array(
  'titre' => "mon super graphique",
  'configuration' => array(
    'unite' => "cm",
    'maximum' => 100
```

```
  ),
  'enregistrements' => array(
    array(
      'nom' => "Un",
      'valeur' => 5
    ),
    array(
      'nom' => "Deux",
      'valeur' => 10
    ),
    array(
      'nom' => "Trois",
      'valeur' => 20
    ),
    array(
      'nom' => "Quatre",
      'valeur' => 35
    ),
    array(
      'nom' => "Cinq",
      'valeur' => 55
    ),
    array(
      'nom' => "Six",
      'valeur' => 80
    )
  )
);
?>
```

Il n'est composé que d'un tableau contenant les données du graphique. Nous pourrons ajouter ou enlever des enregistrements à notre gré.

Créez maintenant le fichier *graphique.php* (toujours dans le répertoire *chap03*) et remplissez-le ainsi :

graphique.php

```
<?php
header("Content-type: image/png");

// importe le tableau :
require_once('graphique_data.php');

$largeur = 800;
$hauteur = 600;
```

```
// information sur le repère :
$repere = array(
  'origine' => array(
    'x' => 50,
    'y' => $hauteur-75
  ),
  'maxY' => 5,
  'maxX' => $largeur-20
);
$repere['longueurAbscisse'] = $repere['origine']['y']
                              -$repere['maxY'];
$repere['longueurOrdonnee'] = $repere['maxX']
                              -$repere['origine']['x'];

// créer une image à palette :
$image = imagecreate($largeur, $hauteur);

// donner un fond à l'image :
$blanc = imagecolorallocate($image, 255, 255, 255);
imagefill($image, 0, 0, $blanc);

// dessiner le repère :
$grisfonce = imagecolorallocate($image, 20, 20, 20);
imagesetthickness($image, 2);
imageline(
  $image,
  $repere['origine']['x'],
  $repere['maxY'],
  $repere['origine']['x'],
  $repere['origine']['y'],
  $grisfonce
);
imageline(
  $image,
  $repere['origine']['x'],
  $repere['origine']['y'],
  $repere['maxX'],
  $repere['origine']['y'],
  $grisfonce
);

/*
--- futur code ---
*/
```

```
imagepng($image); // écrit l'image
imagedestroy($image); // détruit la "resource" image
?>
```

Vous pouvez d'ores et déjà admirer l'incroyable résultat en lançant ce fichier depuis votre navigateur préféré.

Remarquez bien que la première chose que nous faisons après avoir créé notre image avec la fonction `imagecreate()`, c'est de lui assigner une couleur de fond.

Nous dessinons ensuite les deux axes de notre repère orthonormé. Pour ce faire, nous commençons par définir la couleur que nous allons utiliser. Nous utilisons alors la fonction `imagesetthickness()` afin de donner une épaisseur de 2 pixels aux traits que nous allons tracer. Les deux lignes sont ensuite tracées grâce à `imageline()`.

Imageline()

Cette fonction vous permet de tracer une ligne sur une image.

Syntaxe	`bool imageline (resource` *`image`*`, int` *`x1`*`, int` *`y1`*`, int` *`x2`*`, int` *`y2`*`, int` *`color`*`)`
Image	L'image sur laquelle nous allons tracer le trait.
x1	L'abscisse du point de départ de notre ligne.
y1	L'ordonnée du point de départ de notre ligne.
x2	L'abscisse du point d'arrivée de notre ligne.
y2	L'ordonnée du point d'arrivée de notre ligne.

Nous calculons les coordonnées de ces lignes en prenant en compte la hauteur et la largeur de notre image. Cela nous permettra d'avoir des graphiques de tailles différentes simplement en changeant les variables `$largeur` et `$hauteur`.

Nous allons maintenant faire apparaître tout ce qui se doit d'être écrit sur notre histogramme :

```
//écrit le titre du graphique
$rouge = imagecolorallocate($image, 220, 30, 0);
imagestring($image,
            5, 10,
            $hauteur-20,
            $graphique_data['titre'],
            $rouge);
imagestring($image,
            5, 55, 0,
```

```
            $graphique_data['configuration']['unite'],
            $grisfonce);
imagestring($image,
            5, 40,
            $hauteur-75,
            "0",
            $grisfonce);
```

Enfin, nous allons faire apparaître les barres de notre histogramme :

```
//fait apparaître les barres de l'histogramme
// calcul de la largeur des barres :
$largeurBarre = $repere['longueurAbscisse']/
                ((count
                ($graphique_data['enregistrements'])
                *2
                )+1);

$bleu = imagecolorallocate($image, 0, 20, 230);
for($i = 0 ;
    $i < count($graphique_data['enregistrements']) ;
    $i++){
  // calcul des coordonnées des deux angles de la barre
  $barre = array(
    'x1' => $repere['origine']['x']+((($i*2)+1)*$largeurBarre),
    'y1' => $repere['origine']['y']-(($repere['longueurOrdonnee']
    ➥ *$graphique_data['enregistrements'][$i]['valeur'])/$graphique
    ➥ _data['configuration']['maximum']),
    'x2' => $repere['origine']['x']+((($i*2)+2)*$largeurBarre),
    'y2' => $repere['origine']['y']
  );
  // dessine le rectangle (la barre)
  imagefilledrectangle($image, $barre['x1'], $barre['y1'],
                       $barre['x2'], $barre['y2'], $bleu);
}
```

Certaines formules mathématiques utilisées ici peuvent paraître barbares mais, rassurez-vous, la plus compliquée n'est qu'un simple produit en croix.

Lorsque vous testez ce fichier, votre histogramme apparaît correctement dans votre image. Vous pouvez déjà changer les paramètres du fichier *graphique_data.php* pour vous amuser et, par la même occasion, apprécier la puissance de GD2.

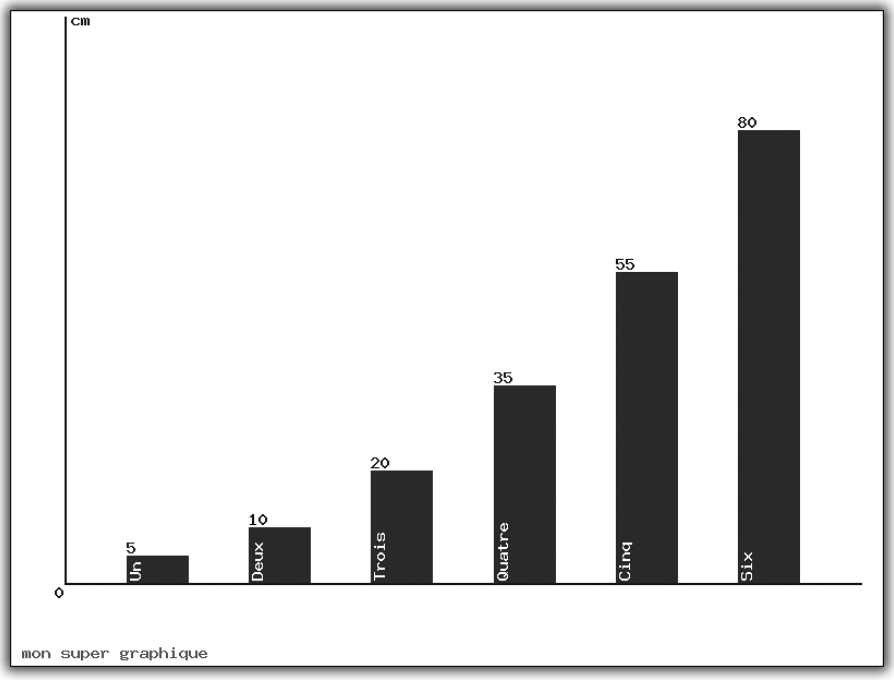
Nous allons maintenant modifier cette dernière portion de code afin d'intégrer sur notre image les dernières informations qui nous manquent. À savoir : le nom de chacun des enregistrements, ainsi que - dans un souci de précision - sa valeur.

Voici l'ensemble du code du fichier *graphique.php* contenant les dernières modifications. À vous de les trouver !

graphique.php

```
<?php
header("Content-type: image/png");

require_once('graphique_data.php'); // importe le tableau

$largeur = 800;
$hauteur = 600;

// information sur le repère
$repere = array(
  'origine' => array(
    'x' => 50,
    'y' => $hauteur-75
  ),
  'maxY' => 5,
  'maxX' => $largeur-20
);
$repere['longueurOrdonnee'] = $repere['origine']['y']
                              -$repere['maxY'];
$repere['longueurAbscisse'] = $repere['maxX']
                              -$repere['origine']['x'];

// créer une image à palette
$image = imagecreate($largeur, $hauteur);

// donner un fond à l'image
$blanc = imagecolorallocate($image, 255, 255, 255);
imagefill($image, 0, 0, $blanc);

// dessiner le repère
$grisfonce = imagecolorallocate($image, 20, 20, 20);
imagesetthickness($image, 2);
imageline($image, $repere['origine']['x'], $repere['maxY'],
          $repere['origine']['x'], $repere['origine']['y'],
          $grisfonce);
imageline($image, $repere['origine']['x'], $repere['origine']['y'],
          $repere['maxX'], $repere['origine']['y'], $grisfonce);

//écrit le titre du graphique
$rouge = imagecolorallocate($image, 220, 30, 0);
```

```
imagestring($image, 5, 10, $hauteur-20, $graphique_data['titre'],
            $rouge);
imagestring($image, 5, $repere['origine']['x']+5,
            0, $graphique_data['configuration']['unite'],
            $grisfonce);
imagestring($image, 5, 40, $repere['origine']['y'],
            "0", $grisfonce);

//fait apparaître les barres de l'histogramme
// calcul de la largeur des barres :
$largeurBarre = $repere['longueurAbscisse']/
                ((count($graphique_data['enregistrements'])
                *2)+1);

$bleu = imagecolorallocate($image, 0, 20, 230);
for($i = 0 ;
    $i<count($graphique_data['enregistrements']) ;
    $i++){
  // calcul des coordonnées des deux angles de la barre
  $barre = array(
    'x1' => $repere['origine']['x']+((($i*2)+1)*$largeurBarre),
    'y1' => $repere['origine']['y']
            -(($repere['longueurOrdonnee']
            *$graphique_data['enregistrements'][$i]['valeur'])
            /$graphique_data['configuration']['maximum']),
    'x2' => $repere['origine']['x']+((($i*2)+2)*$largeurBarre),
    'y2' => $repere['origine']['y']-2
  );

  // dessine le rectangle (la barre)
  imagefilledrectangle($image, $barre['x1'], $barre['y1'],
                       $barre['x2'], $barre['y2'], $bleu);
  // Ecrit le nom de l'enregistrement
  imagestringup($image, 5, $barre['x1'], $barre['y2']-2,
                $graphique_data['enregistrements'][$i]['nom'],
                $blanc);
  // Ecrit la valeur de l'enregistrement
  imagestring($image, 5, $barre['x1'], $barre['y1']-15,
              $graphique_data['enregistrements'][$i]['valeur'],
              $grisfonce);
}
imagepng($image); // écrit l'image
imagedestroy($image); // détruit la "resource" image
?>
```



► Fig. 3.4 : Et voilà le graphique

3.3 Utiliser ImageMagick et MagickWand

Même si GD2 est relativement simple d'installation et d'utilisation, il reste souvent bien limité. Quid des effets d'images traditionnels, comme le flou ? Quid des polices de caractères et de leur orientation ? L'ensemble ImageMagick et MagickWand répond bien à cette problématique, et parfois même, plus rapidement que GD2. En contrepartie, l'installation de cette extension est assez particulière, autant dire qu'elle est rarement présente sur les serveurs mutualisés.

Installation

ImageMagick est un programme de retouche d'images qu'il est nécessaire d'installer sur votre machine afin de pouvoir utiliser ses puissantes fonctionnalités. Pour le télécharger, rendez-vous à l'adresse **http://www.imagemagick.org/**, et choisissez la version 8 ou 16 bits suivant la puissance de votre ordinateur. Double-cliquez ensuite sur le fichier exécutable obtenu, et suivez la procédure d'installation.

Une fois l'installation terminée, ouvrez une fenêtre de commande (choisissez **Invite de commandes** dans le menu **Démarrer/Programmes/Accessoires**). Depuis **l'Invite de**

commandes, déplacez-vous dans un dossier contenant des images, puis entrez la commande (en remplaçant `logo.gif` par le nom d'une de vos images) : `Imdisplay logo.gif`.

L'interface de ImageMagick s'ouvre sur cette image. Le programme est correctement installé, vous pouvez fermer ImageMagick.

MagickWand est ce que l'on appelle un wrapper. Il permet d'utiliser directement les fonctions de ImageMagick en les appelant avec des fonctions PHP.

Comme l'immense majorité des extensions PHP, MagickWand se trouve sous forme de fichiers *.dll*. Pour les récupérer, allez à l'adresse **http://www.magickwand.org/download/php/windows/** et téléchargez les fichiers *php_magickwand_dyn.dll* et *php_magickwand_q16_st.dll*, ainsi que la documentation.

Copiez vos fichiers *.dll* dans le répertoire des extensions de PHP, puis éditez votre fichier *php.ini* afin de déclarer ces nouvelles extensions. Relancez Apache. Le tour est joué.

Nous allons maintenant écrire le fichier *magickwandtest.php*, que nous plaçons dans notre répertoire *chap03*. Ce fichier a pour but d'écrire une phrase sur une image et nous permettra de vérifier le bon fonctionnement de l'installation.

Voici le code de *magickwandtest.php* :

magickwandtest.php

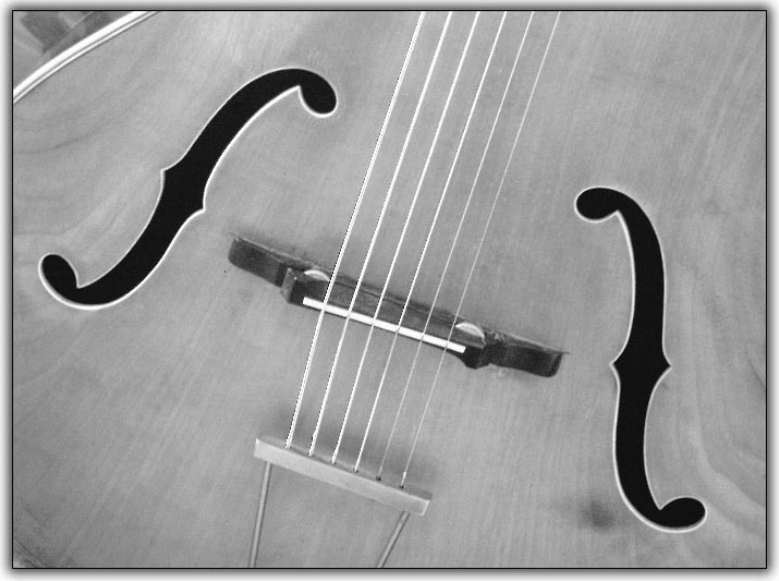
```
<?php
header("Content-type: image/jpeg");
// créer un objet MagickWand
$magick_wand=NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand,'images/guitare.jpg');

// créer un objet DrawingWand
$drawing_wand=NewDrawingWand();
// Choix d'une police de caractères/
DrawSetFont($drawing_wand,"impact.ttf");
// Réglage de la taille du texte/
DrawSetFontSize($drawing_wand,20);
// outil de centrage/
DrawSetGravity($drawing_wand,MW_CenterGravity);

//créer un objet PixelWand
$pixel_wand=NewPixelWand();
PixelSetColor($pixel_wand,"white"); // choix de la couleur
```

```
// choisit la couleur à utiliser pour remplir les objets, à partir de
➥ la couleur d'un objet PixelWand.
DrawSetFillColor($drawing_wand,$pixel_wand);

// Ecrit sur l'image
if (MagickAnnotateImage($magick_wand, $drawing_wand, 0, 0, 0,
                        "salut le monde !!!") != 0) {
  MagickEchoImageBlob( $magick_wand );
} else {
  echo MagickGetExceptionString($magick_wand);
}
?>
```

Avant de lancer ce fichier depuis votre navigateur favori, assurez-vous de cibler une image existante et un fichier de police de caractères également existant (vous pouvez copier un fichier de police de caractères dans le même répertoire que celui de votre fichier *magickwandtest.php*).

Maîtriser des effets pour les images

Explorer entièrement MagickWand demanderait un ouvrage complet sur le sujet : c'est pourquoi nous n'allons voir qu'une toute petite partie de ce programme, mais qui reste impressionnante, puisque nous allons traiter des filtres.



► Fig. 3.5 : L'image que nous utilisons pour illustrer cette section

Appliquer des effets

Nous allons commencer par simplement flouter une image. Pour cela, créez dans votre répertoire *chap03* un fichier intitulé *magick.php* :

magick.php

```
<?php
header("Content-type: image/jpeg");

// créer un objet MagickWand
$magick_wand=NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand,'images/guitare.jpg');

// Ecrit sur l'image
if (!MagickBlurImage($magick_wand, 20, 5)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickEchoImageBlob( $magick_wand );
?>
```

Si vous ouvrez *magick.php* depuis votre navigateur, vous trouverez votre image complètement floue. Regardons le code d'un peu plus près.

La première ligne que nous écrivons concerne la fonction `header()`qui, après avoir vu le fonctionnement de GD2, doit vous sembler évidente.

Tout de suite après, nous créons un objet `MagickWand`, grâce à la fonction `NewMagickWand()`. Cet objet est celui qui chargera l'image que nous souhaitons modifier, et c'est par lui que s'y appliqueront toutes les modifications.

La fonction `MagickReadImage()` nous permet de préciser l'adresse de l'image que nous souhaitons arranger à notre façon. Comme nous chargeons une image de type JPEG, `MagickWand` créera en sortie une image de type JPEG ; ce qui explique le `content-type` de la fonction `header()`.

Maintenant que notre image est en mémoire, nous allons la flouter grâce à la fonction `MagickBlurImage()`.

MagickBlurImage()

Cette fonction permet d'appliquer un effet flou à une image.

Syntaxe	`bool MagickBlurImage(MagickWand` *`mgck_wnd`*`, float` *`radius`*`, float` *`sigma`* `[, int` *`channel_type`*`])`
`mgck_wnd`	L'objet MagickWand, référence de l'image sur laquelle nous souhaitons appliquer l'effet.
`Radius`	Le rayon de notre effet flou.
`Sigma`	Le paramètre `sigma` de notre effet flou.
`channel_type`	Doit correspondre à l'une des constantes correspondant à un canal couleur, permettant de n'appliquer l'effet que sur le canal rouge, par exemple.

Le dernier argument de cette fonction permet de spécifier un canal couleur particulier sur lequel s'appliquera la transformation. Il doit prendre pour valeur la constante correspondant au canal couleur souhaité.

Modifiez votre code de la manière suivante :

```
if (!MagickBlurImage($magick_wand, 20, 5, MW_MagentaChannel)) {
  echo MagickGetExceptionString($magick_wand);
}
```

Maintenant, seul le canal magenta de votre image est flouté, lui donnant ainsi une touche très… fluo.

Voici les différentes constantes utilisables dans ce contexte :

- `MW_RedChannel`
- `MW_GreenChannel`
- `MW_BlueChannel`
- `MW_CyanChannel`
- `MW_MagentaChannel`
- `MW_YellowChannel`
- `MW_BlackChannel`
- `MW_AlphaChannel`
- `MW_OpacityChannel`

- MW_IndexChannel
- MW_AllChannels

MagickWand propose d'autre types de flous, via des fonctions très semblables à MagickBlurImage(). Les voici :

- bool MagickGaussianBlurImage(MagickWand mgck_wnd, float radius, float sigma [, int channel_type])
- bool MagickMotionBlurImage(MagickWand mgck_wnd, float radius, float sigma, float angle)
- bool MagickRadialBlurImage(MagickWand mgck_wnd, float angle)

Bien sûr, il existe dans MagickWand un certain nombre d'effets. En voici quelques-uns :

- bool MagickOilPaintImage(MagickWand mgck_wnd, float radius) : effet "peinture à l'huile" ;
- bool MagickRaiseImage(MagickWand mgck_wnd, float width, float height, int x, int y, bool raise) : ajoute un petit effet 3D à l'image, lui donnant l'apparence graphique d'un bouton standard ;
- bool MagickSwirlImage(MagickWand mgck_wnd, float degrees) : effet tourbillon ;
- bool MagickNegateImage(MagickWand mgck_wnd [, bool only_the_gray [, int channel_type]]) : donne le négatif de l'image ;
- Etc.

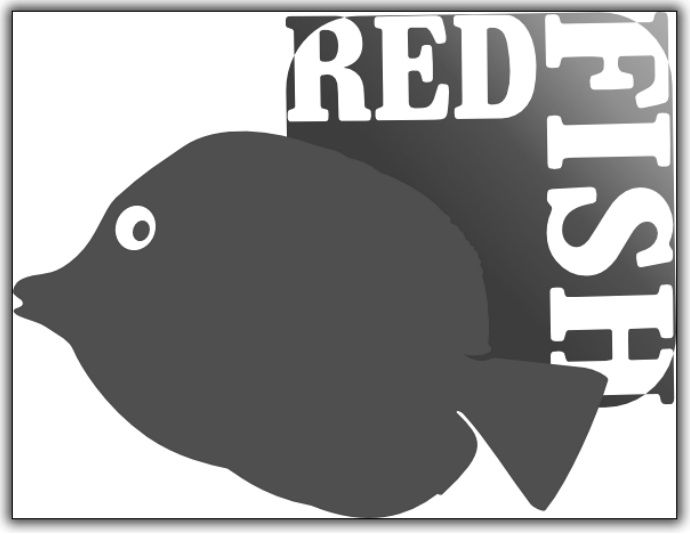
Vous voulez superposer plusieurs filtres ? Voici la marche à suivre :

```
// ouvre une image
MagickReadImage($magick_wand,'images/guitare.jpg');

// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}

// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickEchoImageBlob( $magick_wand );
```

► Fig. 3.6 : Voici à quoi ressemble notre guitare après un tel traitement

Superposer des images

Nous allons maintenant superposer une seconde image par-dessus la première.

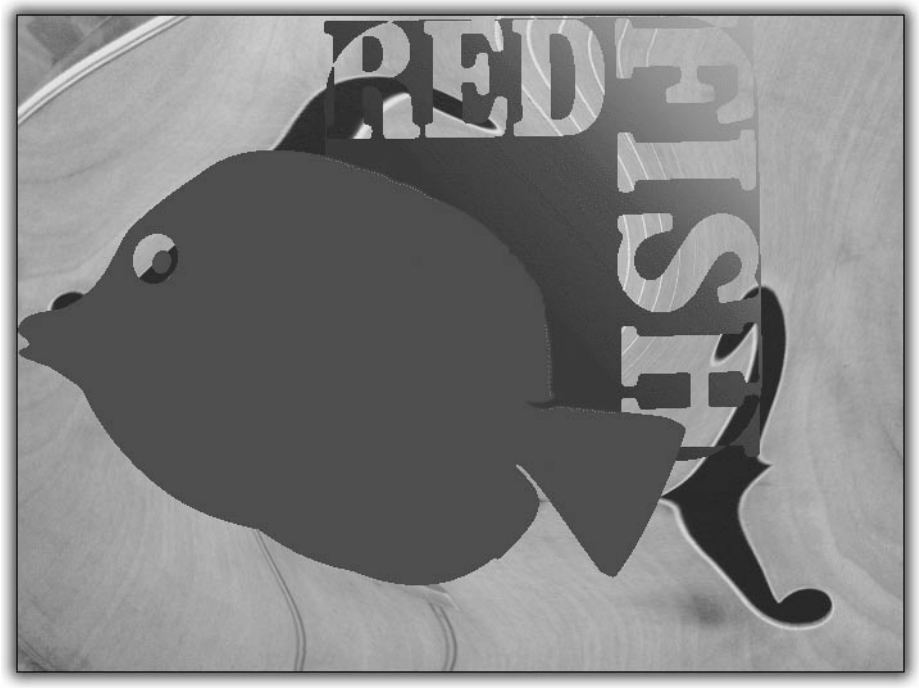


► Fig. 3.7 : Voici l'image que nous allons utiliser pour la superposition

magick.php

```
<?php
header("Content-type: image/jpeg");

// créer un objet MagickWand
$magick_wand=NewMagickWand();

// créer un objet MagickWand
$magick_wand2=NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand,'images/guitare.jpg');

// ouvre une seconde image
MagickReadImage($magick_wand2,'images/redfish.gif');

// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}

// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

// superposer l'image
if (!MagickAddImage($magick_wand, $magick_wand2)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickEchoImageBlob( MagickFlattenImages($magick_wand) );
?>
```

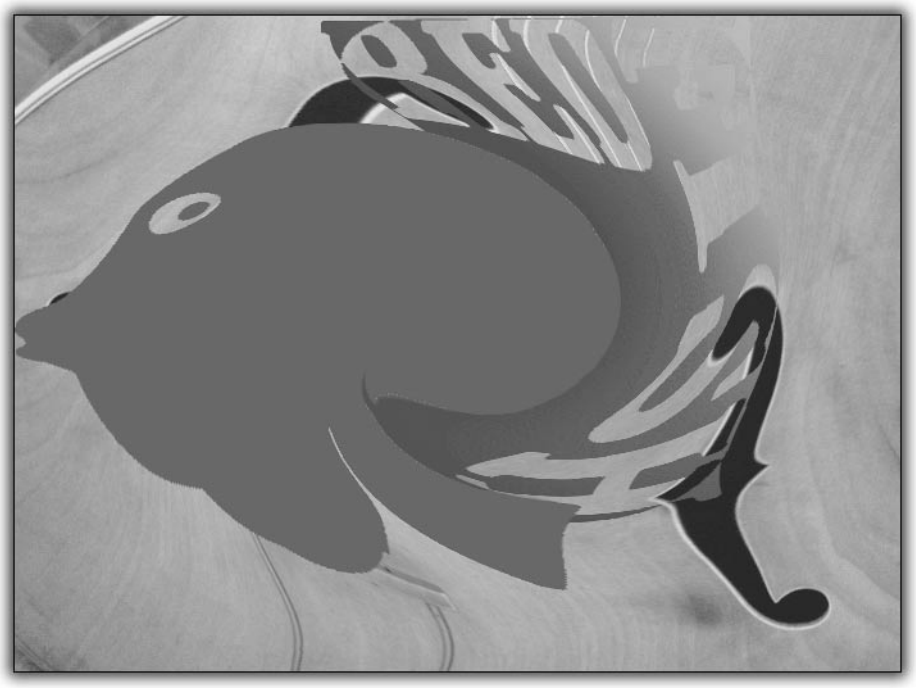
Il est nécessaire de créer un second objet `MagickWand`, afin de charger une seconde image. Des filtres peuvent être appliqués séparément sur chacune des images. La superposition des deux images se fait en deux temps. D'abord, nous les associons grâce à la fonction `MagickAddImage()`. Ce faisant, les deux images se fondent en une, mais sous forme de calques distincts. Chaque objet `MagickWand` peut encore subir des transformations indépendamment de l'autre. C'est la fonction `MagickFlattenImage()` qui va réellement superposer les deux images en une seule.



► Fig. 3.8 : Le résultat de la superposition

Si vous utilisez la fonction `MagickFlattenImage()`avant d'appliquer vos filtres (ou tout autre transformation), ils s'appliqueront sur les deux images :

```
// superposer l'image
if (!MagickAddImage($magick_wand, $magick_wand2)) {
  echo MagickGetExceptionString($magick_wand);
}

$magick_wand = MagickFlattenImages($magick_wand);

// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}

// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}
```



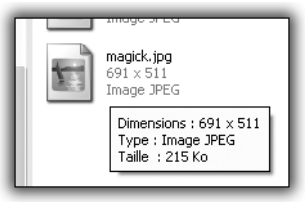
► Fig. 3.9 : Les filtres ont été appliqués sur les deux images.

Compresser et enregistrer une image

À ce stade, il ne nous reste plus qu'à enregistrer l'image sur le disque. Pour ce faire, modifiez votre code de la façon suivante :

```
// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickWriteImage($magick_wand, "images/magick.jpg");

MagickEchoImageBlob( $magick_wand );
```

La fonction `MagickWriteImage()` va créer le fichier *magick.jpg* à l'adresse passée en argument. Par contre, l'image qui a été enregistrée pèse assez lourd.

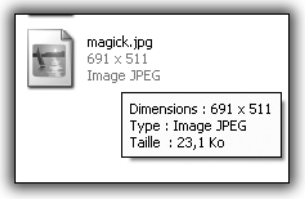


▶ Fig. 3.10 :
Le fichier créé pèse assez lourd

Nous allons donc rajouter une ligne de code pour profiter de la compression JPEG, juste avant la fonction `MagickWriteImage()` (car cela ne sert à rien de compresser alors que l'on a déjà enregistré l'image) :

```
MagickSetImageCompressionQuality($magick_wand, 50);
```

Le second argument de cette fonction représente le taux de compression. À 100, la qualité de votre image sera optimale. À 1, votre image sera d'une qualité déplorable (à moins qu'il ne s'agisse d'un parti-pris artistique), mais elle ne pèsera plus bien lourd.



▶ Fig. 3.11 :
Avec un taux de compression de 50, notre image s'est grandement allégée

Voici, pour terminer, le code complet de *magick.php* :

magick.php

```
<?php
header("Content-type: image/jpeg");

// créer un objet MagickWand
$magick_wand=NewMagickWand();

// créer un objet MagickWand
$magick_wand2=NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand,'images/guitare.jpg');

// ouvre une seconde image
MagickReadImage($magick_wand2,'images/redfish.gif');

// superposer l'image
```

```
if (!MagickAddImage($magick_wand, $magick_wand2)) {
  echo MagickGetExceptionString($magick_wand);
}

$magick_wand = MagickFlattenImages($magick_wand);

// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}

// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickSetImageCompressionQuality($magick_wand, 50);
MagickWriteImage($magick_wand, "images/magick.jpg");

MagickEchoImageBlob( $magick_wand );
?>
```

L'objet DrawingWand

En plus de pouvoir appliquer de puissants effets à nos images, MagickWand nous offre aussi de nombreux outils de dessins. Nous allons nous familiariser avec un premier exemple de l'utilisation de l'objet `DrawingWand`.

Dans *chap03*, créez le *fichier magick_draw.php*, et ajoutez le code suivant :

```
header("Content-type: image/jpeg");

// créer un objet MagickWand
$magick_wand = NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand, "images/guitare.jpg");

//créer un objet PixelWand
$pixel_wand=NewPixelWand();

// créer un objet DrawingWand
$drawing_wand = NewDrawingWand();
```

Nous créons toujours un objet `MagickWand`, et ouvrons toujours une image de la même manière que précédemment. Nous créons aussi deux nouveaux objets : un `PixelWand` et

un `DrawingWand`. Vous pouvez imaginer le `DrawingWand` comme étant un pinceau, un outil de dessin que nous pourrons décrire afin d'en faire varier la forme, et le `PixelWand` comme étant la peinture que nous allons utiliser avec le pinceau.

Complétez le code comme suit :

```
// dessiner 10 cercles dont la taille, l'emplacement, la couleur
// et la transparence seront déterminés par le hasard :

$couleur = Array(
  "white",
  "blue",
  "red",
  "green",
  "yellow");
for($i = 0 ; $i < 10 ; $i++){

  // choix de la couleur :
  PixelSetColor($pixel_wand,$couleur[rand(0, 4)]);

  // on lie la couleur (l'objet $pixel_wand)
  // au dessin (l'objet $drawing_wand) :
  DrawSetFillColor($drawing_wand,$pixel_wand);

  // on détermine la transparence (alpha) de la couleur
  // de remplissage :
  DrawSetFillAlpha($drawing_wand, rand(0, 33)/100);

  // on détermine les coordonnées du cercle :
  $ox = rand(10, MagickGetImageWidth($magick_wand)-10);
  $oy = rand(10, MagickGetImageHeight($magick_wand)-10);
  $px = rand($ox, MagickGetImageWidth($magick_wand)-10);
  $py = rand($oy, MagickGetImageHeight($magick_wand)-10);

  // on rajoute notre cercle dans la liste
  // "dessins à faire" de l'objet $drawing_wand :
  DrawCircle($drawing_wand, $ox, $oy, $px, $py);
}

// on dessine sur l'image la totalité des dessins
// contenus dans la liste des "dessins à faire"
// de l'objet $drawing wand :
MagickDrawImage($magick_wand, $drawing_wand);
```

Ce petit bout de programme va dessiner dix cercles de couleur, de taille et d'opacité variables par-dessus notre image. L'ordre des instructions est important.

À peine la boucle `for` commencée, nous appliquons une couleur à `$pixel_wand`, via la fonction `PixelSetColor()`. Dans notre exemple, la valeur de la couleur est choisie au hasard dans un tableau. Cette fonction accepte les noms de couleur standard sous forme de chaînes de caractères (comme dans notre exemple), ou les couleurs sous leur forme hexadécimale (exemple : `#FFFFFF`).

Nous lions ensuite `$pixel_wand` et `$drawing_wand` avec la fonction `DrawSetFillColor()` : c'est un peu comme tremper notre pinceau dans un seau de peinture.

La fonction `DrawSetFillAlpha()` permet de donner de la "transparence" à notre pinceau. Le second paramètre de cette fonction est de type `float` et est contenu entre 0 (complètement transparent) et 1 (complètement opaque).

Une fois que nous avons déterminé la couleur et l'opacité de notre pinceau et de notre peinture, nous allons nous occuper de tracer un cercle. Pour cela, nous utilisons simplement la fonction `DrawCircle()`. Celle-ci a besoin de savoir quel pinceau va la dessiner (c'est le premier argument de cette fonction), mais elle demande aussi quatre autres arguments de type `float`. Il s'agit des coins supérieur gauche et inférieur droit du rectangle dans lequel notre cercle s'inscrira.

À cette étape, notre cercle n'est pas encore dessiné sur l'image. La fonction `DrawCircle()` n'a fait qu'ajouter notre cercle dans une liste des "trucs à dessiner". Ainsi, notre boucle `for` va répéter dix fois cette suite d'instructions et inscrire dix fois dans la liste des "trucs à dessiner" qu'il faut dessiner un cercle aux propriétés aléatoires.

Vient enfin la fonction qui va dessiner le contenu de la liste de l'objet `DrawingWand` : `MagickDrawImage()`.

Nous terminons alors notre script ainsi :

```
// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}

// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickSetImageCompressionQuality($magick_wand, 85);
MagickWriteImage($magick_wand, "images/magick.jpg");

MagickEchoImageBlob( $magick_wand );
```

De cette manière, les effets "tourbillon" et "négatif" s'appliquent aussi sur les cercles que nous venons de dessiner.

► Fig. 3.12 : Un des nombreux résultats possibles

Voici le code complet du fichier *magick_draw.php* :

magick_draw.php

```
<?php
header("Content-type: image/jpeg");

// créer un objet MagickWand
$magick_wand = NewMagickWand();

// ouvre une image
MagickReadImage($magick_wand, "images/guitare.jpg");

//créer un objet PixelWand
$pixel_wand=NewPixelWand();

// créer un objet DrawingWand
$drawing_wand = NewDrawingWand();
```

```
// dessiner 10 cercles dont la taille, l'emplacement, la couleur
// et la transparence seront déterminés par le hasard :

$couleur = Array(
  "white",
  "blue",
  "red",
  "green",
  "yellow");
for($i = 0 ; $i < 10 ; $i++){

  // choix de la couleur :
  PixelSetColor($pixel_wand,$couleur[rand(0, 4)]);

  // on lie la couleur (l'objet $pixel_wand)
  // au dessin (l'objet $drawing_wand) :
  DrawSetFillColor($drawing_wand,$pixel_wand);

  // on détermine la transparence (alpha) de la couleur
  // de remplissage :
  DrawSetFillAlpha($drawing_wand, rand(0, 33)/100);

  // on détermine les coordonnées du cercle :
  $ox = rand(10, MagickGetImageWidth($magick_wand)-10);
  $oy = rand(10, MagickGetImageHeight($magick_wand)-10);
  $px = rand($ox, MagickGetImageWidth($magick_wand)-10);
  $py = rand($oy, MagickGetImageHeight($magick_wand)-10);

  // on rajoute notre cercle dans la liste
  // "dessins à faire" de l'objet $drawing_wand :
  DrawCircle($drawing_wand, $ox, $oy, $px, $py);
}

// on dessine sur l'image la totalité des dessins
// contenus dans la liste des "dessins à faire"
// de l'objet $drawing wand :
MagickDrawImage($magick_wand, $drawing_wand);


// Joue sur le négatif
if (!MagickNegateImage($magick_wand, false, MW_YellowChannel)) {
  echo MagickGetExceptionString($magick_wand);
}
```

```
// Effet tourbillon
if (!MagickSwirlImage($magick_wand, -180)) {
  echo MagickGetExceptionString($magick_wand);
}

MagickSetImageCompressionQuality($magick_wand, 85);
MagickWriteImage($magick_wand, "images/magick.jpg");

MagickEchoImageBlob( $magick_wand );
?>
```

Créer des avatars

Imaginons que nous souhaitons, alors que nous développons une application communautaire, créer un système d'avatars. Dans le cadre d'un forum, ce peut être "la petite image que j'ai choisie et qui est apposée à côté de chacun de mes messages". Afin que chaque utilisateur dispose d'un avatar très personnalisé mais s'intégrant tout de même à une charte graphique, nous nous proposons de créer un petit script qui superpose différentes parties de visages les unes sur les autres.

▶ Fig. 3.13 :
Un exemple d'avatar tel que notre code nous permettra de le construire.

Nous allons à cet effet créer deux fichiers. L'un, *avatar_fonctions.php*, contiendra un ensemble de fonctions qui manipuleront des objets `DrawingWand`. L'autre, *avatar.php*, assemblera ces `DrawingWand` au sein d'une image.

Le principe général de notre script

Voyons, pour commencer, le fichier *avatar.php* :

avatar.php

```
<?php
header("Content-type: image/jpeg");
```

```
include 'avatar_fonctions.php';

// tableau des paramètres de l'avatar
$avatar = Array(
  'couleur_peau' => "#d4c7a3",
  'couleur_contour' => "#000000",
  'opacite_contour' => 0.5,
  'epaisseur_contour' => 3,
);

// créer un objet MagickWand
$magick_wand = NewMagickWand();

// ouvre le fond blanc
MagickReadImage($magick_wand, "images/blanc.gif");

// Redimensionnement de notre image en 180*240
MagickScaleImage($magick_wand, 180, 240);

// utilisation des fonctions contenues dans
// le fichier avatar_fonctions.php :
/*
 *
 */

// un petit effet flou, pour le plaisir :
MagickGaussianBlurImage($magick_wand, 2, 1, MW_RedChannel);

MagickEchoImageBlob( $magick_wand );
?>
```

Il y a, dans ce fichier, le tableau `$avatar`, qui décrit notre avatar. Il nous permettra de changer à loisir les paramètres de notre image.

Ici, la fonction `MagickReadImage()` ouvre un fichier nommé *blanc.gif*. Il s'agit d'une petite image de 5*5 pixels, toute blanche. C'est elle qui servira de fond au visage que nous allons dessiner. La fonction `MagickScaleImage()` sert à redimensionner notre image à la taille que nous souhaitons.

Le reste du code est, somme toute, classique.

Le second fichier, *avatar_fonctions.php*, va contenir des fonctions manipulant des objets `DrawingWand`. Voici la première fonction que nous y écrirons, et qui aura pour but de dessiner un cercle (représentant la tête de notre avatar) :

avatar_fonctions.php

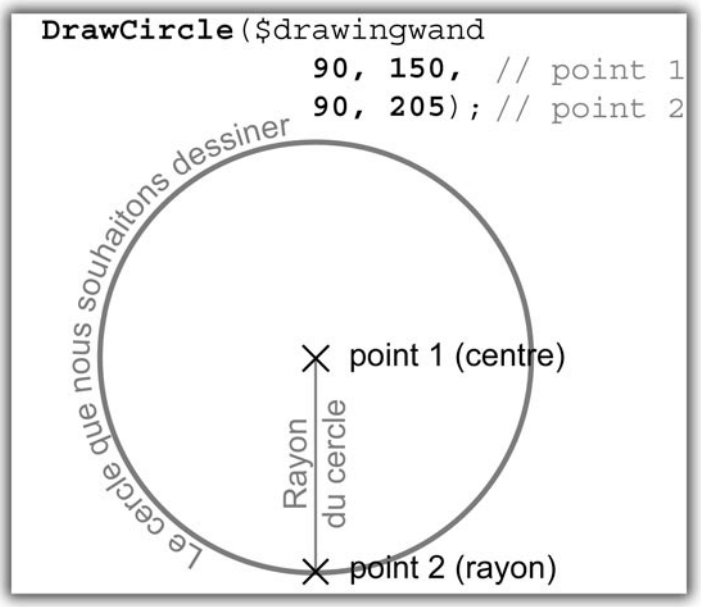
```
<?php
function dessineTete($conf){

  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_peau = NewPixelWand();
  $pxl_contour = NewPixelWand();

  PixelSetColor($pxl_peau, $conf['couleur_peau']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  DrawSetFillColor($drawing_wand,$pxl_peau);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);

  DrawCircle($drawing_wand,
             90, 150,     //centre
             90, 205);    //rayon

  return $drawing_wand;
}
?>
```

Notre fonction, `dessineTete()`, prend un argument en paramètre : `$conf`. Il s'agit en fait du tableau `$avatar` décrit dans le fichier *avatar.php*, que nous lui passerons en paramètre lors de son appel. Ce tableau est utilisé pour récupérer des paramètres concernant la tête : la couleur de la peau, l'épaisseur du contour, etc.

À propos de l'épaisseur du contour, il nous semble pertinent de signaler que chaque objet que nous allons dessiner dispose d'un fond et d'un contour, que vous pouvez faire apparaître ou non en y attachant une couleur. Ainsi, dans notre exemple, nous nous assurons d'avoir un fond coloré en appelant la fonction `DrawSetFillColor()` ; et un contour visible en appelant la fonction `DrawSetStrokeColor()`.

Nous utilisons aussi la fonction `DrawSetStrokeAlpha()` : il s'agit de modifier la transparence de notre contour (*stroke,* en anglais). Il existe une fonction similaire, `DrawSetFillAlpha()`, qui permet de modifier la transparence de votre couleur de fond, de remplissage (*fill,* en anglais). En fait, beaucoup de fonctions dont le nom est composé avec le mot "*stroke*" ont une sœur jumelle, composée avec le mot "*fill*", et vice versa.

Une fois que nous avons configuré nos "peintures" (les PiwelWand) et notre "pinceau" (le DrawingWand), nous pouvons dessiner le cercle grâce à la fonction `DrawCircle()`.



► Fig. 3.14 : La fonction DrawCircle()

Le premier paramètre de cette fonction est, vous l'aurez deviné, notre objet `DrawingWand`. Les quatre paramètres suivants décrivent deux points se situant dans le système de coordonnées de notre image (où l'origine se trouve dans le coin supérieur gauche). Le premier point (dans notre exemple : x = 90 et y = 150) représente le centre de notre cercle. Le second (dans notre exemple : x = 90 et y = 205) représente un point sur le cercle, c'est-à-dire que la droite formée par nos deux points est en fait le rayon du cercle que nous voulons tracer.

Pour finir, notre fonction `dessineTete()` renvoie notre objet `DrawingWand`. Nous pouvons maintenant l'utiliser dans notre fichier *avatar.php*. Modifiez donc son code comme suit :

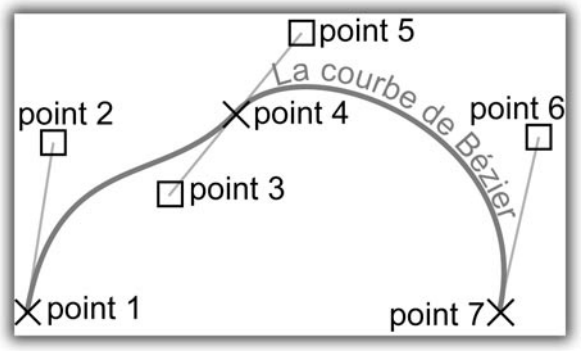
```
// utilisation des fonctions contenues dans
// le fichier avatar_fonctions.php :
MagickDrawImage($magick_wand, dessineTete($avatar));
/*
 *
 */
```

Si vous lancez maintenant *avatar.php*, vous pourrez admirer un incroyable rond au bas de votre image.

Nous allons donc continuer à créer des fonctions qui se chargeront de dessiner les yeux, le nez, la bouche, les oreilles (ou plutôt, "l'oreille") et les cheveux. Mais avant cela, savez-vous ce qu'est une courbe de Bézier ?

Les courbes de Bézier

Les courbes de Bézier, comme leur nom l'indique, sont… des courbes. Comme vous vous en doutez, nous allons avoir besoin de tracer des courbes pour dessiner nos avatars, et les courbes de Bézier sont un outil idéal, mais sûrement un peu compliqué à la première approche. Rien de mieux qu'un bon dessin pour comprendre :



► Fig. 3.15 :
Une courbe de Bézier

Comme vous le voyez sur cette image, la courbe de notre exemple se compose de sept points. Néanmoins, ils ne suivent pas tous la courbe. En effet, seuls trois points (symbolisés par des croix) se trouvent sur la courbe. Les points symbolisés par des carrés sont "reliés" aux points en forme de croix. Ils indiquent la "direction" que doit prendre la courbe. Observez le cheminement de notre courbe entre le point 1 et le point 4 : la courbe part du point en direction du point 2, puis change de trajectoire pour aller vers le point 3 avant de passer sur le point 4. Bien sûr, plus le point de direction est loin de son point de passage, plus son impact sur la courbe sera grand. Terminons cette brève explication par une règle importante : il y a toujours deux points de direction entre deux points de passage.

Pour tracer une courbe de Bézier avec MagickWand, nous allons, vous vous en doutez, avoir besoin des coordonnées de chacun de ces points. Voici comment nous coderions la courbe de notre exemple :

```
DrawBezier($objet_drawing_wand, array(
           20, 140,     // point 1
           30, 60,      // point 2
           70, 70,      // point 3
           100, 40,     // point 4
```

```
            130, 10,     // point 5
            210, 60,     // point 6
            180, 160));  // point 7
```

⚠ La fonction DrawBezier ne prend que 2 arguments.

La liste de nos points doit être passée en paramètre à cette fonction sous la forme d'un tableau.

Maintenant que nous savons coder une courbe de Bézier, nous allons rajouter deux fonctions de dessin : l'une pour dessiner la bouche, l'autre pour dessiner l'oreille. Rajoutez le code suivant dans *avatar_fonctions.php* :

```
function dessineOreille($conf){
  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_peau = NewPixelWand();
  $pxl_contour = NewPixelWand();

  PixelSetColor($pxl_peau, $conf['couleur_peau']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  DrawSetFillColor($drawing_wand,$pxl_peau);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);

  DrawBezier($drawing_wand, array(
             132, 150,     //point 1
             138, 132,     //point 2
             159, 144,     //point 3
             150, 165,     //point 4
             138, 186,     //point 5
             120, 177,     //point 6
             126, 168));   //point 7

  return $drawing_wand;
}

function dessineBouche($conf){
  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();
```

```
  //créer un PixelWand
  $pxl_contour = NewPixelWand();

  PixelSetColor($pxl_contour, $conf['couleur_contour']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  DrawSetFillAlpha($drawing_wand,0);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);

  DrawBezier($drawing_wand, array(
             42, 168,     //point 1
             54, 198,     //point 2
             111, 192,    //point 3
             114, 168));  //point 4

  return $drawing_wand;
}
```

Ce n'est pas très compliqué, n'est-ce pas ? Notez cependant que nous n'utilisons pas de couleur de fond pour la bouche : nous n'avons que le trait du contour… Alors que l'oreille est quand même coloriée avec une couleur, même si notre forme n'est pas fermée.

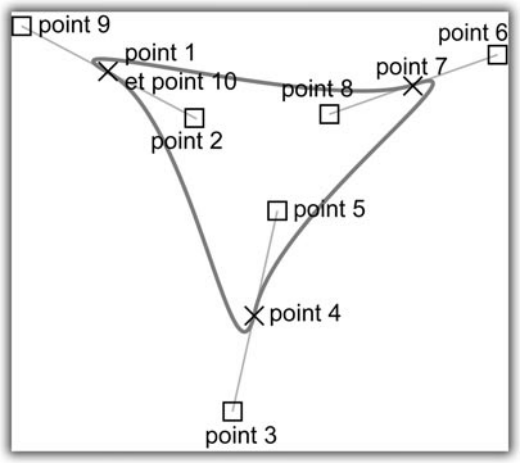
Complétons notre fichier *avatar.php* afin qu'il prenne en compte nos nouvelles fonctions et dessine effectivement une bouche et une oreille :

```
// utilisation des fonctions contenues dans
// le fichier avatar_fonctions.php :
MagickDrawImage($magick_wand, dessineTete($avatar));
MagickDrawImage($magick_wand, dessineOreille($avatar));
MagickDrawImage($magick_wand, dessineBouche($avatar));
```

Et voilà ! Notre magnifique rond se retrouve affublé d'une bouche et d'une oreille.

Fermer proprement ses courbes de Bézier

La fonction que nous allons créer maintenant dessinera les cheveux. Vu la complexité de la forme, nous allons utiliser une courbe de Bézier que nous allons refermer. Mais comment fermer une courbe de Bézier ? Rien de plus facile, observez ce schéma :



► Fig. 3.16 :
Une courbe de Bézier fermée

Le dernier point de la courbe est tout simplement le même que le premier. Voici comment nous pourrions coder cette courbe avec MagickWand :

```
DrawBezier($objet_drawing_wand, array(
           50, 60,     // point 1
           90, 100,    // point 2
           80, 190,    // point 3
           90, 160,    // point 4
           100, 130,   // point 5
           210, 70,    // point 6
           160, 80,    // point 7
           110, 90,    // point 8
           10, 20,     // point 9
           49, 60));   // point 10, bouclage
```

Comme vous le remarquez justement, le dernier point ne se trouve pas exactement aux mêmes coordonnées que le premier. En effet, si le premier et le dernier point se superposent parfaitement, MagickWand fera une petite erreur de rendu.

Revenons à nos cheveux. Comme nous avons besoin d'un paramètre supplémentaire, qui est la couleur des cheveux, nous allons le rajouter au tableau `$avatar` du fichier *avatar.php* :

```
// tableau des paramètres de l'avatar
$avatar = Array(
  'couleur_peau' => "#d4c7a3",
  'couleur_contour' => "#000000",
  'opacite_contour' => 0.5,
  'epaisseur_contour' => 3,
```

```
  'couleur_cheveux' => "#985d36"
);
```

Ensuite, rajoutez cette fonction dans votre fichier *avatar_fonctions.php* :

```
function dessineCheveux($conf){
  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_cheveux = NewPixelWand();
  $pxl_contour = NewPixelWand();

  PixelSetColor($pxl_cheveux, $conf['couleur_cheveux']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  DrawSetFillColor($drawing_wand,$pxl_cheveux);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);

  DrawBezier($drawing_wand, array(
            24, 123,     //point 1
            35, 68,      //point 2
            165, 68, 165, 85, 166, 121,  153, 177,   //points 3 à 6
            144, 183, 141, 180, 132, 174, 132, 174, //points 7 à 10
            138, 165, 138, 165, 132, 150, 132, 150,//points 11 à 14
            126, 162, 126, 162, 126, 162, 114, 165,//points 15 à 18
            111, 162, 117, 156, 120, 138, 154, 92, //points 19 à 22
            102, 138, 36, 130, 25, 123));    //points 23 à bouclage
  return $drawing_wand;
}
```

Et voilà ! La fonction `DrawBezier()` peut sembler effrayante, avec ces coordonnées de 24 points. Mais finalement, rien de sorcier, n'est-ce pas ?

Complétons donc notre fichier *avatar.php* afin qu'il dessine aussi les cheveux :

```
// utilisation des fonctions contenues dans
// le fichier avatar_fonctions.php :
MagickDrawImage($magick_wand, dessineTete($avatar));
MagickDrawImage($magick_wand, dessineCheveux($avatar));
MagickDrawImage($magick_wand, dessineOreille($avatar));
MagickDrawImage($magick_wand, dessineBouche($avatar));
```

Notez que nous utilisons la fonction dessinant les cheveux avant celle dessinant l'oreille. Les deux formes se superposant, il faut bien que l'une apparaisse au-dessus de l'autre.

C'est donc l'ordre dans lequel nous appelons nos fonctions qui va déterminer quelle forme sera au-dessus.

Que nous manque-t-il, pour terminer notre avatar ? Les yeux et le nez. Mais vous avez déjà compris comment les dessiner. Nous allons corser un peu l'exercice en donnant la possibilité de paramétrer le rayon des yeux. De plus, comme nous avons choisi une pose de trois quarts pour le visage de notre avatar, le nez se trouvera, en termes de superposition de formes, entre les deux yeux. C'est-à-dire qu'il devra apparaître par-dessus l'œil droit de notre avatar, et par-dessous l'œil gauche. Voici comment nous nous y prenons pour réaliser une telle fonction :

```
function dessineYeuxEtNez($conf){

  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_oeil = NewPixelWand();
  $pxl_iris = NewPixelWand();
  $pxl_contour = NewPixelWand();
  $pxl_peau = NewPixelWand();

  PixelSetColor($pxl_oeil, $conf['couleur_oeil']);
  PixelSetColor($pxl_iris, $conf['couleur_iris']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);
  PixelSetColor($pxl_peau, $conf['couleur_peau']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  // l'oeil droit :
  DrawSetFillColor($drawing_wand,$pxl_oeil);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);
  DrawCircle($drawing_wand,
             60, 150,     //centre
             60, 150+$conf['rayon_oeil']);  //rayon

  // l'iris droit :
  DrawSetFillColor($drawing_wand,$pxl_iris);
  DrawCircle($drawing_wand,
             63, 150,            //centre
             60, 150+($conf['rayon_oeil']/4));  //rayon

  // le nez
  DrawSetFillColor($drawing_wand,$pxl_peau);
  DrawBezier($drawing_wand, array(
             75, 150, 63, 159, 75, 171,  78, 168));
  // l'oeil gauche :
```

```
  DrawSetFillColor($drawing_wand,$pxl_oeil);
  DrawCircle($drawing_wand,
             90, 150,          //centre
             90, 150+$conf['rayon_oeil']);  //rayon

  // l'iris gauche :
  DrawSetFillColor($drawing_wand,$pxl_iris);
  DrawCircle($drawing_wand,
             93, 150,          //centre
             93, 150+($conf['rayon_oeil']/4));  //rayon


  return $drawing_wand;
}
```

Naturellement, n'oubliez pas de rajouter les nouveaux paramètres (la couleur de l'œil, celle de l'iris et le rayon de l'œil) dans le tableau `$avatar` du fichier *avatar.php* :

```
$avatar = Array(
  'couleur_peau' => "#d4c7a3",
  'couleur_contour' => "#000000",
  'opacite_contour' => 0.5,
  'epaisseur_contour' => 3,
  'couleur_cheveux' => "#985d36",
  'couleur_oeil' => "#ffffff",
  'couleur_iris' => "#cccccc",
  'rayon_oeil' => 16,
);
```

Enfin, n'oublions pas, toujours dans le fichier *avatar.php*, d'utiliser notre nouvelle fonction :

```
// utilisation des fonctions contenues dans
// le fichier avatar_fonctions.php :
MagickDrawImage($magick_wand, dessineTete($avatar));
MagickDrawImage($magick_wand, dessineCheveux($avatar));
MagickDrawImage($magick_wand, dessineOreille($avatar));
MagickDrawImage($magick_wand, dessineYeuxEtNez($avatar));
MagickDrawImage($magick_wand, dessineBouche($avatar));
```

Programmer un second nez et un second visage

Nous pouvons déjà avoir de nombreux avatars différents, mais ils auront finalement toujours la même tête. Pour remédier à cela, nous allons programmer une seconde forme de nez ainsi qu'une seconde forme de visage de manière à ce que l'on puisse choisir la

forme de notre choix directement depuis le tableau `$avatar`. Commençons donc par lui ajouter quelques paramètres :

```
// tableau des paramètres de l'avatar
$avatar = Array(
  'couleur_peau' => "#d4c7a3",
  'couleur_contour' => "#000000",
  'opacite_contour' => 0.5,
  'epaisseur_contour' => 3,
  'couleur_cheveux' => "#985d36",
  'couleur_oeil' => "#ffffff",
  'couleur_iris' => "#cccccc",
  'rayon_oeil' => 16,
  'nez' => 1,
  'tete' => 1,
);
```

Modifions maintenant la fonction `DessineTete()` de notre *fichier avatar_fonctions.php* :

```
function dessineTete($conf){

  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_peau = NewPixelWand();
  $pxl_contour = NewPixelWand();

  PixelSetColor($pxl_peau, $conf['couleur_peau']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  DrawSetFillColor($drawing_wand,$pxl_peau);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);

  switch($conf['tete']){
    case 1:
      DrawBezier($drawing_wand, array(
                42, 129,36, 141,40, 220,   //points 1 à 3
                40, 220,70, 230,167, 194,  //points 4 à 6
                167, 150167, 12651, 108,   //points 7 à 9
                43, 129));    //bouclage
      break;
  default :
      DrawCircle($drawing_wand,
```

```
                90, 150,      //centre
                90, 205);     //rayon
      break;
  }

  return $drawing_wand;
}
```

Grâce à l'instruction `switch()`, le script saura s'il lui faut dessiner une courbe de Bézier ou un cercle. Faisons de même avec la fonction `dessineYeuxEtNez()` :

```
function dessineYeuxEtNez($conf){

  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_oeil = NewPixelWand();
  $pxl_iris = NewPixelWand();
  $pxl_contour = NewPixelWand();
  $pxl_peau = NewPixelWand();

  PixelSetColor($pxl_oeil, $conf['couleur_oeil']);
  PixelSetColor($pxl_iris, $conf['couleur_iris']);
  PixelSetColor($pxl_contour, $conf['couleur_contour']);
  PixelSetColor($pxl_peau, $conf['couleur_peau']);

  DrawSetStrokeWidth($drawing_wand, $conf['epaisseur_contour']);
  DrawSetStrokeAlpha($drawing_wand, $conf['opacite_contour']);

  // l'oeil droit :
  DrawSetFillColor($drawing_wand,$pxl_oeil);
  DrawSetStrokeColor($drawing_wand, $pxl_contour);
  DrawCircle($drawing_wand,
             60, 150,          //centre
             60, 150+$conf['rayon_oeil']);  //rayon

  // l'iris droit :
  DrawSetFillColor($drawing_wand,$pxl_iris);
  DrawCircle($drawing_wand,
             63, 150,          //centre
             60, 150+($conf['rayon_oeil']/4));  //rayon

  // le nez
  DrawSetFillColor($drawing_wand,$pxl_peau);
  switch($conf['nez']){
    case 1 :
```

```
      DrawBezier($drawing_wand, array(
                  75, 150,63, 159,75, 171,78, 168));
      break;
    default :
      DrawBezier($drawing_wand, array(
                  72, 150,72, 150,42, 156,42, 156,
                  39, 159,36, 174,45, 174,45, 174,
                  78, 168,78, 168,
      break;
  }
  // l'oeil gauche :
  DrawSetFillColor($drawing_wand,$pxl_oeil);
  DrawCircle($drawing_wand,
             90, 150,          //centre
             90, 150+$conf['rayon_oeil']);  //rayon

  // l'iris gauche :
  DrawSetFillColor($drawing_wand,$pxl_iris);
  DrawCircle($drawing_wand,
             93, 150,          //centre
             93, 150+($conf['rayon_oeil']/4));  //rayon

  return $drawing_wand;
}
```

Ainsi, vous pouvez créer à loisir autant de formes d'yeux, de nez, de bouche, etc.

Écrire un nom au-dessus de notre avatar

Pour en finir avec l'objet `DrawingWand`, nous allons voir comment écrire quelque chose sur une image, avec la police de votre choix.

Commençons donc par compléter notre tableau `$avatar` avec les paramètres dont nous avons besoin pour le texte :

```
$avatar = Array(
  'couleur_peau' => "#d4c7a3",
  'couleur_contour' => "#000000",
  'opacite_contour' => 0.5,
  'epaisseur_contour' => 3,
  'couleur_cheveux' => "#cccccc",
  'couleur_oeil' => "#ffffff",
  'couleur_iris' => "#50bb00",
  'rayon_oeil' => 10,
  'nez' => 2,
  'tete' => 1,
  'fond_epaisseur_contour' => 1,
```

```
  'fond_couleur' => "#aa0000",
  'fond_couleur_contour' => "#000000",
  'nom' => "Maurice",
  'taille' => 35,
  'angle' => 10,
  'font' => 'Comic.ttf'
);
```

En ce qui concerne les polices de caractères, n'oubliez pas de les copier dans le même répertoire que le fichier qui les utilise, ou d'en indiquer correctement le chemin.

Et voici, dans le fichier *avatar_fonctions.php*, le code de la fonction `DessineNom()` :

```
function dessineNom($conf){
  // créer un objet DrawingWand
  $drawing_wand = NewDrawingWand();

  //créer des PixelWand
  $pxl_fond_couleur = NewPixelWand();
  $pxl_fond_couleur_contour = NewPixelWand();

  PixelSetColor($pxl_fond_couleur,
                $conf['fond_couleur']);
  PixelSetColor($pxl_fond_couleur_contour,
                $conf['fond_couleur_contour']);

  DrawSetStrokeWidth($drawing_wand,
                     $conf['fond_epaisseur_contour']);

  DrawSetFillColor($drawing_wand,$pxl_fond_couleur);
  DrawSetStrokeColor($drawing_wand,
                     $pxl_fond_couleur_contour);

  // la taille du texte en pixels :
  DrawSetFontSize($drawing_wand, $conf['taille']);
  // choix de la font :
  DrawSetFont($drawing_wand, $conf['font']);
  // changer l'orientation du système de coordonnées :
  DrawRotate($drawing_wand, $conf['angle']);

  // Cet objet MagickWand ne nous servira à rien d'autre
  // qu'à l'utilisation des fonctions MagickGetStringWidth()
  // et MagickGetStringHeight()
  $magick_tmp = NewMagickWand();

  // La fonction MagickGetStringWidth() nous permet de savoir
  // quelle sera la largeur (en pixel) de la chaîne de caractères
```

```
  // contenue dans $conf['nom'] :
  $largeurTexte = MagickGetStringWidth($magick_tmp,
                    $drawing_wand, $conf['nom']);
  // Pareil, mais pour la hauteur de la chaîne de caractères :
$hauteurTexte = MagickGetStringheight($magick_tmp,
                   $drawing_wand, $conf['nom']);

  // Déterminer les coordonnées de
  // notre chaine de caractères :
  $posX = (180-$largeurTexte)/2;
  $posY = 10+$hauteurTexte;

  // Dessiner la chaîne de caractères :
  DrawAnnotation($drawing_wand,
                 $posX, $posY,
                 $conf['nom']);

  return $drawing_wand;
}
```

Pour terminer, voici quelques exemples d'avatars créés avec ce code :

► Fig. 3.17 : Maurice

► Fig. 3.18 : Hulk !

3.4 Check-list

Dans ce chapitre, vous avez pu aborder la gestion des images et vous avez pu découvrir qu'il est possible de générer des images à la volée. Nous avons vu :

- ✔ comment créer des images avec GD2 ;
- ✔ comment les redimensionner, les signer, les convertir ;
- ✔ comment faire des montages et des incrustations ;
- ✔ comment dessiner sur des images
- ✔ comment appliquer des effets sur vos images avec ImageMagick et MagickWand ;
- ✔ comment réaliser des dessins complexes.

4

Générer des fichiers PDF

La classe FPDF permet de générer un document PDF grâce à PHP. Ce document s'ouvrira dans le navigateur de l'utilisateur ou dans son lecteur PDF. Cette extension très pratique vous permettra de créer dynamiquement des documents de toute nature, que l'utilisateur pourra conserver et diffuser indépendamment.
FPDF est une classe libre, que vous pourrez utiliser dans vos projets commerciaux ou non.

4.1 Télécharger et utiliser FPDF

Vous pouvez télécharger FPDF sur le site **http://www.fpdf.org**. À l'heure où ces lignes sont écrites, nous en sommes à la version 1.53. Créez un répertoire *chap04*, puis un répertoire *fpdf* à l'intérieur. Et voilà, FPDF est installé.

Pour utiliser FPDF, il vous faudra, à chaque fichier l'utilisant, importer la classe FPDF :

```
require('fpdf/fpdf.php');
```

Créer un document

Tout au long de ce chapitre, nous allons travailler autour d'un seul petit programme, que nous ferons évoluer petit à petit. Ce petit programme aura pour but de générer un fichier PDF représentant un relevé de compte. Créez donc, dans votre répertoire *chap04*, un nouveau fichier que vous appellerez *fpdf_banque.php*, et complétez-le de la manière suivante :

fpdf_banque.php

```
<?php
require('fpdf/fpdf.php');

//Créer une instance de FPDF
$pdf=new FPDF();

//Ajouter une page
$pdf->AddPage();

//envoyer à la sortie standard
$pdf->Output();
?>
```

Si vous lancez tout de suite ce fichier depuis votre navigateur favori, vous vous trouverez face à un splendide document PDF, en mode Portrait et au format A4, entièrement vide. Vous vous en doutiez, ce n'est que le début du chapitre…

Vous pouvez remarquer que nous n'appelons pas la fonction `header()`, pour l'en-tête http. En effet, c'est FPDF qui s'en charge.

Nous commençons par importer le contenu du fichier *fpdf.php*, c'est-à-dire le fichier contenant la classe FPDF que nous allons utiliser.

La création d'un nouveau document

Un document se crée tout simplement par l'appel du constructeur `new FPDF()`. C'est par cet appel que nous définissons les dimensions de notre document, son unité de mesure de base, ou son orientation (Portrait ou Paysage). Le constructeur dispose de trois paramètres, facultatifs, qui vont vous permettre de créer le nouveau document de vos rêves :

```
FPDF([chaîne $orientation (, chaîne $unite [, mixe $dimension]])
```

Le premier paramètre du constructeur, `$orientation`, définit l'orientation de vos pages. Non renseigné, votre document prendra l'orientation Portrait par défaut. Cette chaîne de caractères peut prendre les valeurs suivantes :

- `'P'` : portrait ;
- `'L'` : paysage (*landscape*).

Le deuxième paramètre, `$unite`, définit l'unité de notre système de coordonnées. L'unité par défaut est le millimètre, mais vous pouvez aussi choisir une autre unité parmi les suivantes :

- `'pt'`: point (il faut 72 points alignés pour faire une ligne de 1 pouce, soit 2,54 cm) ;
- `'mm'` : millimètre ;
- `'cm'` : centimètre ;
- `'in'` : pouce.

Le dernier paramètre, `$dimension`, vous permet de spécifier les dimensions de votre document. Vous pouvez le renseigner avec un tableau contenant des nombres flottants représentant la largeur et la hauteur dudit document. Ces nombres seront interprétés avec l'unité que vous avez choisie. Vous pouvez aussi renseigner ce paramètre avec une des chaînes de caractères suivantes :

- `'A3'` ;
- `'A4'` ;
- `'A5'` ;
- `'Letter'` ;
- `'Legal'`.

Faire sortir son document

Regardons maintenant la fonction `Output()`. Sans aucun paramètre, elle se contente de renvoyer notre document sur la sortie standard c'est-à-dire, le plus souvent, le navigateur.

Cette fonction dispose de deux paramètres, nous permettant de donner un nom au fichier, puis de savoir si on l'enregistre localement ou si on l'envoie sur le navigateur de l'utilisateur, ou encore si on lui propose de télécharger le document.

Le premier paramètre est une chaîne de caractères donnant un nom de fichier à notre document. Modifiez votre code comme ceci :

```
$pdf->Output("fichier.pdf");
```

Si vous exécutez votre fichier, rien n'apparaîtra dans le navigateur. Le plug-in Acrobat Reader ne se lancera même pas. Par contre, votre document aura été enregistré sur votre disque, sous le nom *fichier.pdf*. Vous pouvez influer sur cette action en spécifiant le deuxième paramètre. Il s'agit d'une chaîne de caractères pouvant prendre les valeurs suivantes :

- `'I'` : envoie au navigateur. Si l'utilisateur décide d'enregistrer le document sur son disque, le nom proposé sera celui que vous aviez indiqué avec le premier paramètre.
- `'D'` : propose le fichier en téléchargement à l'utilisateur.
- `'F'` : enregistre le fichier localement, avec le nom proposé par le premier argument.
- `'S'` : renvoie le document sous la forme d'une chaîne de caractères.

Si aucun paramètre n'est renseigné, la fonction envoie le document au navigateur. Si seul le deuxième paramètre est omis, la fonction enregistre le fichier localement.

Pour continuer ce chapitre et faire en sorte que votre document apparaisse dans votre navigateur, il vous faut modifier à nouveau votre fichier en enlevant l'argument que vous venez d'ajouter à la fonction `Output()`.

Placer des images

À présent, il est temps de commencer à remplir notre document. Traditionnellement, nous vous ferions écrire une petite phrase, mais placer une image est tellement simple que c'est ce que nous allons faire. Juste après l'appel à la fonction `AddPage()` de votre fichier *fpdf_banque.php*, ajoutez la ligne suivante :

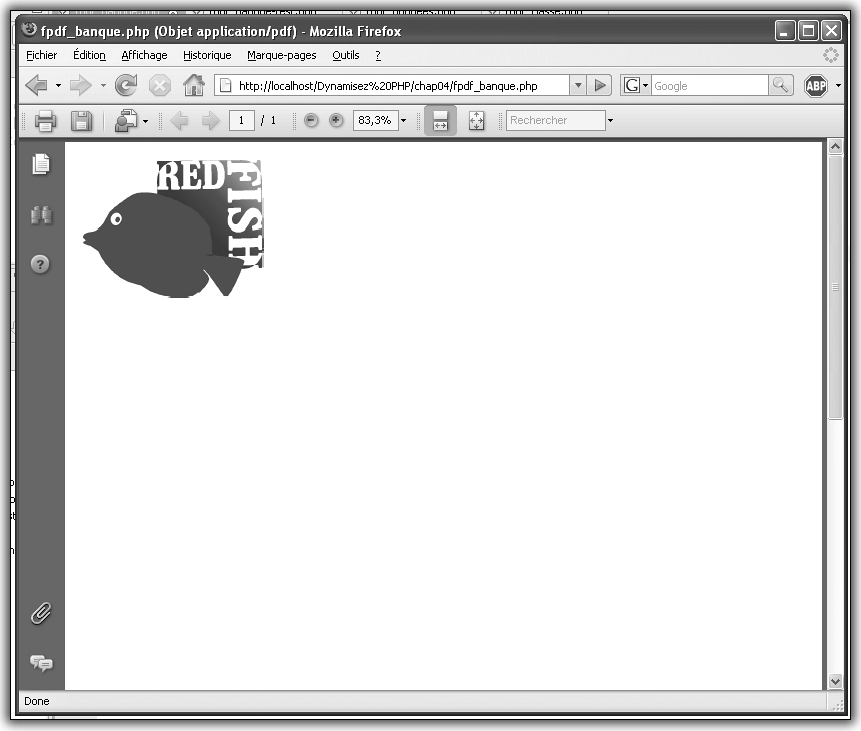
```
//Poser une image :
$pdf->Image('redfish.png', 5, 5, 50);
```

Voici la fonction `Image()`. Concrètement, elle va afficher l'image passée au premier argument dans notre document. Elle placera le coin supérieur gauche de notre image au point de coordonnées (5, 5), et dimensionnera sans la déformer (en respectant l'homothétie) notre image de manière à ce qu'elle fasse 50 unités (dans notre exemple, l'unité est le millimètre) de large. La hauteur de l'image sera adaptée.

Le système de coordonnées

Avec FPDF, l'origine de notre système de coordonnées se trouve dans le coin supérieur gauche de la page PDF.

Les types d'images que vous pouvez utiliser sont le JPEG et le PNG.



► Fig. 4.1 : Notre image

Placer du texte

Nous allons, avant de commencer à écrire du texte sur notre document PDF, créer un nouveau fichier appelé *fpdf_donnees.php*. Ce fichier contiendra toutes les données que nous inscrirons dans notre document PDF. Le fait qu'il soit séparé de *fpdf_banque.php* nous permettra de changer ces données plus facilement. Voici donc la première de nos données, à mettre dans le fichier *fpdf_donnees.php* :

```
<?php
$titre = "RELEVE DE COMPTE";
?>
```

N'oubliez pas de préciser, dans *fpdf_banque.php*, que nous allons utiliser des données provenant d'un autre fichier :

```
<?php
require('fpdf/fpdf.php');
include 'fpdf_donnees.php';
...
...
```

À présent, nous allons écrire le premier texte dans notre document. Toujours dans le fichier *fpdf_banque.php*, insérez les lignes suivantes juste après l'appel de l'image :

```
//Ecrire une chaîne
$pdf->SetFont('Arial','B',16);
$largeur = $pdf->GetStringWidth($titre);
$pdf->SetX(105-($largeur/2));
$pdf->Cell($largeur, 0, $titre);
```

Si vous lancez ce fichier maintenant, vous allez voir votre titre apparaître.

► Fig. 4.2 : Votre premier texte

Définir la police de caractères

Mais que se cache-t-il, dans ces quatre lignes de code ? Tout d'abord, nous définissons la police de caractères courante grâce à la fonction `SetFont()`. Le premier des trois arguments de cette fonction précise, d'une chaîne de caractères, quelle famille de police utiliser. Les familles disponibles sont les suivantes :

- `''` : chaîne vide : aucune police, ou bien conserve la police précédente.
- `'Courier'`
- `'Arial'`
- `'Helvetica'`
- `'Times'`
- `'Symbol'`
- `'ZapfDingbats'`

Le deuxième paramètre indique le style de la police. Il peut prendre les valeurs suivantes :

- `''` : chaîne vide : style normal.
- `'B'` : votre texte sera écrit en gras.
- `'I'` : votre texte sera écrit en italique.
- `'U'` : votre texte sera souligné.
- Vous pouvez aussi écrire `'BIU'`, votre texte sera en gras italique souligné. Ou alors `'UB'`, votre texte sera écrit en gras et souligné…

Enfin, le troisième paramètre indique la taille de la police, en point. Dans notre exemple, et ce jusqu'au prochain appel de la fonction `SetFont()`, nos textes seront écrits en Arial, gras, et auront une taille de 16 pt.

Positionner une chaîne : connaître sa largeur

Comme nous souhaitons centrer notre titre, et qu'il n'existe pas vraiment de fonction faite spécialement pour cela, il nous faut savoir quelle est la largeur de notre chaîne de caractères `$titre`, lorsqu'elle est écrite en Arial, gras, et avec une taille de 16 pt. C'est tout simplement ce que se propose de faire la fonction `GetStringWidth()`. Nous récupérons le résultat qu'elle nous renvoie dans la variable `$largeur`.

Nous allons utiliser la fonction `SetX()` pour définir le point de coordonnées courant, c'est-à-dire le point à partir duquel nous allons écrire, dessiner, ou placer une image. Ce n'est pas la peine d'utiliser la fonction `SetY()` (ou la fonction `SetXY()`), car le point de coordonnées courant est déjà verticalement défini en accord avec nos souhaits.

Écrire le texte

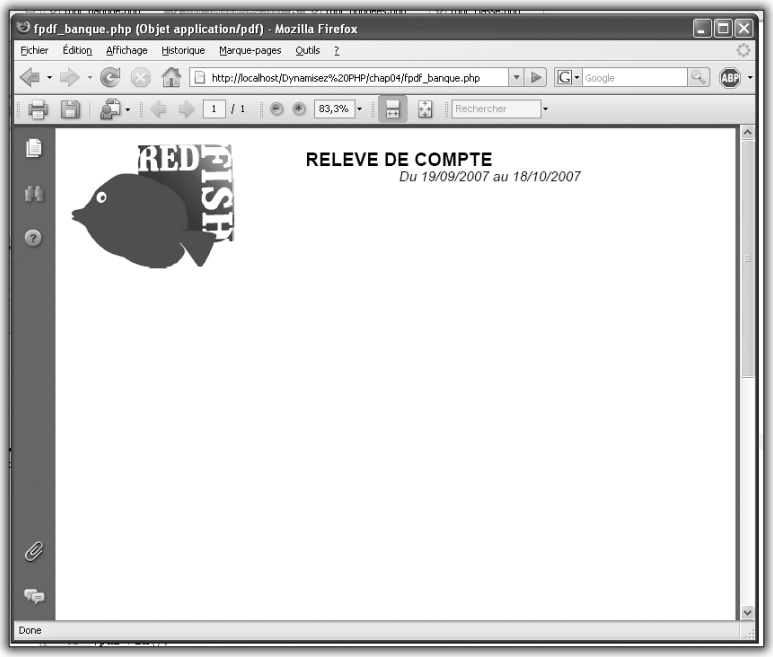
Nous avons déterminé notre police de caractères ainsi que l'emplacement de notre texte. Il ne nous reste plus qu'à l'écrire. À cet effet, nous utilisons la fonction `Cell()`. Voici comment nous l'avons mise en scène :

```
$pdf->Cell($largeur, 0, $titre);
```

En fait, notre texte va se placer à l'intérieur d'une cellule. Le premier argument de cette fonction définit donc la largeur de cette cellule, et le deuxième sa hauteur. Une hauteur de 0 mm n'est, pour le moment, pas encore un problème. Le troisième argument est, bien sûr, la chaîne de caractères à écrire sur le document.

Écrivons une deuxième chaîne. Commencez par créer, dans votre fichier *fpdf_donnes.php*, la variable `$dateReleve` contenant la chaîne de caractères que nous allons afficher. Ensuite, ajoutez ces lignes dans *fpdf_banque.php* :

```
//Ecrire une chaîne
$pdf->SetFont('Arial','I',12);
$largeur = $pdf->GetStringWidth($dateReleve);
$pdf->SetXY(105, 10);
$pdf->Cell($largeur, 10, $datereleve);
```



► Fig. 4.3 : Un deuxième texte ! Mais jusqu'où ira-t-on ?

Écrire des blocs de texte

Pour le moment, nous n'avons écrit du texte que sous forme de lignes, mais il est possible de le faire sous forme de paragraphes : les blocs de texte. Nous allons donc créer, dans *fpdf_donnees.php*, une variable `$adresse` contenant une chaîne de caractères suffisamment longue pour pouvoir constituer un paragraphe. Voici celle que nous utilisons dans notre exemple :

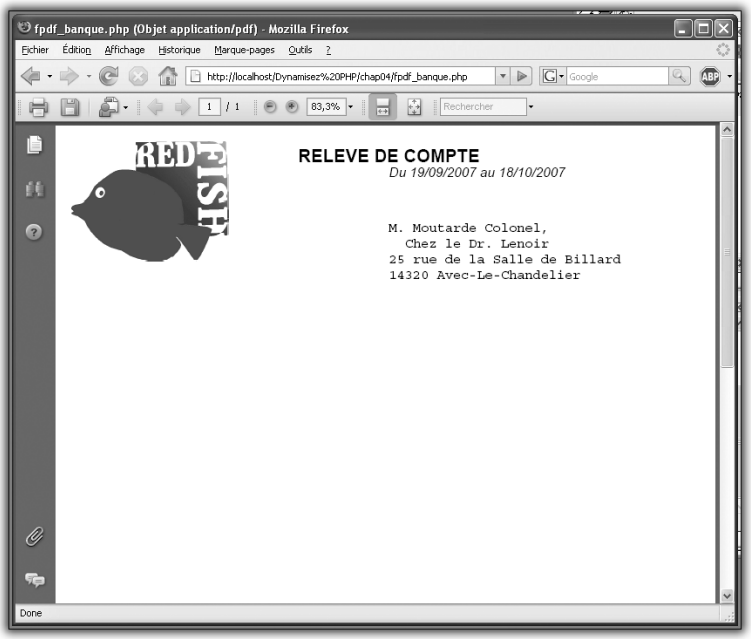
```
$adresse = "M. Moutarde Colonel,\n  Chez le Dr. Lenoir\n";
$adresse .= "25 rue de la Salle de Billard\n";
$adresse .= "14320 Avec-Le-Chandelier";
```

Vous pouvez noter la présence, dans notre chaîne, du caractère de retour chariot `'\n'`, afin de signifier les endroits où nous demandons explicitement à retourner à la ligne.

Ajoutons maintenant, dans *fpdf_banque.php*, de quoi constituer notre paragraphe :

```
//Ecrire un paragraphe
$pdf->SetXY(105, 30);
$pdf->SetFont('Courier','',12);
$pdf->Multicell(80, 5, $adresse);
```

Et voilà. Il n'y a rien de compliqué, si ce n'est que nous utilisons la fonction `Multicell()` au lieu de `Cell()`.



► Fig. 4.4 : Un bloc de texte

> **La hauteur de la cellule**
>
> Un bloc de texte est fait de plusieurs cellules à la suite. Il est donc important de spécifier une hauteur pour ces cellules. Si vous laissez le deuxième paramètre de la fonction `Multicell()` à `0`, toutes vos lignes s'écriront les unes par-dessus les autres. En somme, la hauteur de la cellule est ce qui sert de référence à FPDF pour savoir où placer une ligne (cellule) par rapport à la ligne précédente.

Centrer, justifier ou aligner sur la droite le contenu d'un bloc de texte

Eh oui, la fonction `Multicell()` est assez complète ! En plus, cela se fait simplement.

Comme nous avons besoin d'une nouvelle chaîne de caractères pour le nouveau paragraphe que nous allons faire, nous vous laissons le soin d'en créer une dans le fichier *fpdf_donnees.php* (dans une variable `$parCentre`). Voici celle que nous utilisons dans notre exemple :

```
$parCentre = "La banque \"RedFish\" - la banque à qui on ";
$parCentre .= "fait confiance - est très très fière, ";
$parCentre .= "de vous présenter son ";
$parCentre .= "nouveau service de relevés de comptes ";
$parCentre .= "en ligne !\n***\nUn service entièrement ";
$parCentre .= "dynamique qui nous permet de vous faire ";
$parCentre .= "des relevés de comptes quand vous le ";
$parCentre .= "souhaitez, et couvrant la période de ";
$parCentre .= "votre choix !\nTrop chouette, hein ?!!";
```

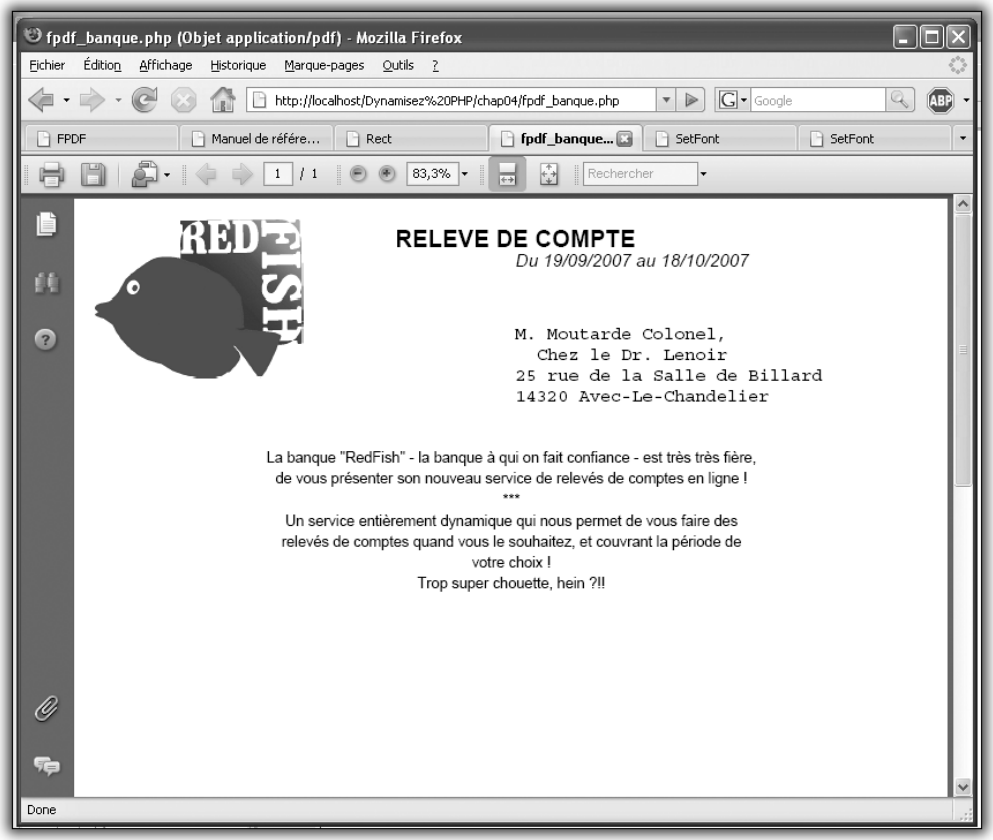
Voyons maintenant le code à ajouter à *fpdf_banque.php* :

```
//Ecrire un paragraphe centré
$pdf->SetXY(45, 60);
$pdf->SetFont('Arial','',10);
$pdf->Multicell(120, 5,
  utf8_decode($parCentre),
  0, 'C');
```

Comme vous pouvez le constater, nous avons rajouté deux arguments à la fonction `Multicell()`. Celui qui vaut `0` (l'avant-dernier argument) signifie que nous ne voulons pas de cadre à notre paragraphe (nous parlerons des cadres un peu plus loin). Le dernier paramètre est en fait une chaîne de caractères qui indique comment aligner le texte de votre paragraphe. Cette chaîne peut prendre les valeurs suivantes :

- `'L'` : votre paragraphe est aligné à gauche.
- `'R'` : votre paragraphe est aligné à droite.

- `'C'` : pour centrer le texte du paragraphe.
- `'J'` : pour justifier le texte du paragraphe.



► Fig. 4.5 : Un paragraphe où le texte est centré.

Des lignes et des colonnes : faire un tableau

Nous allons examiner un peu plus en détail ce que l'on peut faire avec une cellule. Pour le moment, notre point de coordonnées courant se trouve à la fin du paragraphe que nous venons d'écrire. Nous allons commencer par effectuer un retour chariot (sauter une ligne), de manière à ce que notre point de coordonnées courant se trouve une ligne en dessous de notre paragraphe, et le long de la marge de gauche (c'est-à-dire à 1 cm du bord gauche).

Les marges

Il existe une marge de 1 cm de côté, le long des quatre bords de notre page. Lorsque le point de coordonnées courant est déplacé automatiquement, il prend en compte la marge pour définir sa nouvelle position. Lorsque vous utilisez les fonctions `SetX()`, `SetY()` ou `SetXY()` pour changer le point de coordonnées courant, vous pouvez passer outre les marges (et c'est d'ailleurs ce que nous avons fait lorsque nous avons placé notre image).

Ajoutez la ligne suivante à la suite de *fpdf_banque.php* :

```
//sauter une ligne
$pdf->Ln(4);
```

La fonction `Ln()` effectue un retour chariot. Le paramètre détermine la hauteur de la ligne à sauter (ici, 4 mm. Attention : si l'unité que vous avez choisie est le point, notre ligne ne fera pas 4 mm de hauteur, mais 4 pt). Si on ne précise pas la hauteur de la ligne à sauter, celle-ci prendra automatiquement pour valeur la hauteur de la ligne précédente.

Afin de réaliser notre tableau, nous allons entrer de nouvelles données dans le fichier *fpdf_donnees.php* :

```
$ancienSolde = 100;

$operation = array(
  array(
    'date' => "19/09",
    'nature' => 'Cotisation mensuelle',
    'debit' => 10
  ),
  array(
    'date' => "20/09",
    'nature' => 'Carte bleue',
    'debit' => 32.5
  ),
  array(
    'date' => "20/09",
    'nature' => 'Commerce électronique',
    'debit' => 26.10
  ),
  array(
    'date' => "21/09",
    'nature' => 'Trouvé par terre',
    'credit' => 50
  ),
);
```

La moindre des choses sur un relevé de compte, avant même de commencer le tableau des opérations, c'est de rappeler à l'usager le solde restant sur son compte. Rajoutons donc ces quelques lignes dans notre fichier *fpdf_banque.php* :

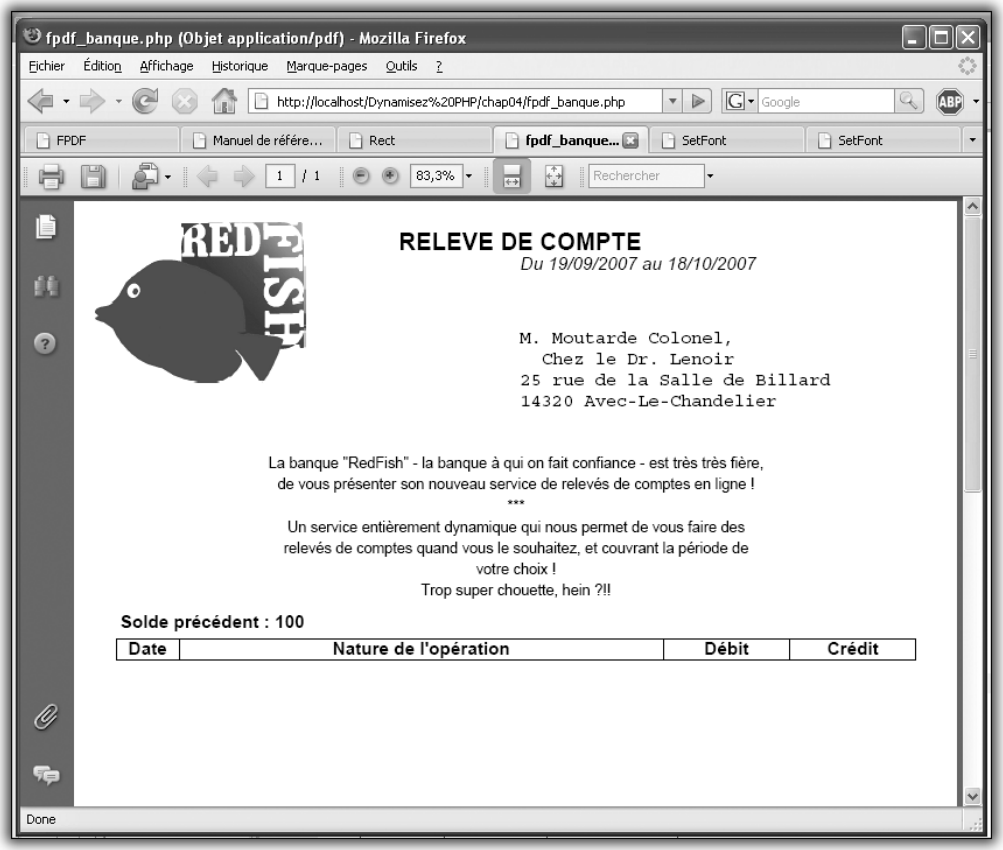
```
$soldePrecedent = "Solde précédent : ".$ancienSolde;
$pdf->SetFont('Arial','B',12);
$pdf->Cell(100, 0, utf8_decode($soldePrecedent));
```

Attaquons-nous à la première ligne de notre tableau : les en-têtes de colonnes. Ajoutez le code suivant à *fpdf_banque.php* :

```
//saut de ligne
$pdf->Ln(4) ;

// Un tableau

//Les en-têtes
$pdf->SetLineWidth(0.4);

$pdf->SetX(10);
$pdf->Cell(15, 5, 'Date', 1, 0, 'C');
$pdf->Cell(115, 5,
  utf8_decode('Nature de l\'opération'),
  1, 0, 'C');
$pdf->Cell(30, 5, utf8_decode('Débit'),
  1, 0, 'C');
$pdf->Cell(30, 5, utf8_decode('Crédit'),
  1, 0, 'C');
```

Tout d'abord, nous appelons la fonction `SetLineWidth()`, qui nous permet de choisir une épaisseur pour nos traits. Ensuite, nous appelons la fonction `SetX()`, afin d'être sûr de la position du point de coordonnées courant. C'est seulement après que nous commençons à placer les cellules. Nous pouvons déjà voir que nous ne touchons pas au point de coordonnées courant : dès qu'il a fini une cellule, il se positionne automatiquement à la suite de celle-ci, prêt à en démarrer une autre !

Dernier petit détail : le quatrième paramètre est à 1, ce qui signifie que chaque cellule va se retrouver habillée par une bordure de l'épaisseur précisée par la fonction `SetLineWidth()`. Regardez donc votre travail depuis votre navigateur préféré.



► Fig. 4.6 : Des cellules encadrées

Les lignes du tableau

Nous avons, dans le fichier *fpdf_donnees.php*, créé la variable `$operation`, qui est en fait un tableau contenant toutes les opérations que nous allons lister maintenant. Voici le code à ajouter au fichier *fpdf_banque.php* :

```
//les lignes
$pdf->SetFont('Arial','',12);
$pdf->SetFillColor(230, 230, 230);

$mouvementDebit = 0;
$mouvementCredit = 0;

for($i = 0 ; $i < count($operation) ; $i++){
  $pdf->Ln();
```

```
  //définit s'il faut remplir cette ligne
  //avec la couleur de remplissage ou non.
  $remplir = 0;
  if($i % 2 != 0){
    $remplir = 1;
  }

  //Cellule 'Date'
  $pdf->Cell(15, 5, $operation[$i]['date'], 'L', 0, 'C', $remplir);
  //Cellule 'Nature'
  $pdf->Cell(115, 5, utf8_decode($operation[$i]['nature']), 'L', 0, 'L',
➥ $remplir);
  //Cellule 'Debit'
  $pdf->Cell(30, 5, utf8_decode($operation[$i]['debit']), 'L', 0, 'R',
➥ $remplir);
  $mouvementDebit += $operation[$i]['debit'];
  //Cellule 'Credit'
  $pdf->Cell(30, 5, utf8_decode($operation[$i]['credit']), 'LR', 0, 'R',
➥ $remplir);
  $mouvementCredit += $operation[$i]['credit'];
}
```

La fonction `SetFillColor()` que nous utilisons ici permet de définir la couleur de remplissage par défaut. Les paramètres sont des nombres compris entre `0` et `255` et servent à définir la proportion de couleur pour chacune des trois composantes de la lumière. Le premier paramètre décrit la composante rouge, le deuxième la composante verte, et le troisième la composante bleue.

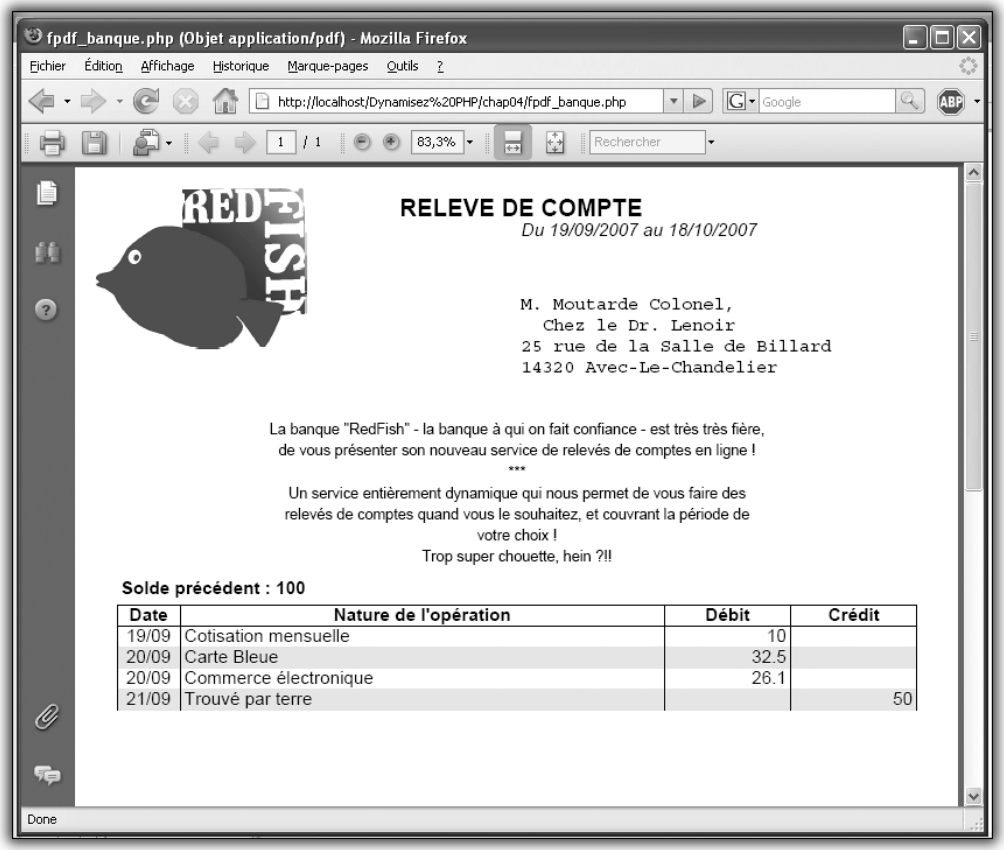
Le reste du code se passe dans la boucle `for` : nous commençons par sauter une ligne (fonction `Ln()`) puis, grâce à un petit test if, nous déterminons si le numéro de la ligne est pair ou impair. Cela nous sera utile pour colorer le fond d'une ligne sur deux.

Viennent alors les cellules en elles-mêmes. Nous observons tout d'abord que le quatrième paramètre de la fonction `Cell()`, celui qui sert à mettre ou non la bordure, n'est plus un entier, mais une chaîne de caractères. Voici les différentes valeurs que peut prendre ce quatrième paramètre :

- `0` : pas de bordure.
- `1` : la bordure est présente.
- `'L'` : (comme *left*) seulement la bordure droite de notre cellule.
- `'T'` : (comme *top*) seulement la bordure supérieure de notre cellule.
- `'R'` : (comme *right*) seulement la bordure droite de notre cellule.

- `'B'` : (comme *bottom*) seulement la bordure inférieure de notre cellule.
- Vous pouvez aussi utiliser plusieurs de ces lettres. Par exemple, la chaîne `'LR'` (ou la chaîne `'RL'`, l'ordre n'a pas d'importance) fera apparaître la bordure sur les côtés gauche et droit de notre cellule.

Enfin, un dernier paramètre a fait son apparition au sein de la fonction `Cell()`. Paramétré à `1` ou à `0`, il définit si oui (`1`) ou non (`0`) la cellule doit être remplie avec la couleur de remplissage courante ; ce qui nous permet d'alterner les lignes colorées avec celles non colorées.



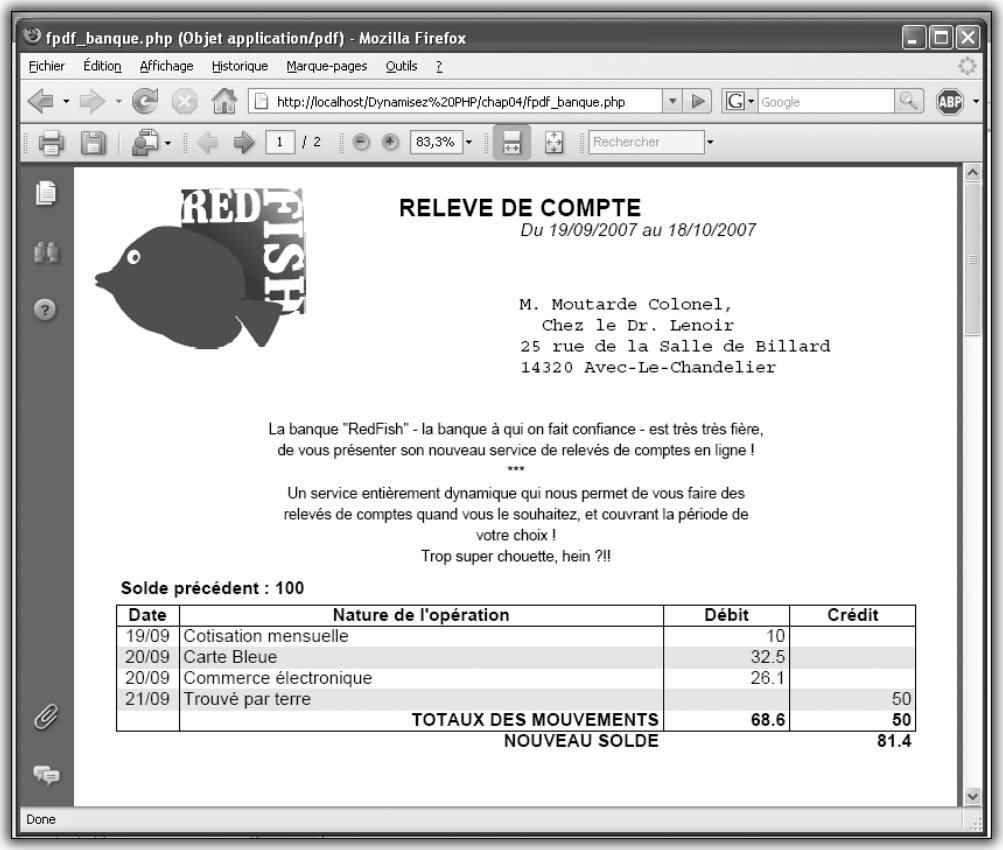
► Fig. 4.7 : Les lignes du tableau

Finir le tableau

Nous avons pratiquement terminé le tableau. D'ailleurs, avec les connaissances que vous avez acquises, vous devriez être capable de terminer tout seul ce tableau. Voici néanmoins comment nous nous y prenons :

```
//Dernière ligne
$pdf->SetFont('Arial','B',12);
$pdf->Ln();
$rempli = 0;
if(count($operation) % 2 != 0){
  $rempli = 1;
}
//Cellule 'Date'
$pdf->Cell(15, 5, '', 'LB', 0, 'C', $rempli);
//Cellule 'Nature'
$repere = $pdf->GetX();
$pdf->Cell(115, 5, 'TOTAUX DES MOUVEMENTS', 'LB', 0, 'R', $rempli);
//Cellule 'Debit'
$pdf->Cell(30, 5, $mouvementDebit, 'LB', 0, 'R', $rempli);
//Cellule 'Debit'
$pdf->Cell(30, 5, $mouvementCredit, 'LBR', 0, 'R', $rempli);

//ligne "nouveau solde"
$pdf->Ln();
$pdf->SetX($repere);
$pdf->Cell(115, 5, 'NOUVEAU SOLDE', 0, 0, 'R');

//calcul du nouveau solde
$nouveauSolde = $mouvementCredit-$mouvementDebit+$ancienSolde;
$celluleDebit = '';
$celluleCredit = '';
if($nouveauSolde < 0){
  $celluleDebit .= $nouveauSolde;
} else {
  $celluleCredit .= $nouveauSolde;
}
$pdf->Cell(30, 5, $celluleDebit, 0, 0, 'R');
$pdf->Cell(30, 5, $celluleCredit, 0, 0, 'R');
```

Comme vous pouvez le constater, nous avons déjà utilisé et expliqué toutes les fonctions présentes dans ce bout de code. Admirez donc le résultat de votre travail. Et profitez-en pour le personnaliser ou le peaufiner.

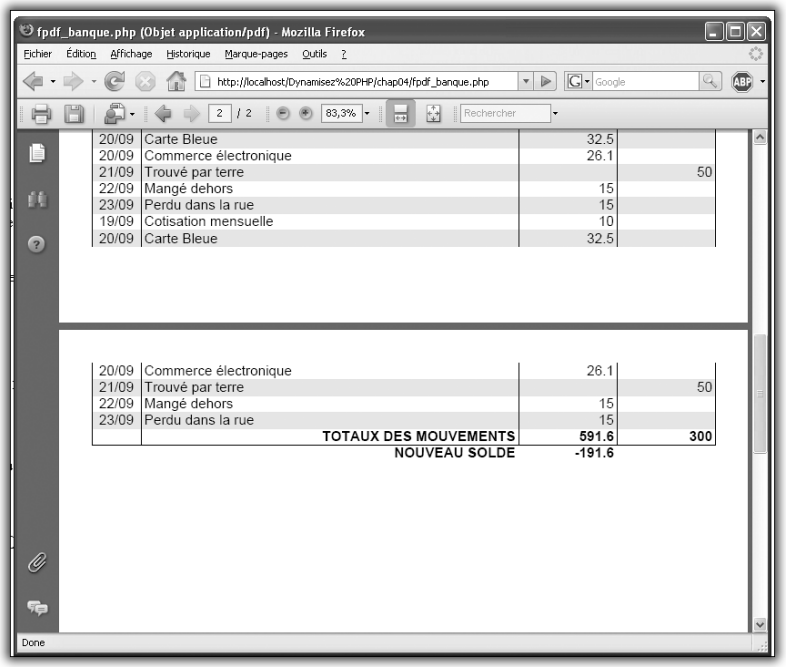


► Fig. 4.8 : Notre tableau est fini.

4.2 À propos des pages

Un document PDF peut être composé d'une ou plusieurs pages. Nous avons déjà vu la fonction `AddPage()` (au tout début de ce chapitre), qui nous permet justement de rajouter une page. Il existe cependant un moyen de rajouter une page implicitement.

Dans votre fichier *fpdf_donnees.php*, veuillez ajouter une quinzaine d'entrées au tableau `$operation`, puis relancez le fichier *fpdf_banque.php* dans votre navigateur.



► Fig. 4.9 : Création d'une page automatique

Comme vous le constatez, notre tableau est à présent trop grand pour ne tenir que sur la première page. FPDF va donc créer une seconde page. Il en sera de même pour un bloc de texte.

Un petit peu de dessin

Comme tous les logiciels générant du PDF, FPDF dispose de quelques fonctions de dessins, que nous allons examiner. À la toute fin de votre fichier *fpdf_banque.php* (avant l'appel à la fonction `Output()`), ajoutez le code suivant :

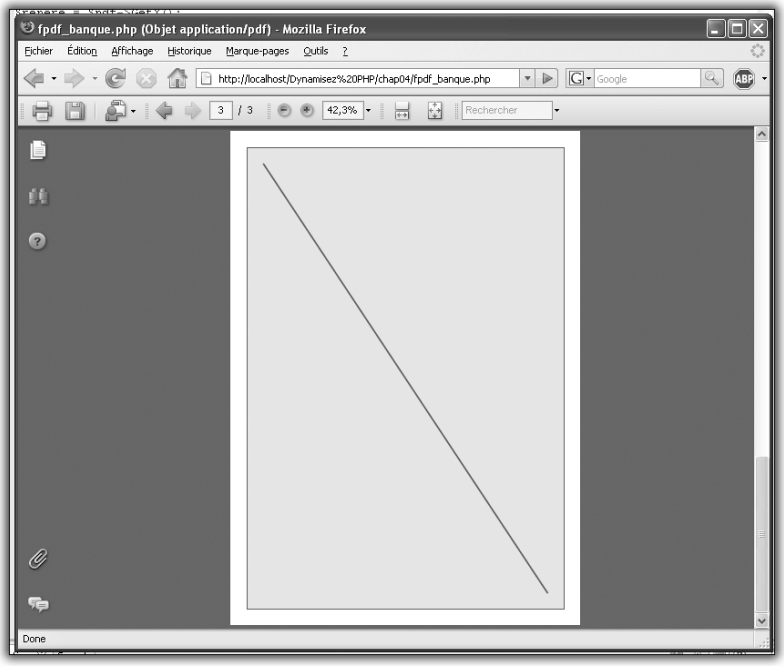
```
//Ajouter une page
$pdf->AddPage();

$pdf->SetDrawColor(100, 100, 100);

//dessiner un rectangle
$pdf->Rect(10, 10, 190, 277, 'DF');

//dessiner une ligne
$pdf->SetLineWidth(1);
$pdf->Line(20, 20, 190, 277);
```

Nous commençons par utiliser la fonction `SetDrawColor()`, dont le but est d'allouer une couleur de tracé courante. Elle s'utilise avec les trois composantes couleurs de la lumière (rouge, verte et bleue), exactement comme son homologue `SetFillColor()`.

Nous utilisons ensuite la fonction `Rect()` qui dessine un rectangle. Dans notre exemple, le coin supérieur gauche du rectangle est placé au point de coordonnées (10, 10), comme l'indiquent les deux premiers arguments. Notre rectangle fait aussi 190 mm de largeur (troisième argument) sur 277 de hauteur (quatrième paramètre). Le dernier paramètre indique comment colorer notre rectangle. Il peut prendre les valeurs suivantes :

- `'D'` ou `''` (chaîne vide) : dessine seulement le contour du rectangle (utilise la couleur de tracé courante).
- `'F'` : remplit le rectangle avec la couleur de remplissage courante, mais ne dessine pas son contour.
- `'DF'` ou `'FD'` : remplit le rectangle avec la couleur de remplissage courante, et dessine son contour avec la couleur de tracé courante.

Enfin, nous appelons la fonction `Line()` qui, comme son nom l'indique, trace une ligne avec la couleur de tracé courante. Les deux premiers arguments de cette fonction définissent les coordonnées du point de départ de notre ligne, et les deux derniers, le point d'arrivée.



► Fig. 4.10 : Un rectangle et un trait

Nous avons fait le tour des fonctions de dessins de FPDF. Comme vous avez pu le constater, il existe seulement deux fonctions pour dessiner, ce qui est effectivement très limité. Mais PDF n'a pas été conçu pour dessiner. Et puis, rien ne vous empêche de concevoir des images et des graphiques avec GD2 ou MagickWand, pour étayer ensuite vos documents PDF.

> ***Vous pouvez vous référer au chapitre* Manipuler des images *pour plus de détails sur l'utilisation de MagickWand ou de GD2.***

En-têtes et pieds de page

Il existe, dans FPDF, deux fonctions qui sont appelées à chaque création de page. Il s'agit des fonctions `Header()` et `Footer()`. Enfin, elles existent, mais… elles sont vides.

En fait, nous avons la possibilité de surcharger ces fonctions, de manière à établir un en-tête et un pied de page en accord avec votre document, son nombre de pages, etc.

Comme il nous faut surcharger ces deux fonctions, nous allons créer une classe dérivant de FPDF dans un nouveau fichier, *fpdf_classe.php*, que voici :

fpdf_classe.php

```
<?php
require('fpdf/fpdf.php');

class PDF extends FPDF{
  //En-tête
  function Header(){

    $this->SetFont('Arial','I',8);
    $this->SetXY(2.5, 2.5);
    $this->Cell(30,0,'Banque RedFish');
    $this->SetXY(10, 10);
  }

  //Pied de page
  function Footer(){
    //Positionnement à 10mm du bas
    $this->SetXY(5, -10);
    $this->SetFont('Arial','I',8);
    //Numéro de page
    $this->Cell(0,10,'Page '.$this->PageNo().'/{nb}',0,0,'C');
  }
}
?>
```

Rien de difficile : nous nous contentons de dériver la classe FPDF en une nouvelle classe appelée PDF, puis nous redéfinissons simplement les fonctions `Header()` et `Footer()`. Ainsi, nous précisons dans la fonction `Header()` que nous voulons un texte écrit en Arial, italique, 8 pt, que nous plaçons au point de coordonnées (2.5, 2.5).

> ⚠ Faites attention à vos unités
>
> Notre unité de mesure, dans *fpdf_banque.php*, est le mm. Ainsi, lorsque nous plaçons nos éléments dans les fonctions `Header()` et `Footer()`, c'est le millimètre qui est utilisé. Si d'aventure, vous souhaitez travailler avec des pouces, n'oubliez pas de convertir les valeurs de ces deux fonctions.

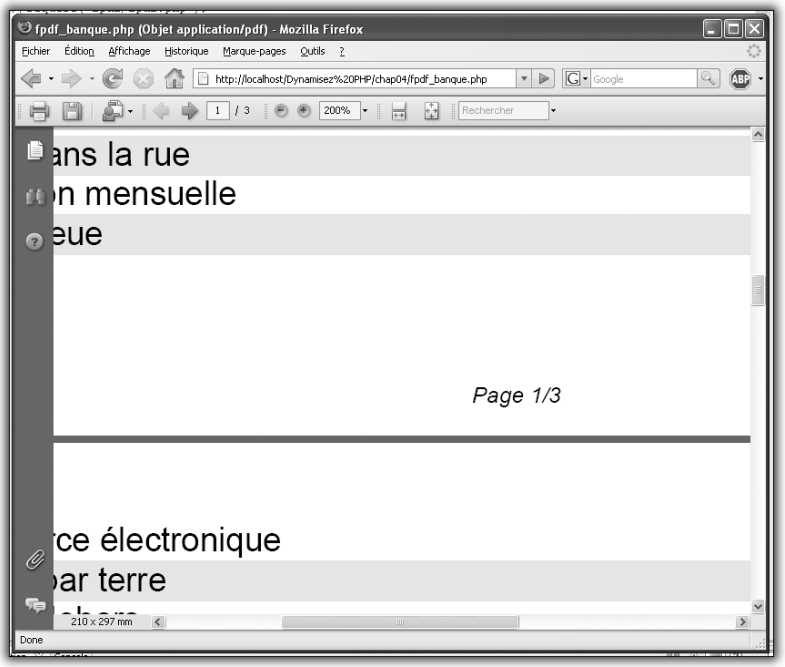
Nous utilisons ici, dans ce petit bout de classe, une nouvelle fonction : `PageNo()`. Cette fonction renverra le numéro de la page courante (si nous sommes sur la première page, elle renverra `1`, si nous sommes sur la deuxième, elle renverra `2`, etc.). Cette fonction est à utiliser de concert avec la petite chaîne de caractères `{nb}`, qui renvoie toujours le nombre de pages du document.

Avant de procéder à un test, il reste des modifications à effectuer dans le fichier *fpdf_banque.php*. Nous allons effectivement changer les premières lignes de ce programme :

```
<?php
//require('fpdf/fpdf.php');
include 'fpdf_donnees.php';
include 'fpdf_classe.php';

//Créer une instance de FPDF
//$pdf=new FPDF();
$pdf = new PDF();
$pdf->AliasNbPages();
...
...
```

Le fichier *fpdf.php* n'est plus requis, puisque nous utilisons la classe `PDF`, contenue dans le fichier *fpdf_classe.php*, qu'il ne nous faut pas oublier d'inclure. Le constructeur aussi change : nous n'utilisons plus `FPDF()` mais `PDF()`.

Juste après le constructeur, nous appelons une nouvelle fonction : `AliasNbPages()`. Cette fonction n'a rien d'exceptionnel ou de remarquable. Simplement, si l'on veut utiliser la fonction `PageNo()` dans notre en-tête ou dans notre pied de page, il nous faut appeler cette fonction. Et pas n'importe où : seulement entre l'appel du constructeur et l'ajout de la première page !



► Fig. 4.11 : Le footer

4.3 Importer une police de caractères

Utiliser seulement les quelques polices de caractères parmi les plus classiques lors de la création d'un document PDF est assez ennuyeux. Heureusement, FPDF nous permet d'avoir recours à des polices TrueType. Attention, la démarche est assez fastidieuse.

Première étape : générer un fichier de métrique

Bien évidemment, nous n'allons pas utiliser de police TrueType en trois lignes de code.

Allez donc chercher une petite archive zip contenant un utilitaire à l'adresse suivante : **http://www.fpdf.org/fr/dl.php?id=33**.

Créez ensuite, à la racine de votre disque dur *C :*, un répertoire que vous nommerez *ttf2pt1*. Décompressez votre archive dans ce répertoire. Vous devriez avoir deux fichiers : *ttf2pt1.exe* et *COPYRIGHT*. Copiez ensuite, toujours dans le répertoire *C:\ttf2pt1*, un fichier de police TrueType. Dans notre exemple, nous allons utiliser le fichier *comic.ttf*.

C'est maintenant que les barbarismes commencent. Ouvrez une nouvelle fenêtre d'**Invite de commandes** (menu **Démarrer** puis **Exécuter**. Tapez ensuite `cmd` dans la fenêtre qui

est apparue, puis cliquez sur OK). Naviguez jusqu'à rejoindre votre répertoire *c:\ttf2pt1*, puis entrez la commande suivante :

```
ttf2pt1 -a comic.ttf comic
```

Ici, *comic.ttf* est le nom du fichier de police, et *comic* est le nom de la police auquel nous nous référerons lorsque nous souhaiterons l'utiliser au sein de FPDF.

► Fig. 4.12 : Pour générer un fichier de métrique…

Regardez maintenant le contenu de votre répertoire *C:\ttf2pt1* : les fichiers *comic.afm* et *comic.t1a* sont bel et bien présents.

Deuxième étape : générer le fichier de définition de police

Regardez le contenu de votre répertoire *chap04\fpdf\font* : ici, des fichiers PHP sont stockés. Chacun d'entre eux sert à définir une des polices que FPDF peut utiliser. Vous les reconnaissez d'ailleurs à leurs noms très explicites : *courier.php*, *helvetica.php*… Notre but est maintenant de rajouter le fichier *comic.php*.

Dans votre répertoire *chap04*, créez encore un nouveau répertoire que vous nommerez *font*. Copiez dans ce répertoire les fichiers *comic.ttf* et *comic.afm* (le fichier que nous venons de générer avec l'utilitaire *ttf2pt1*). Toujours dans ce même répertoire, créez le fichier PHP *fpdf_export_police.php* et remplissez-le comme suit :

fpdf_export_font.php

```
<?php
require('../fpdf/font/makefont/makefont.php');
MakeFont('comic.ttf','comic.afm','cp1252');
?>
```

Nous utilisons ici la fonction `MakeFont()`, issue du fichier *makefont.php* (ce fichier fait partie de l'archive FPDF que vous avez téléchargée au début de ce chapitre).

Le premier paramètre de cette fonction indique le fichier TrueType. Il sera intégré au fichier PDF. Si toutefois vous ne souhaitez pas l'intégrer, car vous pensez que ce fichier TrueType est présent sur la machine de l'utilisateur, ou que vous ne voulez pas augmenter le poids du fichier PDF que vous allez générer, ou pour toute autre raison, vous pouvez laisser une chaîne vide.

Le deuxième paramètre indique où trouver le fichier *.afm* que nous avons généré ultérieurement. Il est indispensable.

Le troisième paramètre indique, par une chaîne de caractères, le type d'encodage à utiliser. Il peut prendre les valeurs suivantes :

- `'cp1250'` (Europe centrale)
- `'cp1251'` (cyrillique)
- `'cp1252'` (Europe de l'Ouest)
- `'cp1253'` (grec)
- `'cp1254'` (turc)
- `'cp1255'` (hébreu)
- `'cp1257'` (pays baltes)
- `'cp1258'` (vietnamien)
- `'cp874'` (thaï)
- `'ISO-8859-1'` (Europe de l'Ouest)
- `'ISO-8859-2'` (Europe centrale)
- `'ISO-8859-4'` (pays baltes)
- `'ISO-8859-5'` (cyrillique)
- `'ISO-8859-7'` (grec)
- `'ISO-8859-9'` (turc)
- `'ISO-8859-11'` (thaï)
- `'ISO-8859-15'` (Europe de l'Ouest)
- `'ISO-8859-16'` (Europe centrale)
- `'KOI8-R'` (russe)
- `'KOI8-U'` (ukrainien)

Il faut, bien entendu, que votre fichier de police dispose des caractères correspondant à l'encodage choisi. Vous n'avez plus qu'à lancer le fichier *fpdf_export_font.php* dans votre

navigateur. Le fichier *comic.php* aura été généré dans le même répertoire que *fpdf_export_font.php*.

Compresser sa police

Si vous avez la compression ZLIB activée, le fichier *comic.z* sera aussi généré. Il contient aussi votre police, mais compressée.

Troisième et dernière étape : utiliser la police

Copiez les fichiers *comic.php*, *comic.z* (s'il a été généré) et éventuellement (pour le ranger au même endroit) le fichier *comic.ttf* dans votre répertoire *chap04\fpdf\font*. Nous pouvons enfin utiliser la police Comic dans notre prochain document PDF. Voyons comment s'y prendre. Créez, dans votre répertoire *chap04*, le fichier *fpdf_comic.php* et remplissez-le comme suit :

fpdf_comic.php

```
<?php
require('fpdf/fpdf.php');
$pdf=new FPDF();
$pdf->AddPage();

//Ecrire une chaîne
$pdf->AddFont('Comic','');
$pdf->SetFont('Comic','U',16);
$pdf->Cell(100, 0, utf8_decode('Une police importée !'));

$pdf->Output();
?>
```

Vous pouvez, si vous le souhaitez, lancer ce fichier depuis votre navigateur préféré.

► Fig. 4.13 : La police importée

Voyons maintenant la dernière fonction de notre étude de FPDF : la fonction `AddFont()`.

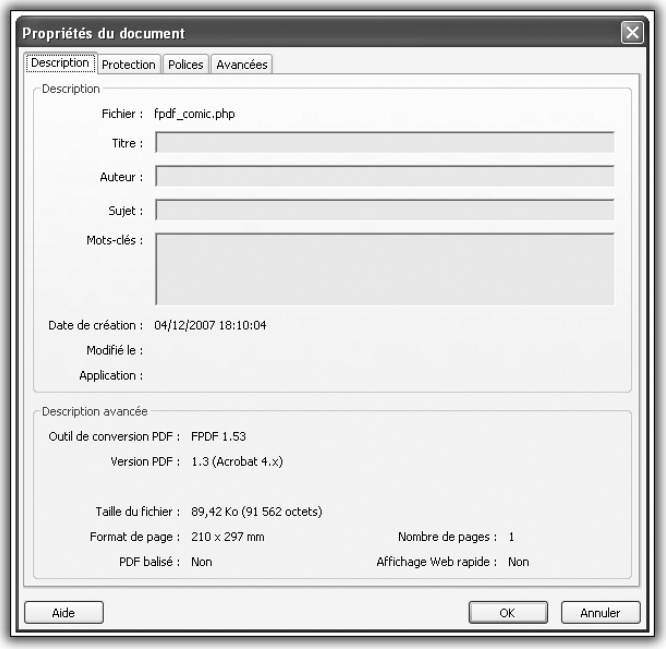
Cette fonction sert à importer dans notre document la police de caractères dont nous avons besoin. Le deuxième paramètre n'est pas obligatoire, mais il définit le style de la police.

Une police TrueType est en fait organisée en plusieurs fichiers : un fichier pour les caractères normaux, un pour les caractères en italique, un pour les gras et un pour les gras italiques. Nous n'avons, dans notre exemple, traité que le fichier contenant les caractères normaux. C'est pour cela que nous mettons une chaîne vide comme deuxième argument. Si nous avions choisi de traiter aussi les caractères gras, nous aurions appelé une seconde fois la fonction `AddFont()`, de cette manière :

```
$pdf->AddFont('Comic', 'B') ;
```

4.4 Signer un document PDF

Ouvrez votre fichier *fpdf_comic.php* dans votre navigateur, puis faites la combinaison de touches Ctrl+D, afin d'ouvrir la fenêtre des propriétés du document. Vous remarquerez que tous les champs sont vides.

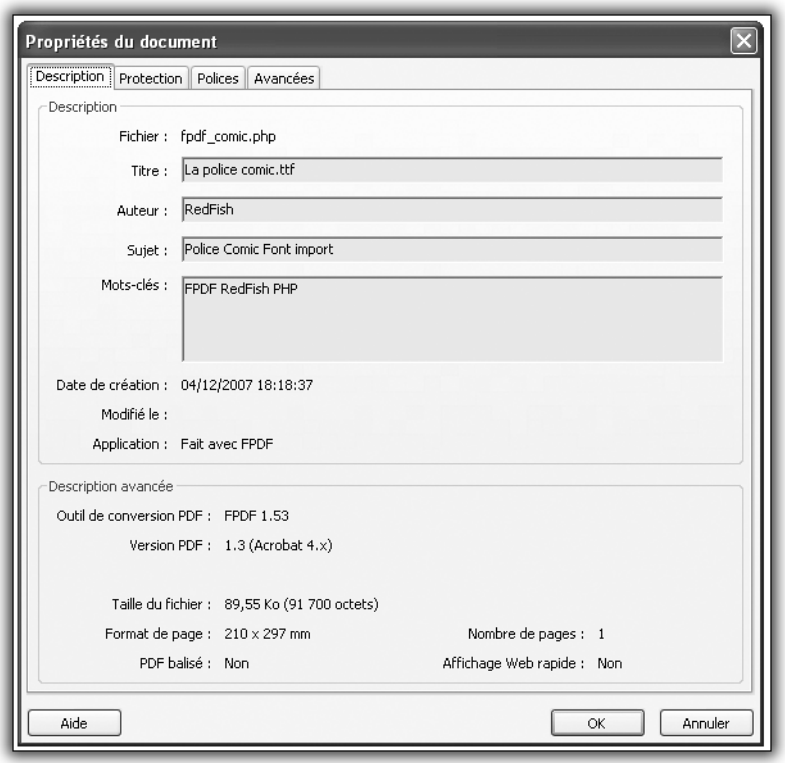


► Fig. 4.14 : Que de champs vides…

Nous allons y remédier. Ajoutez ce code à votre fichier *fpdf_comic.php*, juste avant l'appel à la fonction `Output()` :

```
//propriétés du document
$pdf->SetAuthor('RedFish');
$pdf->SetCreator('Fait avec FPDF');
$pdf->SetKeywords('FPDF RedFish PHP');
$pdf->SetSubject('Police Comic Font import');
$pdf->SetTitle(utf8_decode('La police comic.ttf'));
```

Les noms de ces fonctions sont explicites, ne trouvez-vous pas ? Elles ne prennent toutes qu'une seule chaîne de caractères comme argument. Vous pouvez relancer votre *fpdf_comic.php* depuis votre navigateur, et regarder les propriétés de votre document.



▶ Fig. 4.15 : Les champs sont remplis !

4.5 Check-list

Nous avons terminé l'étude de FPDF. Dans ce chapitre, nous avons vu comment :

- ✔ générer des documents PDF avec PHP, les enregistrer localement, les afficher sur le navigateur ou proposer aux utilisateurs de les télécharger ;
- ✔ importer des images dans un document PDF ;
- ✔ écrire du texte, des blocs de texte et créer des tableaux ;
- ✔ utiliser les fonctions de dessins de FPDF ;
- ✔ créer des en-têtes et des pieds de page ;
- ✔ importer une police TrueType ;
- ✔ signer vos documents PDF.

5

Gérer vos annuaires avec LDAP

Aujourd'hui, les bases de données sont utilisées à tout bout de champ et ce, même lorsque d'autres solutions s'avèrent plus efficaces et plus rapides. Vous allez comprendre que LDAP s'adapte aussi très bien lorsqu'il s'agit de gérer un ensemble d'êtres humains (employés, contacts professionnels, etc.).

5.1 LDAP ? Qu'est-ce que c'est ?

LDAP est un acronyme pour *Lightweight Directory Access Protocol*. Comme vous pouvez le voir par son dernier mot (*Protocol*), LDAP est un protocole comme il en existe beaucoup en informatique : HTTP, TCP, IP, etc.

En français, on pourrait le traduire, plus ou moins littéralement, par "protocole d'accès à un répertoire de taille légère", le terme Lighweight étant aussi utilisé en boxe pour désigner une catégorie de poids… très léger bien sûr.

Le mot Directory signifiant répertoire, LDAP obéit bien à la logique des fichiers et dossiers au niveau de l'arborescence. En fait, LDAP fonctionne comme un système d'arborescence de dossiers et fichiers mais pour des informations afin de créer un annuaire contenant une section (souvent appelée Monde), des sous-sections, des sous sous-sections, et ainsi de suite.

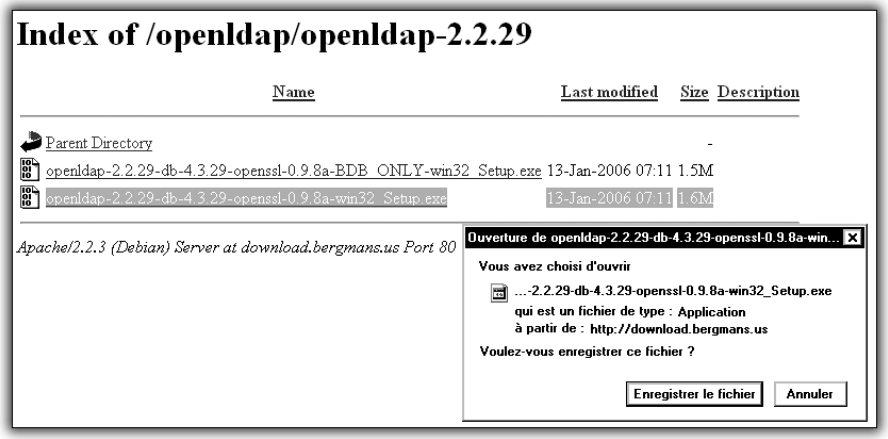
Vous allez découvrir, dans la partie qui suit, l'installation sous Windows. En ce qui concerne l'installation sous Linux, LDAP est souvent installé par défaut avec la distribution. Cependant, pour ceux qui ne l'ont pas, reportez-vous à l'aide de votre distribution de Linux. Les commandes diffèrent quelque peu et les versions aussi en fonction de la distribution de Linux que vous utilisez : Debian, Gentoo, RedHat, Mandrake, etc. Puis, afin de bien comprendre comment fonctionne LDAP, vous concevrez l'arborescence de l'annuaire qu'il sera nécessaire de mettre en place dans le cas pratique et vous verrez comment agir sur LDAP avec PHP. Pour finir, vous mettrez en pratique l'arborescence conçue pour créer un annuaire qui vous permettra de mémoriser tous vos contacts professionnels et/ou personnels. LDAP étant très simple à utiliser, le cas pratique se fera tout au long de la section *Manipulations dans LDAP* et vous aurez le code complet placé en fin de chapitre comme pour les autres extensions étudiées précédemment. Seuls des formulaires HTML de saisie seront ajoutés dans les scripts du cas pratique

5.2 Installation

Windows

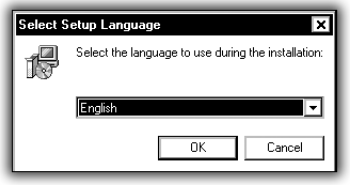
Allez sur le site **http://download.bergmans.us/openldap** et cliquez avec le bouton gauche de la souris sur le répertoire *openldap-x.x.xx*. Au moment de l'écriture de cet article, la dernière version était 2.2.29.

1 Une fois dans ce répertoire, vous devez télécharger le fichier *openldap-2.2.29-db-4.3.29-openssl-0.9.8a-win32_Setup.exe*.



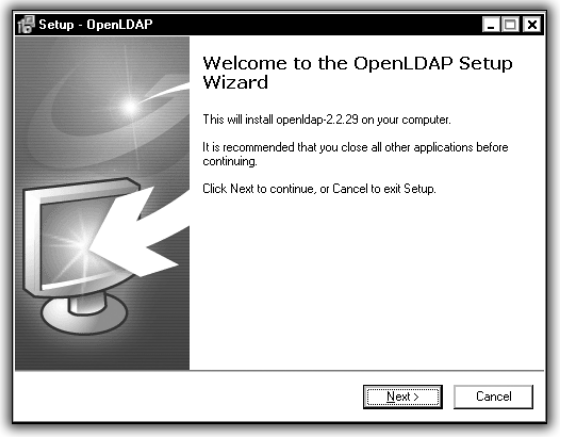
▶ Fig. 5.1 : Télécharger le fichier .exe

2 Puis installez-le. Cela se fait en quelques étapes seulement. Commencez par choisir la langue. Le choix est restreint : anglais (*English*) ou allemand (*Deutsch*).



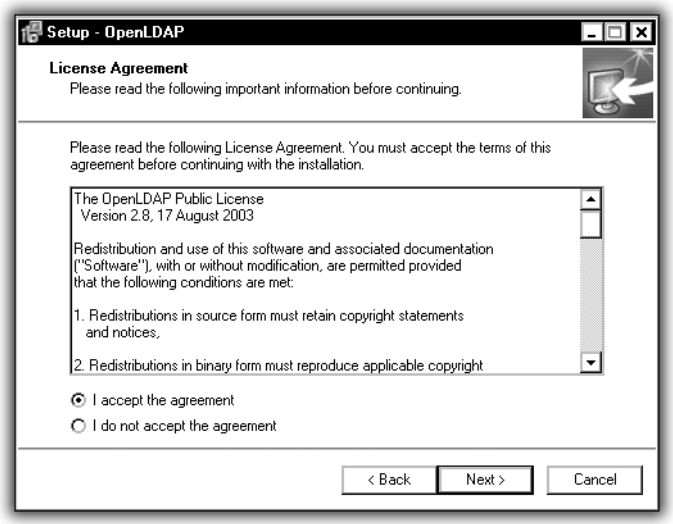
▶ Fig. 5.2 : Choisir la langue

3 Fermez tous les programmes actifs, pensez surtout à désactiver l'antivirus, et cliquez sur **Suivant**.



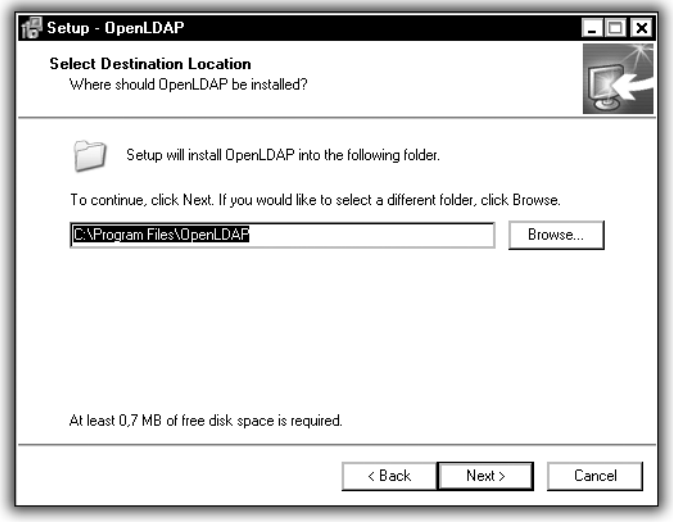
▶ Fig. 5.3 : Départ de l'installation

4 Acceptez la licence et cliquez sur **Suivant**.



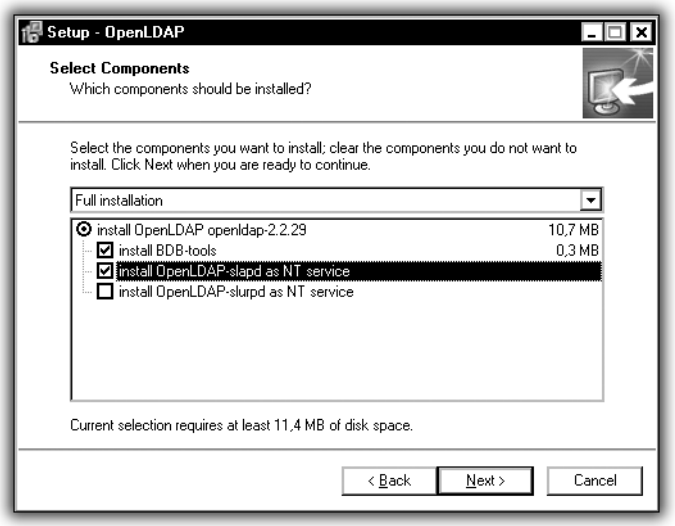
► Fig. 5.4 :
Acceptez la licence

5 Choisissez ensuite la destination d'installation, vous pouvez laisser le chemin par défaut.



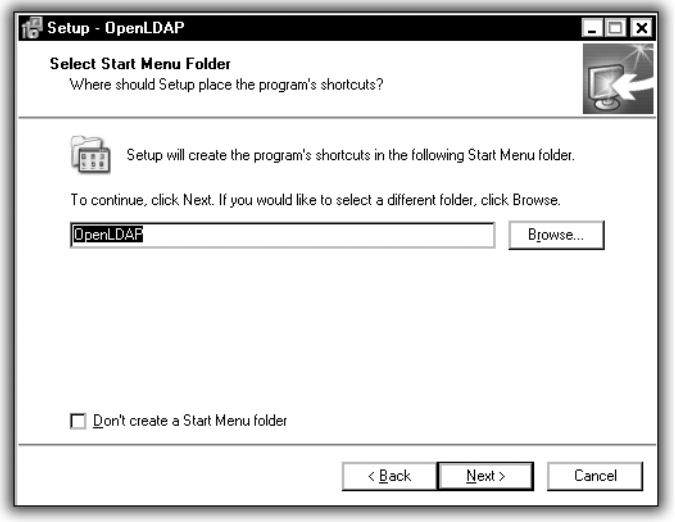
► Fig. 5.5 :
Destination d'installation

6 Si vous désirez installer aussi *slurpd*, cochez la case correspondante. *slurpd et slapd.*



▶ Fig. 5.6 : Choisir les outils à installer

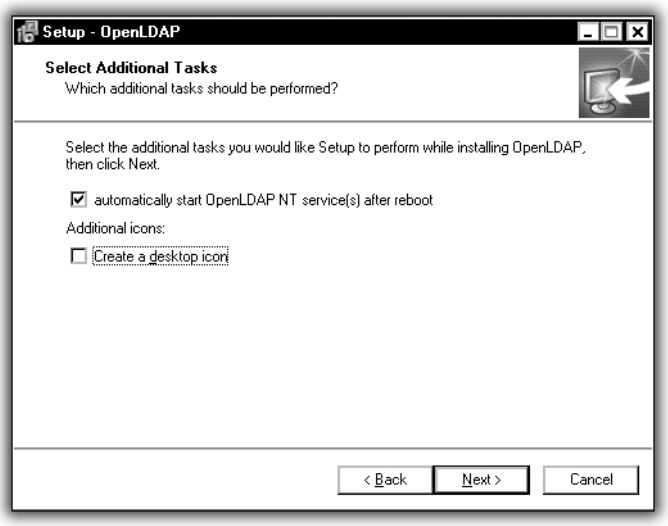
7 Si vous désirez un dossier dans le menu **Démarrer**, saisissez un nom ou laissez celui par défaut. Si vous ne voulez par qu'un dossier soit dans le menu **Démarrer**, cochez la case en bas à gauche.



▶ Fig. 5.7 : Dossier dans menu Démarrer

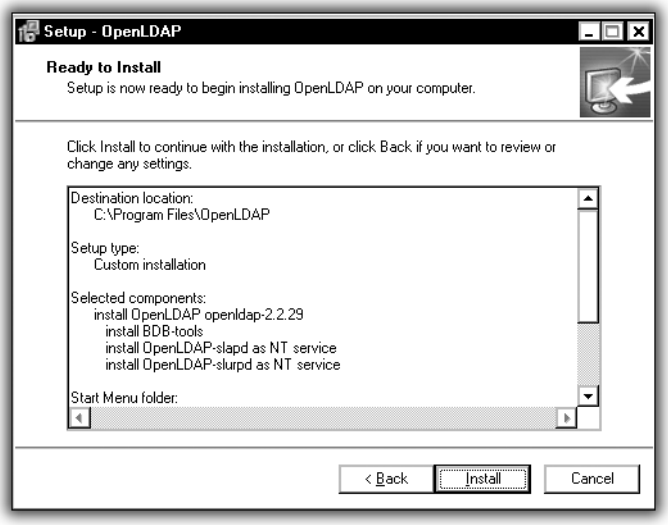
8 Pour l'étape suivante, il vaut mieux laisser la première option cochée. Cela permettra de démarrer le démon LDAP chaque fois que vous allumerez votre ordinateur. À vous de décider si vous désirez placer une icône sur le bureau ou non. Dans

l'affirmative, il est vivement conseillé de cocher l'option. Cela vous permettra de démarrer *slapd* aussitôt l'installation sans avoir à redémarrer votre ordinateur.



▶ Fig. 5.8 : Démarrage à chaque boot de Windows

9 Procédez à un dernier contrôle pour être sûr que toutes les bonnes informations ont été sélectionnées. Si tout est correct, cliquez sur **Install**.



▶ Fig. 5.9 : Vérifications avant installation

10 Terminez l'installation.

► Fig. 5.10 : Cliquez sur Finish

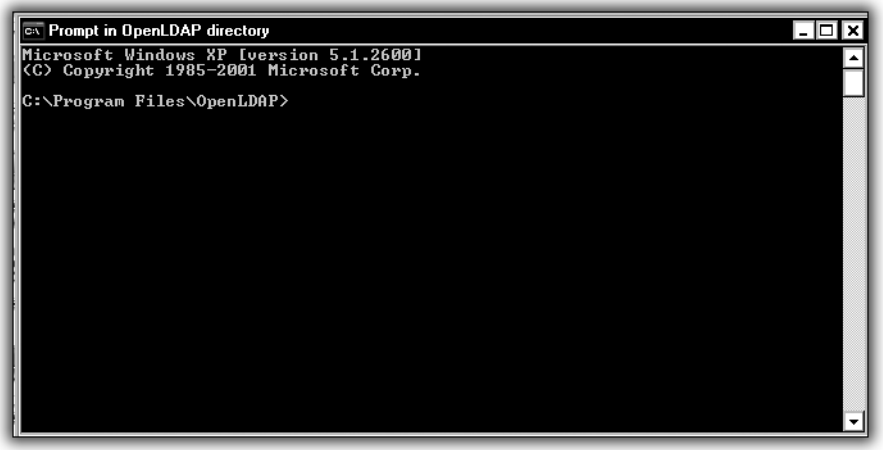
Et voilà, vous avez réussi l'installation.

Maintenant que l'installation est terminée, il ne vous reste plus qu'à passer à la configuration. Éditez le fichier *slapd.conf*. Si vous avez laissé la destination par défaut lors de l'installation, ce fichier se situe dans *C:\Program Files\OpenLDAP*. À la fin du fichier, vous pouvez prendre connaissance de la configuration par défaut. Vous pouvez laisser en l'état pour les tests de ce chapitre.

```
52  #######################################################################
53  # BDB database definitions
54  #######################################################################
55
56  database    bdb
57  suffix      "dc=domaine,dc=org"
58  rootdn      "cn=Manager,dc=domaine,dc=org"
59  # Cleartext passwords, especially for the rootdn, should
60  # be avoid.  See slappasswd(8) and slapd.conf(5) for details.
61  # Use of strong authentication encouraged.
62  rootpw      secret
63  # The database directory MUST exist prior to running slapd AND
64  # should only be accessible by the slapd and slap tools.
65  # Mode 700 recommended.
66  directory   ./data
67  # Indices to maintain
68  index   objectClass eq
```

► Fig. 5.11 : slapd.conf édité avec Notepad++

Fermez le fichier et cliquez deux fois rapidement (double-clic) sur l'icône de votre bureau. Une fenêtre MS-DOS s'ouvre. Cette fenêtre doit rester ouverte autant de temps que vous utilisez LDAP. Par confort de travail, certains prennent l'habitude de la réduire.



► Fig. 5.12 : Lancement de OpenLDAP

Toujours dans le répertoire *C:\Program Files\OpenLDAP*, créez un fichier que vous allez nommer *defaut.ini* et saisissez les lignes suivantes :

```
dn: dc=domaine,dc=org
objectclass: top
objectclass: dcObject
objectclass: organization
o: Domaine
dc: domaine
```

Puis ouvrez une fenêtre MS-DOS en double-cliquant sur l'icône qui s'est placée sur votre Bureau lors de l'installation. Une fenêtre MS-DOS va s'ouvrir avec le prompt directement placé dans *C:\Program Files\OpenLDAP*. Saisissez la commande `slapadd –f slapd.conf –l config.ini`. Si tout s'est bien passé, rien ne s'est affiché à part le prompt dans la ligne en dessous. Fermez la fenêtre MS-DOS puis allez dans le répertoire LDAP et double-cliquez sur l'icône *slapd.exe*. LDAP est maintenant démarré et est fonctionnel.

Il vous reste à présent une autre procédure à effectuer concernant la partie PHP. Modifiez le fichier *php.ini* et dans la section `Dynamic Extensions` du fichier, enlevez le point-virgule (**;**) à la ligne `;extension=php_ldap`, enregistrez le changement, fermez le fichier *php.ini* et redémarrez le serveur web (Apache2). Vous allez maintenant vérifier si tout s'est bien passé en vous connectant à LDAP via PHP. Créez un répertoire *chap05*

dans le répertoire *dynamisez_php*. Puis créez le fichier *connexion_deconnexion.php* dans le répertoire *chap05* et insérez dans le fichier le code suivant :

tester_ldap.php

```
<?php
$ldc = ldap_connect('localhost');
if(!$ldc){
  echo 'Impossible de se connecter au serveur LDAP';
} else {
  echo 'Connexion au serveur LDAP effectuée avec succès';
}

ldap_close($ldc);
?>
```

Exécutez le script dans votre navigateur et si tout s'est bien passé, le message "Connexion au serveur LDAP effectuée avec succès" se sera affiché. Si c'est votre cas, vous n'avez plus qu'à vous féliciter. Vous pouvez passer à la section : *Interaction entre PHP et LDAP*.

Linux

En ce qui concerne l'installation sous Linux, voici quelques sites en fonction de votre distribution sur lesquels vous pourrez aller pour installer LDAP sur votre distribution Linux.

Debian

Pour cette installation, nous nous sommes basés sur la ressource trouvée à la page web de Coagul, **www.coagul.org/article.php3?id_article=172**.

En principe, LDAP fait partie intégrante de Linux. Cependant, dès que l'on prend un peu d'assurance avec le système Linux, on a très vite envie de se faire sa propre configuration en installant les packages en fonction de ses besoins. Nous partons donc du principe que LDAP n'est pas encore installé. Nous allons vous expliquer étape par étape comment installer et configurer OpenLDAP sur une Debian. Pour cette installation, vous pouvez utiliser Etch (stable) sur un ordinateur portable premier prix.

Ouvrez une console. Il en existe plusieurs sous Linux : *xterm*, *eterm*, etc. Saisissez la commande `su` suivi du mot de passe de l'administrateur Linux (*root*) demandé. Vous savez que vous êtes connecté en tant qu'administrateur *root* lorsque, juste avant le curseur, vous avez le signe dièse (#) au lieu du signe dollar ($).

Listez les paquets concernés par OpenLDAP en saisissant la commande suivante : `apt-cache search openldap` :

- *libldap2 – OpenLDAP libraries* ;
- *gforge-ldap-openldap – collaborative development tool – LDAP directory (using OpenLDAP)* ;
- *ldap-utils – OpenLDAP utilities* ;
- *ldapscripts – Add and remove user and groups (stored in a LDAP directory)* ;
- *libdbd-ldap-perl – Perl extension for LDAP access via an SQL/Perl DBI interface* ;
- *libldap-2.3-0 – OpenLDAP libraries* ;
- *libldap-ruby1.8 – OpenLDAP library binding for Ruby1.8* ;
- *libldap2-dev – OpenLDAP development libraries* ;
- *libsasl2-modules-ldap – Pluggable Authentification Modules for SASL (LDAP)* ;
- *python-ldap – A LDAP interface module for Python* ;
- *slapd – OpenLDAP server (slapd)* ;
- *smbldap-tools – scripts to manage Unix and Samba accounts stored on LDAP.*

Tous ces paquets ne sont pas indispensables pour utiliser LDAP via PHP. En fait, seuls le serveur et le client LDAP sont nécessaires. Dans certaines documentations, la commande indiquée pour installer LDAP est :

```
# apt-get install ldap-server ldap-client
```

Or, sous Etch, cette commande affiche une note comme quoi `ldap-server` sera remplacé par `slapd` et `ldap-client` sera remplacé par `ldap-utils`. D'ailleurs, il est très facile de remarquer que `slapd` et `ldap-utils` sont visibles dans la liste des paquets précédents alors que `ldap-server` et `ldap-client` ne le sont pas.

1. Commencez l'installation :

```
# apt-get install slapd ldap-utils
```

2. Éditez le fichier de configuration :

```
# nano /etc/ldap/slapd.conf
```

3. Modifiez la ligne 57 du fichier :

```
suffix      "dc=mon_annuaire,dc=org"
```

4. Décommentez la ligne 61 et modifiez-la :

```
rootdb     "cn=Manager,dc=mon_annuaire,dc=org"
```

5 Puis ajoutez à la ligne suivante :

```
rootpw     secret
```

6 Pour finir, modifiez les lignes 96 à 100 :

```
access to attrs=UserPassword,shadowLastChange
       by dn="cn=Manager,dc=mon_annuaire,dc=org"
       by anonymous auth
       by self write
       by * none
```

7 Ainsi que les lignes 115 à 177 :

```
access to *
      by dn="cn=Manager,dc=mon_annuaire,dc=org" write
      by * read
```

8 Et c'est fini. Il ne reste plus qu'à redémarrer le serveur LDAP avec la commande suivante :

```
# /etc/init.d/slapd restart
```

Vous devez ensuite vérifier que LDAP est reconnu par PHP. Pour cela, exécutez le fichier PHP qui contient la fonction `phpinfo()`. Ensuite, assurez-vous que la connexion à LDAP fonctionne par le biais de PHP avec la configuration qui a été modifiée. Exécutez dans un navigateur le script suivant :

tester_ldap.php

```
<?php
$ldc = ldap_connect('localhost');
if(!$ldc){
  echo 'Impossible de se connecter au serveur LDAP';
} else {
  echo 'Connexion au serveur LDAP effectuée avec succès';
}

ldap_close($ldc);
?>
```

Enfin… il y a eu bonne découverte de LDAP avec PHP.

Gentoo

Rendez-vous sur le site suivant : **www.gentoo.org/doc/fr/ldap-howto.xml**.

RedHat

Rendez-vous sur **www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/fr/ref-guide/index.html** et cliquez sur le lien "*13 Protocole LDAP (Lightweight Directory Access Protocol)*".

Suse

LDAP est installé par défaut avec Suse.

5.3 Gérer LDAP avec PHP

Se connecter/déconnecter

Connexion

Se connecter au serveur LDAP est aussi simple que de se connecter à un serveur MySQL. La fonction se nomme `ldap_connect()` et sa syntaxe est la suivante :

- `ressource ldap_connect([chaîne $nom_serveur[, entier $port]]);`

La fonction prend en premier paramètre, optionnel, le nom du serveur LDAP (`$nom_serveur`) et en second paramètre, également optionnel, le numéro de port (`$port`) sur lequel LDAP écoute les requêtes qui lui sont destinées. Le port par défaut en ce qui concerne LDAP est le port TCP 389. Le résultat retourné par LDAP est un identifiant de connexion en cas de succès et `FALSE` en cas d'échec. Le résultat de `FALSE` concerne la version 3 de LDAP uniquement.

LDAP 2.X.X

Avec LDAP 2.X.X, `ldap_connect()` retournera toujours un identifiant, que la connexion réussisse ou non. En cas d'échec, il retournera une ressource en initialisant uniquement les identifiants de connexion.

```
<?php
$ldc = ldap_connect('localhost', 389)
  or die('Impossible de se connecter au serveur LDAP');

/* instructions */
?>
```

die() vs Exception

Afin de faciliter la compréhension, et de raccourcir le code, nous avons utilisé la fonction `die()`. Cependant, cette fonction est à éviter pour une question de sécurité. De plus, depuis la version 5 de PHP, les exceptions sont comprises par défaut dans le noyau de PHP et nous vous conseillons vivement de les utiliser dans vos projets. Si ce n'est pas le cas, étudiez les exceptions sérieusement et intégrez-les dans vos projets.

La variable `$ldc` est l'identifiant de connexion (en anglais : *link_identifier*).

Déconnexion

La déconnexion est encore plus simple que la connexion et se passe totalement de commentaires :

- `int ldap_close(ressource $id_connexion)`

```
<?php
/* instructions */

ldap_close($ldc);
?>
```

Manipulations dans LDAP

Authentification au serveur LDAP : ldap_bind()

- `bool ldap_bind(ressource $id_connexion`
  `[, chaîne $bind_rdn [, chaîne $bind_pass]])`

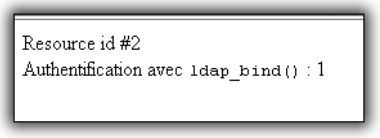
Plutôt que vous donner longue définition, voici du code :

mauvaise_authentification.php

```
<?php
$ldc = ldap_connect('localhost') or die('Impossible de se connecter au
➥ serveur LDAP');
echo $ldc .'<br/>';

// Authentification au serveur LDAP
echo 'Authentification avec <code>ldap_bind()</code> : ';
$ldb = ldap_bind($ldc);
echo $ldb;
```

```
ldap_close($ldc);
?>
```

Utiliser le script tel quel va vous générer un message d'erreur Warning exprimant un problème de protocole :



▶ Fig. 5.13 :
Warning à l'authentification

Pourquoi ? Le serveur LDAP est en version de protocole 3 alors que PHP envoie ses ordres en version de protocole 2. Il est donc nécessaire d'ajouter la ligne :

`ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);` avant la ligne `$ldb = ldap_bind($ldc);` afin de permettre l'authentification (anonyme) au serveur LDAP.

bonne_authentification.php

```
<?php
$ldc = ldap_connect('localhost')
   or die('Impossible de se connecter au serveur LDAP');
echo $ldc .'<br/>';
// Authentification au serveur LDAP
echo 'Authentification avec <code>ldap_bind()</code> : ';

// Voici la ligne qui évite le message de Warning
ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

$ldb = ldap_bind($ldc);
echo $ldb;

ldap_close($ldc);
?>
```

- `bool ldap_set_option(ressource $id_connexion,`
  `entier $option, mixte $nouvelle_valeur_option);`

`$option` correspond donc à une option dont celle-ci possède déjà une valeur, attribuée par défaut par PHP, que vous désirez modifier (la valeur). Le premier paramètre, `$id_connexion`, est l'identifiant de connexion défini lors de l'appel de la fonction `ldap_connect()` et le troisième, `$nouvelle_valeur_option`, correspond à la nouvelle valeur que vous voulez affecter à l'option placée en deuxième paramètre.

Les options possibles sont des constantes prédéfinies dans l'extension LDAP de PHP :

- `LDAP_OPT_DEREF` (entier) détermine comment les aliases sont considérés durant la recherche. Il doit avoir pour valeur une de celles qui suivent : `LDAP_DEREF_NEVER`, `LDAP_DEREF_SEARCHING`, `LDAP_DEREF_FINDING` ou `LDAP_DEREF_ALWAYS`.
- `LDAP_OPT_SIZELIMIT` (entier) définit une limite dans le nombre de résultats retournés par une recherche (vous pouvez par exemple retourner les 100 premiers noms même si la recherche en trouve un nombre supérieur). La valeur `0` à cette option demande de retourner tous les résultats sans limite.
- `LDAP_OPT_TIMELIMIT` (entier) impose une limite de temps maximal, en secondes, pour effectuer une recherche. La valeur `0` indique qu'il n'y a aucune limite de temps imposée.
- `LDAP_OPT_PROTOCOL_VERSION` (entier) indique le protocole de version LDAP utilisé afin de communiquer avec le serveur LDAP primaire.
- `LDAP_OPT_ERROR_NUMBER` (entier).
- `LDAP_OPT_REFERRALS` (booléen). La valeur de cette option doit être définie par `LDAP_OPT_ON` ou `LDAP_OPT_OFF`. Cette option est à `ON` par défaut.
- `LDAP_OPT_RESTART` (booléen) détermine si les opérations d'entrées/sorties de LDAP sont automatiquement redémarrées ou non en cas d'échec. La valeur de cette option doit être définie par `LDAP_OPT_ON` ou `LDAP_OPT_OFF`. Par défaut, cette option est à OFF.
- `LDAP_OPT_HOST_NAME` (chaîne).
- `LDAP_OPT_ERROR_STRING` (chaîne) définit un message d'erreur à retourner lorsqu'une erreur se produit dans la session LDAP.
- `LDAP_OPT_MATCHED_DN` (chaîne) fournit la valeur DN retournée avec la plus récente erreur produite dans la session LDAP.
- `LDAP_OPT_SERVER_CONTROLS` (tableau) fournit une liste par défaut de contrôles du serveur LDAP envoyée avec chaque requête à effectuer.
- `LDAP_OPT_CLIENT_CONTROLS` (tableau) fournit une liste par défaut des contrôles du client affectant la session LDAP.

Écriture dans LDAP

Ajouter un pays

Voici venu le moment d'effectuer des ajouts dans LDAP. Il est nécessaire de bien comprendre LDAP et son fonctionnement auparavant. LDAP fonctionne sous forme d'arborescence. Il y a un répertoire racine, souvent appelé Monde, puis des sous-répertoires. En fait, "Monde" correspond au domaine (`dc=mon_annuaire,dc=org` dans les exemples de ce chapitre).

⚠ Caractères accentués

LDAP ne prend pas en compte les caractères accentués. Par exemple, si vous essayez d'ajouter une personne dont le prénom est Frédéric, il faut insérer `Frederic` sinon vous obtiendrez un message d'erreur de type "*ldap_add() : Invalid Syntax...*"

Si vous avez lu le fonctionnement par hiérarchie de LDAP, vous comprenez que le premier objet à ajouter dans l'annuaire LDAP est le pays. Regardez le fichier *ajouter_un_pays.php* :

ajouter_pays.php

```
<?php
$dn = 'cn=Manager,dc=mon_annuaire,dc=org';
$passldap = 'secret';

$ldc = ldap_connect('localhost')
   or die('Impossible de se connecter au serveur LDAP');

// Voici la ligne qui évite le message de Warning
ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

// Authentification au serveur LDAP avec ldap_bind()
// Authentification avec identifiant
$ldb = ldap_bind($ldc, $dn, $passldap)
   or die('Authentification au serveur LDAP échouée');

// Ajouter  pays : définition des informations
$info['c'] = 'France';
$info['objectClass'][0] = 'country';
$info['objectClass'][1] = 'top';
$rdn = 'c='. $info['c'] .',dc=mon_annuaire,dc=org';

// Ajouter les données dans l'annuaire
$lda = ldap_add($ldc, $rdn, $info);
if($lda){
  echo 'ajout dans annuaire LDAP effectué avec succès';
} else {
  echo 'ajout dans annuaire LDAP échoué';
}

ldap_close($ldc);
?>
```

Analyse du code et explications

Ligne 5 : `$dn = 'cn=Manager,dc=mon_annuaire,dc=org';`

- `dn` signifie *distinguished name* que l'on pourrait traduire en français par nom unique (littéralement : nom distingué) ;
- `cn` signifie *commonName* (nom commun) ;
- `dc` signifie *domainComponent* (composants domaine) et contient toutes les composantes du nom de domaine. Si vous avez un nom de domaine du style `domaine.asso.fr`, alors vous devrez avoir la ligne suivante : `$dn = 'cn=Manager ,dc=domaine,dc=asso,dc=fr';`.

Ligne 16 : `$ldb = ldap_bind($ldc, $dn, $passldap);`

- Pour pouvoir ajouter des informations dans l'annuaire, l'utilisateur doit s'identifier au serveur LDAP. Une connexion anonyme, comme vue dans les exemples précédents, n'est donc pas possible. C'est pour cette raison que la fonction `ldap_bind()` contient la ressource de connexion en premier paramètre et également le domaine en deuxième paramètre et le mot de passe en troisième paramètre.
- La syntaxe de la fonction `ldap_bind()` est la suivante : `bool ldap_bind(resource link_id [, string rdn [, string pass]] )`. Les deuxième et troisième paramètres sont optionnels pour la lecture dans l'annuaire mais obligatoires pour pouvoir y insérer des données. En indiquant le premier paramètre dans la fonction, l'authentification est dite anonyme.

Ligne 20 : `$info['c'] = 'France';`

- `$info['c']` définit le pays. `c` signifie `countryName` (nom du pays). Si vous voulez respecter la norme RFC4519, vous devrez remplacer la valeur `France` par `FR` qui est le code `A2 de la norme ISO-3166-2 pour la France.`

Ligne 26 : `$lda = ldap_add($ldc, $rdn, $info);`

- `ldap_add()` est la fonction qui permet d'ajouter du contenu dans le répertoire LDAP.

LDAP est tellement riche qu'il est impossible d'en faire le tour en quelques pages. Si vous voulez plus de renseignements sur LDAP, nous vous conseillons de consulter les annexes où vous trouverez de nombreuses références à des ouvrages et des sites sur Internet.

Ajouter une organisation

ajouter_organisation

```
<?php
/**
 * ajouter_une_org.php
```

```
 */
$dn = 'cn=Manager,dc=mon_annuaire,dc=org';
$passldap = 'secret';

$ldc = ldap_connect('localhost')
   or die('Impossible de se connecter au serveur LDAP');

// Voici la ligne qui évite le message de Warning
ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

// Authentification au serveur LDAP avec ldap_bind()
// Authentification avec identifiant
$ldb = ldap_bind($ldc, $dn, $passldap)
   or die('Authentification au serveur LDAP échouée');

// Ajouter une organisation : définition des informations
$info['o'] = 'nom_organisation';
$info['objectClass'][0] = 'organization';
$info['objectClass'][1] = 'top';
$rdn = 'o='. $info['o'] .',c=France,dc=mon_annuaire,dc=org';

// Ajouter les données dans l'annuaire
$r = ldap_add($ldc, $rdn, $info);
if($r){
  echo 'données ajoutées : '. $rdn .'<br/>';
} else {
  echo 'ajout dans annuaire LDAP échoué<br/>';
}

ldap_close($ldc);
?>
```

Comme vous pouvez le remarquer, les changements par rapport à l'ajout d'un pays sont minimes. La fonction `ldap_add()` fonctionne de la même manière pour l'ajout d'une organisation que pour l'ajout d'un pays. Elle fonctionnera d'ailleurs de la même manière pour l'ajout d'une unité et également pour l'ajout d'un individu (contact).

Ajouter une unité

ajouter_unite.php

```
<?php
/**
 * ajouter_une_unite.php
 */
$dn = 'cn=Manager,dc=mon_annuaire,dc=org';
$passldap = 'secret';
```

```
$ldc = ldap_connect('localhost')
   or die('Impossible de se connecter au serveur LDAP');

// Voici la ligne qui évite le message de Warning
ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

// Authentification au serveur LDAP avec ldap_bind()
// Authentification avec identifiant
$ldb = ldap_bind($ldc, $dn, $passldap)
   or die('Authentification au serveur LDAP échouée');

// Ajouter une organisation : définition des informations
$info['ou'] = 'contacts_personnels';
$info['objectClass'][0] = 'organizationalUnit';
$info['objectClass'][1] = 'top';
$rdn = 'ou='. $info['ou']
➥ .',o=drapeau_suire,c=France,dc=mon_annuaire,dc=org';

// Ajouter les données dans l'annuaire
$r = ldap_add($ldc, $rdn, $info);
if($r){
  echo 'données ajoutées : '. $rdn .'<br/>';
} else {
  echo 'ajout dans annuaire LDAP échoué<br/>';
}

ldap_close($ldc);
?>
```

Tout comme pour l'ajout d'un pays ou d'une unité, la logique de fonctionnement de LDAP reste la même. Voilà un élément supplémentaire qui fait tout l'intérêt de LDAP : sa simplicité d'utilisation. Il est bien plus compliqué de concevoir l'arbre (arborescence) LDAP en lui-même (en fonction de la complexité de la structure à mettre en place) que de le programmer par la suite pour le rendre opérationnel.

Ajouter un individu

Tout comme pour les bases de données, il est indispensable, dans un annuaire LDAP, de définir un nom unique (une clé) pour chaque entrée. Par exemple, il peut y avoir plusieurs `David DRAPEAU (sn)` mais un seul `ddrapeau (cn)`

ajouter_individu.php

```
<?php
$dn = 'cn=Manager,dc=mon_annuaire,dc=org';
```

```
$passldap = 'secret';

$ldc = ldap_connect('localhost')
   or die('Impossible de se connecter au serveur LDAP');

// Voici la ligne qui évite le message de Warning
ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

// Authentification au serveur LDAP avec ldap_bind()
// Authentification avec identifiant
$ldb = ldap_bind($ldc, $dn, $passldap)
   or die('Authentification au serveur LDAP échouée');

// Ajouter une organisation : définition des informations
$info ["cn"] = "ddrapeau";
$info ["sn"] = "David DRAPEAU";
$info ["userPassword"] = "unmotdepasse";
$info ["objectClass"][0] = "person";
$info['objectClass'][1] = 'top';
$rdn = 'cn='. $info['cn'] .',ou=nom_unite_01,
   o=nom_organisation_01,c=France,dc=mon_annuaire,dc=org';

// Ajouter les données dans l'annuaire
$r = ldap_add($ldc, $rdn, $info);
if($r){
  echo 'données ajoutées : '. $rdn .'<br/>';
} else {
  echo 'ajout dans annuaire LDAP échoué<br/>';
}

ldap_close($ldc);
?>
```

Afin de rester très clair et concis, des éléments comme l'adresse e-mail (liste de diffusion par exemple) ou le numéro de sécurité sociale (employés d'une société par exemple) n'ont pas été pris en compte. Ajouter ces éléments ne complique en rien la programmation LDAP avec PHP.

Recherche/Lecture

ldap_get_entries()

Évidemment, une autre fonctionnalité primordiale d'un annuaire est de pouvoir accéder de nouveau à ses contacts. La fonction `ldap_get_entries()` est indispensable afin de stocker les données dans un tableau multidimensionnel.

- `array ldap_get_entries(ressource $id_connexion, ressource $id_resultat)`

Le second paramètre `$id_resultat` représente la valeur de retour de la fonction `ldap_search()`.

ldap_search()

Puis, il suffit d'utiliser `ldap_search()` associée à `ldap_get_entries()`.

- `resource ldap_search(ressource $id_connexion,`
  `chaîne $base_dn, chaîne $filtre[, tableau $attributs`
  `[, entier $attrs_seuls [, entier $taille_max[, entier $timelimit`
  `[, entier $deref]]]]])`

Comme vous pouvez vous en apercevoir, la fonction `ldap_search()` possède beaucoup de paramètres. En voici la description :

- `$id_connexion` est l'identifiant de connexion défini par `ldap_connect`
- `$base_dn` est le DN (nom distingué) de base du dossier
- `$filtre` est le motif de recherche à effectuer ;
- `$attributs` permet de filtrer au niveau du retour de résultat. C'est-à-dire qu'il ne peut retourner que le `sn` par exemple. Quel que soit votre choix, le DN sera toujours retourné ;
- `$attrs_seuls` est par défaut à la valeur `0`. Cela signifie qu'il retourne pour chaque attribut son type et sa valeur. Mettre `$attrs_seuls` à la valeur `1` signifie que seuls les types d'attributs seront retournés, sans les valeurs ;
- `$taille_max` représente le nombre de résultats maximal à retourner ;
- `$temps_max` spécifie le temps maximal, en secondes, de durée de la recherche ;
- `$deref` précise comment les alias doivent être gérés durant la recherche.

Ce qui donne pour lire les informations insérées grâce aux scripts précédents, le script *lecture_ldap.php* :

lecture_ldap.php

```
<?php
echo "<h3>Lecture dans LDAP</h3>";
$ldc = ldap_connect('localhost')
   or die('impossible de se connecter au serveur LDAP');

ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

ldap_bind($ldc)
   or die('authentification échouée');
```

```
// Liste de toutes les personnes
$lds = ldap_search($ldc, "dc=mon_annuaire,dc=org", "sn=*");
$ldge = ldap_get_entries ($ldc, $lds);
echo "Lecture de l'annuaire<br/>";

for ($n=0; $n < $ldge ["count"]; $n++){
  echo "dn : ". $ldge[$n]["dn"] ."<br/>";
  echo "cn : ". $ldge[$n]["cn"][0] ."<br/>";
  echo "sn : ". $ldge[$n]["sn"][0];
}

ldap_close ($ldc);
?>
```

Modifier une entrée

Un contact ne reste jamais immobile. Il change de numéro de téléphone plusieurs fois dans sa vie. Il déménage et il change donc également d'adresse, de code postal et de commune, etc. Il est donc indispensable de pouvoir modifier des entrées de l'annuaire. Pour effectuer toutes ces modifications, une seule fonction est nécessaire :

- `bool ldap_modify(ressource $id_connexion, chaîne $dn, tableau $entrees)`

Supprimer une entrée

Lorsqu'un compte ne sert plus à rien (départ en retraite d'un employé), supprimer ledit compte est la moindre des actions à effectuer afin de garder un répertoire propre et à jour. La fonction `ldap_delete()` permet de supprimer une entrée dans un répertoire LDAP :

- `bool ldap_delete(ressource $id_connexion, chaîne $dn)`

Observez le script *supprimer_contact.php* qui effectue la suppression du compte `ddrape02` :

supprimer_individu.php

```
<?php
echo "<h3>Suppression du compte ddrape02</h3>";
$ldc = ldap_connect('localhost')
  or die('impossible de se connecter au serveur LDAP');

ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);
$dn = "cn=Manager,dc=mon_annuaire,dc=org";
ldap_bind($ldc, $dn, "secret")
  or die('authentification échouée');
```

```
// Ajouter une organisation : définition des informations
$info ["cn"] = "ddrape02";
$rdn = 'cn='. $info['cn'] .',ou=contacts_professionnels,o=drapeau
➥ _suire,c=France,dc=mon_annuaire,dc=org';

// Liste de toutes les personnes
if($ldd = ldap_delete($ldc, $rdn)){
  echo 'suppression effectuée avec succès';
} else {
  echo 'échec lors de la tentative de suppression';
}

ldap_close ($ldc);
?>
```

Aller plus loin avec LDAP

Gestion des erreurs

Comme beaucoup d'extensions PHP, LDAP fournit deux fonctions pour récupérer les erreurs : `ldap_errno()` et `ldap_error()`. Vous apprendrez également à utiliser une troisième fonction qui est `ldap_err2str()`.

- `entier ldap_errno(ressource $id_connexion)` retourne le numéro d'erreur courant.
- `chaîne ldap_error(ressource $id_connexion)` retourne le message d'erreur sous forme de caractères.
- `chaîne ldap_err2str(entier $errno)` retourne le message d'erreur grâce au numéro d'erreur (`$errno`).

Le script *gestion_erreurs_ldap.php* sera beaucoup plus parlant :

gestion_erreurs_ldap.php

```
<?php
$ldc = ldap_connect('mauvais_serveur');
$ldb = ldap_bind($ldc);
if(!$ldb){
  $numero_erreur = ldap_errno($ldc);
  $message_erreur = ldap_error($ldc);
  $message_erreur_de_errno = ldap_err2str($numero_erreur);
  echo 'numéro erreur = '. $numero_erreur .'<br/>';
  echo 'message erreur = '. $message_erreur .'<br/>';
  echo 'message de errno = '. $message_erreur_de_errno;
} else {
```

```
  echo 'connexion au serveur LDAP effectuée avec succès';

}
ldap_close($ldc);
?>
```

Libérer la mémoire

Avec PHP, la mémoire est automatiquement libérée à la fin du script. Aussi, il est tout de même utile, à chaque fois que possible, de laisser un maximum d'espace mémoire libre à tout instant. La fonction `ldap_free_result()` effectue la même action que `mysql_free_result()` pour les BDD MySQL. C'est-à-dire qu'elle libère la mémoire une fois l'opération effectuée. Voici un exemple de l'utilisation de `ldap_free_result()` avec le script *gestion_memoire.php* :

- `bool ldap_free_result(ressource $id_resultat)`

`$id_resultat` est la variable `$ressource` à laquelle est affectée la manipulation LDAP. Le script *gestion_memoire.php* en donne un exemple :

gestion_memoire.php

```
<?php
/* instructions LDAP */

$identifiant_resultat = ldap_delete($ldc, $dn);

// Libération de l'espace mémoire allouée
// à l'exécution de la fonction ldap_delete()
mysql_free_result($identifiant_resultat);

/* Autres instructions LDAP */

/* Autres instructions */
?>
```

Les premiers… et les suivants

first_attribute()

- `chaîne ldap_first_attribute(ressource $id_connexion, resource $result_entry_identifier, int &$ber_identifier)`

Permet de récupérer le premier attribut d'une entrée. Tous les autres attributs peuvent être récupérés grâce aux fonctions `ldap_next_attribute()` et `ldap_get_attribute()`.

first_entry()

- `ressource ldap_first_entry(ressource $id_connexion, ressource $result)`

Permet de récupérer la première entrée d'un annuaire LDAP. Toutes les autres entrées peuvent être récupérées grâce aux fonctions `ldap_next_entry()` et `ldap_get_entries()`.

first_reference()

- `ressource ldap_first_reference(ressource $id_connexion,ressource $result)`

Permet de récupérer la première référence d'un annuaire LDAP. Toutes les autres références peuvent être récupérées grâce aux fonctions `ldap_next_reference()` et `ldap_get_reference()`.

5.4 Cas pratique : gérer ses contacts

Bien entendu, les exemples cités précédemment ne sont pas anodins. Ils vous ont permis de préparer le cas pratique. Maintenant que l'arbre LDAP est fonctionnel, il ne reste plus qu'à créer les formulaires qui permettent, entre autres, d'ajouter/modifier/supprimer un contact. Vous remarquerez que lors du script d'insertion d'un individu, un identifiant et un mot de passe ont été créés pour l'individu afin que le cas pratique de ce chapitre puisse s'adapter au plus grand nombre de situations possibles. C'est-à-dire que si vous désirez simplement créer un annuaire de contacts, vous pouvez éviter la création d'un mot de passe pour chaque utilisateur (individu). Mais s'il s'agit d'un annuaire de contacts pour un site Internet (ou toute autre application), et que chaque contact doit pouvoir se connecter à un back-office, alors ce cas pratique sera également fonctionnel.

Le script *ajouter_un_contact.php* présente un formulaire qui vous permet d'ajouter un nouvel individu. Il est simple et reprend en grande partie le script *ajouter_individu.php* étudié précédemment :

ajouter_entree.php

```
<html>
<head>
<title>Ajouter un contact</title>

</head>
<body>
<h1>Ajouter un contact dans l'annuaire</h1>

<?php
```

```
if(isset($_POST['submit']) && $_POST['nom'] != ''){
  $dn = 'cn=Manager,dc=mon_annuaire,dc=org';
  $passldap = 'secret';

  $ldc = ldap_connect('localhost')
    or die('Impossible de se connecter au serveur LDAP');

  // Voici la ligne qui évite le message de Warning
  ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

  // Authentification au serveur LDAP avec ldap_bind()
  // Authentification avec identifiant
  $ldb = ldap_bind($ldc, $dn, $passldap)
    or die('Authentification au serveur LDAP échouée');
  // Ajouter une organisation : définition des informations
  $info ["cn"] = $_POST['identifiant'];
  $prenom = ucfirst($_POST['prenom']);
  $nom = strtoupper($_POST['nom']);
  $info["sn"] = $_POST['prenom'] .' '. $_POST['nom'];
  $info["userPassword"] = $_POST['mot_de_passe'];
  $info ["objectClass"][0] = "person";
  $info['objectClass'][1] = 'top';
  $rdn = 'cn='. $info['cn'] .',ou=contacts_professionnels,o=drapeau
  ➥ _suire,c=France,dc=mon_annuaire,dc=org';

  // Ajouter les données dans l'annuaire
  $r = ldap_add($ldc, $rdn, $info);
  if($r){
  echo 'données ajoutées : '. $rdn .'<br/>';
  } else {
    echo 'ajout : '. $rdn .'<br/>dans annuaire LDAP échoué<br/>';
  }

  ldap_close($ldc);

}
?>

Informations du contact<br/>

<form action="#" method="POST">

Identifiant <input type="text" name="identifiant" />
Mot de passe <input type="password" name="mot_de_passe" /><br/>
Nom <input type="text" name="nom" />
Prénom <input type="text" name="prenom" /><br/>
```

```
<input type="submit" name="submit" value="Valider" /><br/>
</form>

</body>
</html>
```

Le script *ajouter_un_contact.php* présente un formulaire qui contient des champs de texte pour que vous puissiez saisir les informations du contact et un bouton **Valider** afin de les insérer dans LDAP. Gardez en mémoire, qu'avec LDAP, vous ne pouvez pas saisir de caractères accentués. Cela signifie que le prénom Stéphane devra être saisi `"Stephane"` (sans les guillemets, cela va de soi).

L'intérêt de créer un annuaire de contact est de pouvoir retrouver un contact très rapidement grâce à une recherche ciblée par un utilisateur. Le script *lister_contacts.php* montre un exemple de recherche d'un ou plusieurs contacts sur le `sn` :

lister_entrees.php

```
<html>
<head>
<title>Afficher la liste</title>

</head>

<body>
<form action="#" method="POST">
Rechercher <input type="text" name="rechercher" size="30" />
<br/>
<input type="submit" name="submit" value="Lancer la recherche" />
</form>
<br/>
<?php
if(isset($_POST['submit']) && $_POST['rechercher'] != ''){
  $rechercher = trim(htmlspecialchars(strip_tags($_POST['rechercher']),
  ➥ ENT_QUOTES));
  echo "<h3>Résultat de la recherche</h3>";
  $ldc = ldap_connect('localhost')
    or die('impossible de se connecter au serveur LDAP');

  ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);

  ldap_bind($ldc)
    or die('authentification échouée');


  // Liste de toutes les personnes
```

```
  $lds = ldap_search($ldc, "dc=mon_annuaire,dc=org", "sn=$rechercher*");
  $ldge = ldap_get_entries ($ldc, $lds);
  echo "Lecture de l'annuaire<br/>";

  for ($n=0; $n < $ldge ["count"]; $n++){
    echo "dn : ". $ldge[$n]["dn"] ."<br/>";
    echo "cn : ". $ldge[$n]["cn"][0] ."<br/>";
    echo "sn : ". $ldge[$n]["sn"][0];
  }

  ldap_close ($ldc);

}

?>
</body>
</html>
```

Bien entendu, il peut être nécessaire de supprimer un contact. Par exemple, un employé quitte l'entreprise pour une quelconque raison (fin de contrat, démission, licenciement, etc.), et il est donc bien utile de pouvoir supprimer son compte afin de ne pas laisser dans l'annuaire LDAP des comptes inactifs. Regardez le script *supprimer_entree.php* qui effectue cette opération à merveille :

supprimer_entree.php

```
<html>
<head>
<title>Supprimer une entrée</title>

</head>

<body>
<form action="#" method="POST">
Rechercher <input type="text" name="supprimer" size="30" />
<br/>
<input type="submit" name="submit" value="Lancer la recherche" />
</form>
<br/>
<?php
if(isset($_POST['submit']) && $_POST['supprimer'] != ''){
  $supprimer = trim(htmlspecialchars(strip_tags($_POST['supprimer']),
  ➥ ENT_QUOTES));
  echo "<h3>Supprimer une entrée</h3>";
  $ldc = ldap_connect('localhost')
    or die('impossible de se connecter au serveur LDAP');
```

```
  ldap_set_option($ldc, LDAP_OPT_PROTOCOL_VERSION, 3);
  $dn = "cn=Manager,dc=mon_annuaire,dc=org";
  ldap_bind($ldc, $dn, "secret")
    or die('authentification échouée');

  // Ajouter une organisation : définition des informations
  $rdn = 'cn='. $supprimer .',ou=contacts_professionnels,o=drapeau
  ➥ _suire,c=France,dc=mon_annuaire,dc=org';

  // Liste de toutes les personnes
  if($ldd = ldap_delete($ldc, $rdn)){
    echo 'suppression effectuée avec succès';
  } else {
    echo 'échec lors de la tentative de suppression';
  }

  ldap_close ($ldc);
}
?>
</body>
</html>
```

Vous voilà maintenant équipé d'une application sommaire pour gérer vos contacts, employés et autres personnes grâce à un annuaire LDAP. Comme pour tous les autres chapitres, afin de satisfaire au plus grand nombre, les cas pratiques sont volontairement basiques et très ouverts à toutes évolutions ultérieures en fonction de vos besoins. Très utiles dans cet exemple pour des authentifications, vous pouvez toujours ajouter des informations supplémentaires pour chaque individu comme par exemple une adresse e-mail ou encore un numéro de téléphone. Nous vous souhaitons ainsi de faire de belles découvertes dans le monde de PHP/LDAP. Vous pouvez passer maintenant à un chapitre tout autant utile et intéressant qui est l'abstraction d'accès aux bases de données.

5.5 Check-list

Au cours de ce chapitre, vous avez pu approcher les techniques de gestion d'annuaire avec LDAP. Nous avons vu notamment :

- ✔ l'installation d'OpenLDAP sous Windows et Linux Debian ;
- ✔ la conception d'une arborescence LDAP ;
- ✔ la création de l'arborescence LDAP par le biais de PHP ;
- ✔ l'ajout, la modification et la suppression d'une entrée dans l'annuaire LDAP ;
- ✔ des fonctions complémentaires comme la gestion des erreurs, ou la mémoire.

6

Gérer les données avec PDO

PDO (*PHP Data Object*) est une bibliothèque programmée en C pour PHP. Son avantage par rapport à d'autres bibliothèques de bases de données telles que PEAR::DB ou AdoDB, qui sont les plus connues, est que PDO est beaucoup plus rapide par rapport aux deux autres qui sont programmées en PHP et non en C comme l'est PDO.

6.1 Installation

Windows

Installer PDO sous Windows est très simple. Éditez le fichier *php.ini* et décommentez la ligne `;extension=php_pdo.dll` en enlevant le point-virgule (`;`) en début de ligne. Faites de même pour les extensions des pilotes des SGBD (Système de gestion de bases de données) que vous désirez utiliser : `extension=php_pdo_*.dll` où * est le nom du pilote du SGBD (par exemple `extension=pdo_php_mysql.dll`). Redémarrez le serveur web.

Linux

L'exemple cité ici est pour Debian Etch. Les autres versions ne doivent guère être beaucoup plus différentes.

Par le gestionnaire de paquets apt-get

Avec les sources

Téléchargement

Allez sur le site **ftp://ftp.fr.postgresql.org/latest** et téléchargez la version adéquate. Lors de la rédaction de ce chapitre, la dernière version (*latest*) était *postgresql-8.2.5.tar.gz*. Ensuite, placez-vous en `root` pour la suite des manipulations.

Vérification de l'archive téléchargée

Pour cet exemple, les sources ont été téléchargées dans le répertoire */usr/local/src*.

```
# cd /usr/local/src
```

Vérification du checksum

```
# md5sum postgresql-8.2.5.tar.gz
```

Si le checksum est OK, alors l'installation peut se poursuivre.

Décompression du fichier

```
# tar -xzvf postgresql-8.2.5.tar.gz && cd postgresql-8.2.5
```

Configuration

```
# ./configure --prefix=/usr/local/pgsql
```

Compilation

```
# make && make install
```

Démarrage

```
# cd /usr/local/pgsql
# adduser postgres
Suivre les indications
# mkdir /usr/local/pgsql/data
# chown postgres /usr/local/pgsql/data
# su - postgres
postgres@domus:~$ /usr/local/pgsql/bin/inidb -D /usr/local/pgsql/data
...
success
```

Il y a deux méthodes pour démarrer le serveur PostgresSQL. Soit en avant-plan avec la commande suivante :

```
postgres@domus:~$ /usr/local/pgsql/bin/postgres -D \
   /usr/local/pgsql/data
```

Soit en arrière-plan avec la commande suivante :

```
postgres@domus:~$ nohup /usr/local/pgsql/bin/postgres -D \
   /usr/local/pgsql/data </dev/null >>server.log 2>&1 </dev/null
```

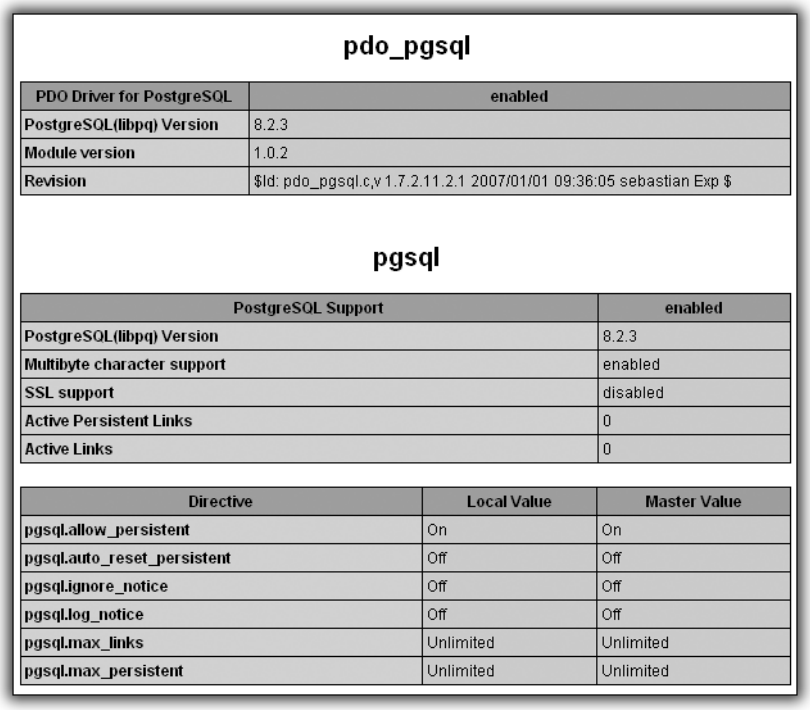
Pour arrêter le serveur en arrière-plan, il suffit d'exécuter la commande :

```
postgres@domus:~$ kill 'cat /usr/local/pgsql/data/postmaster.pid'
```

Et il ne reste plus qu'à activer PDO dans PHP. Les tests ont été effectués sur Debian Etch avec PHP6 que vous pouvez télécharger (les sources) sur **http://snaps.php.net/**. À l'heure où ces lignes sont rédigées, la version PHP6 n'est pas encore sortie officiellement. Il est nécessaire que PHP soit compilé avec les options `--with-pgsql=/DIR` et `--with-pdo-pgsql=/DIR`.

Vérifier l'installation

Exécutez votre script favori, *phpinfo.php*, qui contient la fonction `phpinfo()`. Et voici le résultat :



pdo_pgsql

PDO Driver for PostgreSQL	enabled
PostgreSQL(libpq) Version	8.2.3
Module version	1.0.2
Revision	$Id: pdo_pgsql.c,v 1.7.2.11.2.1 2007/01/01 09:36:05 sebastian Exp $

pgsql

PostgreSQL Support	enabled
PostgreSQL(libpq) Version	8.2.3
Multibyte character support	enabled
SSL support	disabled
Active Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
pgsql.allow_persistent	On	On
pgsql.auto_reset_persistent	Off	Off
pgsql.ignore_notice	Off	Off
pgsql.log_notice	Off	Off
pgsql.max_links	Unlimited	Unlimited
pgsql.max_persistent	Unlimited	Unlimited

► Fig. 6.1 : PostgreSQL reconnu par PHP

6.2 Utiliser PDO

L'intérêt d'utiliser PDO réside dans le fait qu'avec PDO, quel que soit le SGBD que vous utilisez, les fonctions sont toujours les mêmes. Vous pouvez entièrement vous concentrer sur les requêtes SQL. De plus, en utilisant au maximum les normes SQL2 ou SQL3, vous supprimerez (presque) toute dépendance au SGBD utilisé et cela facilitera une migration vers un autre SGBD si cela s'avère nécessaire un jour. Les SGBD ayant tous leurs forces et leurs faiblesses, avec PDO vous pouvez vous connecter à plusieurs SGBD différents afin de consacrer chaque partie d'une application au SGBD qui lui soit le plus compétent.

Se connecter/déconnecter

Connexion

Se connecter à PDO peut paraître complexe, voire un peu déroutant pour quelqu'un ayant utilisé MySQL durant des années avec les fonctions natives (`mysql_*()`). En fait, il s'agit juste d'un petit (vraiment petit) temps d'adaptation et de comprendre ce qu'est le DSN (*Data Source Name*).

La connexion se fait toujours de la même manière quel que soit le SGBD auquel vous désirez accéder. Seuls les paramètres passés dans le constructeur varient. Pour tous les SGBD, le premier paramètre contient les DSN et des informations complémentaires.

Connexion MySQL

connexion_mysql.php

```
<?php
$host = 'localhost';
$user = 'root';
$pass = 'xxxxxxx';
$db = 'test';

$oPDOLink = new pdo("mysql:host=$host;dbname=$db", $user, $pass);
if(!$oPDOLink){
  echo 'Connexion au serveur MySQL impossible';
} else {
  echo 'Connexion au serveur MySQL effectuée avec succès';
}

$oPDOLink = NULL;

?>
```

Connexion PostgreSQL

connexion_pgsql.php

```
<?php

$host = 'localhost';
$user = 'postgres';
$pass = 'xxxxxxx';
```

```
$db = 'test';

$oPDOLink = new pdo("pgsql:host=$host user=$user password=$pass");
if(!$oPDOLink){
  echo 'Connexion au serveur PostgreSQL impossible';
} else {
  echo 'Connexion au serveur PostgreSQL effectuée avec succès';
}

/* Instructions */

$oPDOLink = NULL;

?>
```

Les points-virgules comme séparateurs

Vous verrez souvent dans les exemples, sur Internet ou dans des ouvrages spécialisés, les paramètres du DSN séparés par des espaces (comme indiqué dans le script *connexion_pgsql.php* en ce qui concerne PostgreSQL. Afin de respecter une norme au sein de PDO, il est préférable de toujours utiliser les points-virgules (`;`) comme séparateurs.

Déconnexion

La déconnexion est sans commentaires tellement celle-ci est simple.

```
$oPDOLink = NULL;
```

PDO et les exceptions

PDO est constitué de trois classes qui sont `PDO`, `PDOStatement` et `PDOException`. Cela signifie que PDO gère son propre système d'exceptions même si la classe `PDOException` hérite de la classe générique `Exception` de PHP.

Pour simplifier les exemples suivants, deux fichiers vont être créés et qui seront appelés par un `require_once()` dans les exemples. Un sera consacré à MySQL (*connect_mysql.inc.php*) :

connect_mysql.inc.php

```
<?php
$host = 'localhost';
$user = 'root';
```

```
$pass = 'xxxxxxx';
$db = 'test';

try {
  $oPDOLink = new pdo("mysql:host=$host;dbname=$db", $user, $pass);
} catch(Exception $e){
  print('Erreur '. $e->getMessage());
}

?>
```

Et l'autre à PostgreSQL (*connect_pgsql.inc.php*) :

connect_pgsql.inc.php

```
<?php
$host = 'localhost';
$user = 'postgres';
$pass = 'xxxxxxx';
$db = 'test';

try {
  $oPDOLink = new pdo("mysql:host=$host;dbname=$db", $user, $pass);
} catch(Exception $e){
  print('Erreur '. $e->getMessage());
}

?>
```

Manipulations des données

Exécuter des requêtes

Il y a deux méthodes pour exécuter des requêtes avec PDO qui sont `exec()` et `query()`. Cependant, il y a une différence entre ces deux fonctions qui ne sont pas anodines. `exec()` retourne un entier qui représente le nombre de lignes affectées par la requête. `query()` retourne soit un objet de type `PDOStatement`, soit un booléen selon les paramètres qui sont passés dans la fonction. Il existe une syntaxe possible pour `exec()` et quatre autres pour `query()` :

- `int exec(string statement);`
- `PDOStatement PDO::query(chaine statement);`

- `bool PDO::query(chaine statement, entier PDO::FETCH_COLUMN, entier col_no);`
- `bool PDO::query(chaine statement, entier PDO::FETCH_CLASS, chaine class, tableau ctoargs);`
- `bool PDO::query(chaine statement, entier PDO::FETCH_INTO, objet object);`

Les différences entre ces cinq syntaxes seront décrites un peu plus loin dans ce chapitre.

À titre d'exemple, créez dans une base de données nommée *test* une table nommée *users* qui contient les colonnes *use_id*, *use_login*, *use_password*, *use_prenom* et *use_nom* et insérez-y plusieurs lignes. Faites ceci avec MySQL et PostgreSQL :

table_logins.sql

```
CREATE TABLE logins(
    login char(15) NOT NULL,
    pass char(80) NOT NULL,
    prenom CHAR(30) NOT NULL,
    nom CHAR(30) NOT NULL,
    PRIMARY KEY(login)
);

INSERT INTO logins VALUES('ddrapeau', SHA1('xxxxxxxx'),
        'david', 'drapeau');
INSERT INTO logins VALUES('fsuire', SHA1('xxxxxxxx'),
        'frédéric', 'suire');
```

Lire dans une table

Vous allez maintenant afficher le contenu de la table *logins* avec MySQL, PostgreSQL, PDO MySQL et PDO PostgreSQL. Vous verrez ainsi la différence entre l'utilisation des drivers natifs de chaque SGBD et de celle de PDO (`mysql` et `pgsql`).

Drivers natifs

Utiliser les drivers natifs oblige à connaître les fonctions typiques au SGBD utilisé. Par exemple, vous utilisez MySQL depuis des années et on vous demande pour un projet d'utiliser PostgreSQL : vous êtes dans la nécessité d'étudier PostgreSQL et cela va vous prendre beaucoup de temps et d'énergie. Heureusement, les deux se ressemblent beaucoup mais ont chacun des particularités et des différences.

MySQL

lire_logins_mysql.php

```
<?php
$host = 'localhost';
$user = 'root';
$pass = 'xxxxxx';
$db = 'test';
$table = 'logins';

$link = mysql_connect($server, $user, $pass);
mysql_select_db($db);

$request = "SELECT login, prenom, nom FROM ". $table;
$result = mysql_query($request);

while($ligne = mysql_fetch_array($result)){
  echo $ligne['login'] .' '. ucfirst($ligne['prenom']) .' '.
  ➥ strtoupper($ligne['nom']) .'<br/>';
}

mysql_close($link);
?>
```

PostgreSQL

lire_logins_postgresql.php

```
<?php
$host = 'localhost';
$user = 'postgres';
$pass = 'xxxxxx';

$db = 'test';
$table = 'logins';

$link = pg_connect("host=$host dbname=$db user=$user password=$pass");

$request = "SELECT login, prenom, nom FROM ". $table;

$result = pg_query($request);

while($ligne = pg_fetch_array($result)){
```

```
  echo $ligne['login'] .' '. ucfirst($ligne['prenom']) .' '.
  ➥ strtoupper($ligne['nom']) .'<br/>';
}

pg_close($link);

?>
```

PDO

En utilisant PDO, vous n'avez plus besoin de modifier toutes les fonctions dans tous les scripts. Vous utilisez les fonctions PDO avec MySQL. Et si pour une mission, vous devez utiliser un autre SGBD que supporte PDO, vous n'aurez pas à apprendre les fonctions de l'autre SGBD et continuerez à utiliser les fonctions PDO. Donc, le temps et l'énergie économisés sont immenses.

pdo_mysql

lire_logins_pdo_mysql.php

```
<?php
// Définition des variables
$host = 'localhost';
$user = 'root';
$pass = 'xxxxxxx';

$dbname ='test';
$table = 'logins';

$pdc = new pdo("mysql:hostname=$host;dbname=$dbname", $user, $pass)
  or die("Erreur<br/>" . $e->getMessage());

$sql = "SELECT login, prenom, nom FROM ". $table;
$result = $pdc->query($sql);
while($ligne = $result->fetch()){
  echo $ligne['login'] .' '. ucfirst($ligne['prenom']) .' '.
  ➥ strtoupper($ligne['nom']) .'<br/>';
}

$pdc = NULL;
?>
```

pdo_pgsql

On vous demande de faire la même chose avec PostgreSQL depuis la migration de la base de données ? C'est très facile. La seule chose à modifier dans le script est la ligne suivante :

```
$pdc = new pdo("mysql:hostname=$host;dbname=$dbname", $user, $pass);
```

Et il vous suffit de placer dans les parenthèses :

```
"pgsql:host=$host;dbname=$dbname;user=$user;password=$pass"
```

Ce qui donne, pour le script complet :

lire_logins_pdo_pgsql.php

```
<?php
// Définition des variables
$host = 'localhost';
$user = 'postgres';
$pass = 'xxxxxx';

$dbname ='test';
$table = 'logins';

$pdc = new
➥ pdo("pgsql:host=$host;dbname=$dbname;user=$user;password=$pass")
  or die("Erreur<br/>" . $e->getMessage());

$sql = "SELECT login, prenom, nom FROM ". $table;
$result = $pdc->query($sql);
while($ligne = $result->fetch()){
  echo $ligne['login'] .' '. ucfirst($ligne['prenom']) .' '.
  ➥ strtoupper($ligne['nom']) .'<br/>';
}

$pdc = NULL;
?>
```

Bien sûr, il faut aussi remplacer les valeurs des variables si celles-ci sont différentes. C'est pour cette raison que dans beaucoup d'ouvrages, même de très bonne qualité, il vous est souvent conseillé de créer un fichier *defines.inc.php* dans lequel l'auteur vous conseille de définir toutes les constantes du projet ainsi que les paramètres de connexion au SGBD concerné et de définir dans ce même fichier de configuration le DSN pour l'utilisation de PDO. Ensuite, vous appelez ce fichier dans vos scripts de la façon suivante : `require_once('defines.inc.php');`.

Cette façon de faire est déconseillée car à chaque fois que vous allez appeler ce fichier, que ce soit par `include()`, `include_once()`, `require()` ou `require_once()`, toutes les constantes seront automatiquement définies et pas uniquement celles que vous utiliserez dans le fichier appelant. Donc, si pour un gros projet, vous avez dans ce fichier deux cents ou trois cents constantes, voire plus… Cela prend de la place en mémoire et en outre allonge largement le temps d'exécution dans le script. Il suffit d'utiliser les fonctions de `tracing` avec Xdebug et de sauvegarder les résultats dans un fichier de trace. C'est déconcertant ! Évidemment, ce serait trop fastidieux de définir les constantes dans tous les fichiers PHP. Imaginez un projet avec cent fichiers et tous utilisant l'accès aux bases de données ! S'il est nécessaire de changer le nom d'une base de données, cela signifie autant de fichiers à modifier. Ce n'est vraiment pas pratique.

La méthode ici mise en place est une classe qui fonctionne comme le fichier *defines.inc.php* à la différence que quand on appelle la classe par `include()`, ou l'une des trois autres méthodes, rien n'est défini par avance. Cela ne prend donc aucune place en mémoire. Il suffit ensuite d'instancier un objet de la classe, et seules les méthodes de classe utilisées prendront de la place en mémoire.

ClassDefine.php

```
<?php
/**
 * /include/ClassDefine.php
 *
 * Cette classe définit les variables qui sont considérées
 * comme statiques Certaines changent lorsqu'on change
 * d'environnement d'hébergement
 * (la plupart gardent la même valeur)
 */
class ClassDefine {
  // Définitions des variables de classe
  private $_server = 'localhost';  // Serveur SQL
  private $_user = 'postgres';  // Utilisateur SQL
  private $_password = 'xxxxxx';  // Password SQL
  private $_dbname = 'test';  // Nom de la BDD PostgreSQL
  private $_tablename = null;

  /** Définitions des méthodes **/

  // Constructeur
  public function __construct(){}
```

```
  // Destructeur
  public function __destruct(){}

  public function getSQLServer(){
    return $this->_server;
  }

  public function getSQLUser(){
  return $this->_user;
  }

  public function getSQLPassword(){
    return $this->_password;
  }

  public function getSQLDatabase(){
    return $this->_dbname;
  }

  public function getSQLTable($tablename){
    return $this->_tablename = $tablename;
  }
}
```

Et voici à présent un exemple de connexion pour un fichier PHP appelant cette classe et qui utilise PostgreSQL :

connexion_via_ClassDefine.php

```
<?php
require_once('ClassDefine.php');

$D = new ClassDefine();
$host = $D->getSQLServer();
$dbname = $D->getSQLDatabase();
$user = $D->getSQLUser();
$password = $D->getSQLPassword();

$oLink = new PDO("pgsql:host=$host dbname=$dbname
           user=$user password=$password");

/** Instructions **/
?>
```

Écrire dans une table

Vous venez de voir dans la section précédente *Lire dans une table* à quel point la même action dans deux SGBD différents est identique. Changer de SGBD demande de changer uniquement les paramètres de connexion dans le constructeur et leurs valeurs. Rien d'autre. Ce qui prouve bien que vous pouvez vous concentrer uniquement sur le code SQL. Du fait de cette simplicité, et du fait aussi que dans beaucoup de manuels, PDO est expliqué avec MySQL, tous les exemples qui vont suivre seront écrits avec PDO PostgreSQL. Cela vous fournira des exemples et des informations complémentaires.

Dans les exemples de lecture dans une table avec PDO, vous avez utilisé la fonction `query()` et non `exec()`. Pourquoi ? Parce que dans le cas d'une lecture où du contenu est retourné, les données se trouvant dans la table SQL répondent aux critères de la requête SQL `SELECT`.

Contrairement à un `SELECT`, lors d'une écriture dans la table SQL (`INSERT`), il n'y a pas de contenu de retourné mais seulement une valeur de retour pour signaler si tout s'est bien passé ou non. Il faudra donc utiliser la fonction `exec()` et non `query()`.

ecrire_logins_pdo_pgsql.php

```
<?php
// Définition des variables
$host = 'localhost';
$user = 'postgres';
$pass = 'xxxxxx';

$dbname ='test';
$table = 'logins';

$pdc = new pdo("pgsql:host=$host;dbname=$dbname;user
➥ =$user;password=$pass")
  or die("Erreur<br/>" . $e->getMessage());

$sql = "INSERT INTO logins VALUES('rroberto', 'passroberto', 'rinaldo',
➥ 'roberto')";
$result = $pdc->exec($sql);
if(!$result){
  echo 'Insertion dans la table échouée';
} else {
  echo 'Insertion dans la table effectuée avec succès';
}
```

```
$pdc = NULL;
?>
```

Remplacez `exec()` par `query()` pour l'écriture et le message "Insertion dans la table échouée" apparaîtra dans le navigateur.

Utiliser les exceptions

La version 5 de PHP intègre la gestion des exceptions par défaut. Si vous ne maîtrisez pas encore ce domaine, étudiez-le. Utiliser la fonction `die()`, tel que cela a été fait jusqu'à présent, n'est vraiment pas un gage de qualité, ni de sécurité. Cela démontre même une certaine incompétence tout comme l'usage de la fonction `md5()` alors qu'il a été prouvé que des collisions étaient possibles en utilisant cette fonction de cryptage. Mieux vaut utiliser la fonction `sha1()` tout aussi simple à utiliser et beaucoup plus sûre.

PDO contient trois classes dont deux déjà utilisées et une troisième que vous allez découvrir maintenant. Il s'agit de la classe `PDOException` qui hérite de la classe `Exception` de PHP. Analysons la classe `PDOException` :

La classe `PDOException` contient six méthodes que vous pouvez utiliser et qui sont les suivantes :

- `getMessage()` retourne le code erreur et le message d'erreur de PDO.
- `getCode()` retourne la valeur retournée par le code (c'est-à-dire la valeur numérique `0`).
- `getFile()` retourne le nom du fichier (chemin complet) où a été générée l'erreur.
- `getLine()` retourne le numéro de ligne du fichier où a été générée l'erreur.
- `getTrace()` retourne le type contenant le tracing (Array).
- `getTraceAsString()` retourne les mêmes informations que `getCode()`, `getFile()`, `getLine()` associées avec, en plus, la fonction PDO qui a généré l'erreur.

Pour ceux qui utilisent déjà la classe `Exception`, il est facile de remarquer qu'elles sont identiques. `PDOException` récupère les méthodes de la classe `Exception` et définit ses propres codes et messages d'erreurs.

Exécutez le script *exceptions_pdo.php* dans votre navigateur et regardez le résultat par vous-même. Vous comprendrez tout de suite.

exceptions_pdo.php

```
<?php
$host = 'localhost';
```

```
$user = 'postgres';
$pass = 'mauvais_password';
$db = 'test';

try {
  $pdc = new pdo("pgsql:host=$host;dbname=$db;user=$user;
            password=$pass");
  echo 'connexion réussie';
} catch(PDOException $e){
  echo 'getMessage : '. $e->getMessage() .'<br/>';
  echo 'getCode : '. $e->getCode() .'<br/>';
  echo 'getFile : '. $e->getFile() .'<br/>';
  echo 'getLine : '. $e->getLine() .'<br/>';
  echo 'getTrace : '. $e->getTrace() .'<br/>';
  echo 'getTraceAsString : <pre>'. $e->getTraceAsString()
  ➥ .'</pre><br/>';
}

$pdc = NULL;
?>
```

► Fig. 6.2 : Les exceptions PDO

6.3 Check-list

Dans ce chapitre, nous avons vu :

- ✔ l'installation PostgreSQL sur Windows et Linux ;
- ✔ les connexions via PDO ;
- ✔ comment écrire une classe de connexion ;
- ✔ comment utiliser des fonctions PDO ;
- ✔ comment utiliser les exceptions.

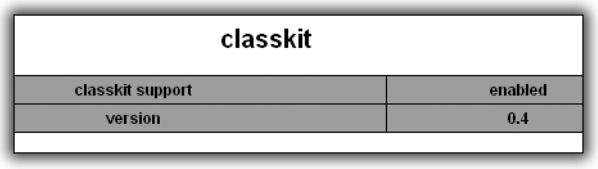
7

Dynamiser PHP5 avec Classkit

Grâce à PHP et à certaines extensions, comme les deux qui vont être expliquées dans ce chapitre, il est possible de supprimer une fonction dangereuse le temps de l'exécution d'un script et de supprimer ainsi une faille de sécurité. Cela, sans rien modifier en dur dans les fichiers *.php* et en quelques lignes de code seulement.

7.1 Classkit : dynamiser vos classes

Classkit est une extension PHP qui permet d'agir dynamiquement sur les classes. Les fonctionnalités sont limitées, cependant, vous pouvez tout de même ajouter dynamiquement une méthode dans une classe et même importer de nouvelles définitions de classe d'un fichier. Vous pouvez également renommer, copier, supprimer et redéfinir dynamiquement les méthodes d'une classe. Comme pour beaucoup d'extensions, pour installer Classkit sous Windows (avec XAMPP en tout cas), vous avez juste à décommenter la ligne dans le *php.ini* et à redémarrer Apache. Puis, vous ouvrez votre superprogramme qui contient la ligne de code magique `<?php phpinfo();?>` et là, vous devriez voir un cadre consacré à l'extension Classkit.



► Fig. 7.1 : L'extension Classkit activée

Classkit ne contient aucune directive à placer dans le *php.ini*. Tout se passe dans le code PHP. Voici un premier exemple dans lequel une méthode est renommée via Classkit. Tout d'abord, il est nécessaire de définir une classe. Vous pouvez utiliser la classe définie dans le script *classe_UneClasse.php*. Observez bien le nom de la méthode.

classe_UneClasse.php

```
<?php
class UneClasse {

  function __construct(){}

  function __destruct(){}

  public function uneMethode($arg1, $arg2,
         $defautlArg = 'Bonjour'){
  $salut = $defautlArg .' '. $arg1 .' '. $arg2;

  return $salut;
  }
}
?>
```

Vous pouvez voir que l'unique méthode définie dans cette classe se nomme `uneMethode()` et qu'elle comprend trois arguments dont le troisième est optionnel puisqu'une valeur par défaut lui a été affectée.

Les fonctions de Classkit

Renommer une méthode

Maintenant, nous allons utiliser la méthode `classkit_method_rename()` afin de renommer `uneMethode()` pour lui donner le nouveau nom suivant : `laNouvelleMethode()`.

- `bool classkit_method_rename(chaine $nom_classe,`

  `chaîne $nom_methode, chaîne $nouveau_nom_methode)`

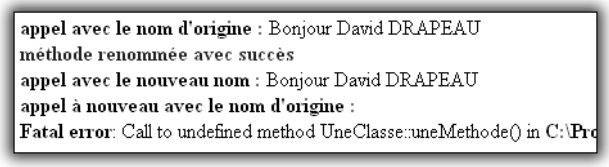
`classkit_methode_rename()` prend trois arguments. Le premier (`$nom_classe`) est le nom de la classe dans laquelle se trouve la méthode (`$nom_methode`) à renommer. Le deuxième argument (`$nom_methode`) est le nom de la méthode à renommer et le troisième argument (`$nouveau_nom_methode`) est le nouveau nom que l'on va justement donner à la méthode ciblée au deuxième argument. `classkit_method_rename()` retourne `true` en cas de succès et `false` en cas d'échec. Le script *renommer_methode.php* montre un exemple qui renomme dynamiquement une méthode :

ck_renommer_methode.php

```
<?php
// On inclut le fichier dans lequel on a défini la classe
// et la méthode que l'on va renommer
require_once('classe_UneClasse.php');

// On instancie un objet de la classe
$CCK = new UneClasse();

// Avant de renommer, on appelle avec le nom d'origine
echo '<b>appel avec le nom d\'origine :</b> ';
echo $CCK->uneMethode('David', 'DRAPEAU') .'<br />';

// Renommer la méthode uneMethode()
// qui va s'appeler maintenant laNouvelleMethode()
$renommer = classkit_method_rename('Uneclasse',
            'uneMethode', 'laNouvelleMethode');
if($renommer){
  echo '<b><font color="blue">
        méthode renommée avec succès
        </font></b><br/>';
```

```
} else {
  echo '<b><font color="red">
        impossible de renommer la méthode
        </font></b><br/>';
}

// Puis on utilise la méthode renommée précédemment
// en utilisant son nouveau nom
echo '<b>appel avec le nouveau nom : </b>';
echo $CCK->laNouvelleMethode('David', 'DRAPEAU') .'<br/>';

// Puis on appelle à nouveau par le nom d'origine
echo '<b>appel à nouveau avec le nom d\'origine :</b> ';
echo $CCK->uneMethode('David', 'DRAPEAU') .'<br />';
?>
```

Le résultat est affiché par le script *ck_renommer_methode.php*.



► Fig. 7.2 : Le script ck_renommer_methode.php exécuté dans le navigateur

Lors de l'appel de la fonction avec le nom d'origine (`uneMethode()`) et avant de l'avoir renommée, la fonction existe bien et fait ce qu'elle doit faire. Une fois renommée `laNouvelleMethode()`, si l'on appelle la même fonction par son nouveau nom, elle est toujours fonctionnelle. Par contre, si on l'appelle à nouveau avec son nom d'origine (`uneMethode()`), elle génère cette fois une erreur fatale, puisque pour PHP, elle n'existe plus sous son nom d'origine.

Supprimez les deux lignes qui génèrent l'erreur fatale ainsi que la ligne de commentaire qui y est associée :

```
// Puis on appelle à nouveau par le nom d'origine
echo '<b>appel à nouveau avec le nom d\'origine :</b> ';
echo $CCK->uneMethode('David', 'DRAPEAU') .'<br />';
```

Placez maintenant un lien, après la balise ?>, qui va pointer vers un fichier dans lequel vous allez appeler la méthode avec son nouveau nom. Le lien est :

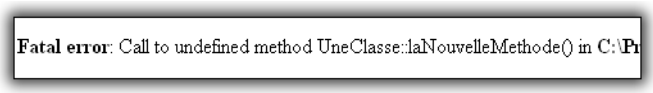
```
<a href="methode_renommee.php">Méthode renommée</a>
```

Le script *ck_methode_renommee.php* montre les trois lignes de codes à utiliser. Il serait très difficile de faire plus simple :

ck_methode_renommee.php

```
<?php
require_once('classe_UneClasse.php');

// On instancie un objet de la classe
$CCK = new UneClasse();

// Puis on utilise la méthode renommée précédemment
// en utilisant son nouveau nom
echo $CCK->laNouvelleMethode('David', 'DRAPEAU') .'<br/>';
?>
```

Et voici ce qui s'affiche dans le navigateur à l'exécution du script.



► Fig. 7.3 : Exemple d'affichage dans le navigateur

Alors, vous êtes surpris ?

⚠ Code dynamique, écriture dans fichier

Ce n'est pas parce que vous avez modifié dynamiquement le code que la modification s'est inscrite en dur en modifiant le fichier PHP. Pour cela, vous devez utiliser les fonctions de lecture et d'écriture dans les fichiers.

Si vous ouvrez le fichier PHP, vous verrez que le code n'a changé en rien. Et c'est pour cela que quand vous cliquez sur le lien pour exécuter le script *ck_methode_renommee.php* dans le navigateur en gardant le nouveau nom de la méthode (`laNouvelleMethode()`), PHP génère une erreur fatale. Pour exécuter à nouveau la méthode dans ce script, ainsi que dans tous les autres où la fonction n'a pas été renommée auparavant, vous devez utiliser le nom d'origine (`uneMethode()`) de la méthode.

Ajouter une méthode

Nous avons suffisamment insisté sur la différence entre le code dynamique et l'écriture dans un fichier. Voici à présent comment ajouter dynamiquement une fonction à la classe `UneClasse{}` :

ck_ajouter_methode.php

```
<?php
require_once('classe_UneClasse.php');
```

```
// On écrit dans le heredoc les instructions
// qui seront situées dans le corps de la fonction
$code =<<<codemethodeajoutee
\$sum = \$num1 + \$num2;
return \$sum;
codemethodeajoutee;


// On ajoute une nouvelle méthode dans la classe UneClasse
// avec un accès public (CLASSKIT_ACC_PUBLIC)
// la fonction contient deux arguments
classkit_method_add('UneClasse', 'methodeAjoutee',
   '$num1, $num2', $code, CLASSKIT_ACC_PUBLIC);

// création d'un objet Example
$oUC = new UneClasse();
echo $oUC->methodeAjoutee(12, 14);
?>
```

La fonction est bien ajoutée à la classe. Mais elle l'est dynamiquement. Si vous éditez à nouveau votre fichier PHP dans lequel est définie la classe, celle-ci n'a en rien changé. Quelles que soient les manipulations que vous effectuerez avec l'extension Classkit, rien ne sera modifié en dur (écrit) dans le fichier. La fonction utilisée pour ajouter dynamiquement du code est : `classkit_method_add()`.

- `bool classkit_method_add(chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux])`

Le premier paramètre (`$nom_classe`) définit la classe dans laquelle la nouvelle méthode va être ajoutée. Le nom de la nouvelle méthode est défini dans le deuxième paramètre (`$nom_methode`). Le troisième paramètre (`$args`) définit la liste des variables qui seront intégrées dans la nouvelle méthode. Le quatrième paramètre (`$code`) représente le corps de la fonction dans lequel les instructions sont définies et le cinquième paramètre définit le type d'accès à la méthode ajoutée. En ce qui concerne le cinquième paramètre, les valeurs pour le type d'accès sont définies par des constantes prédéfinies avec l'extension Classkit. Les constantes sont `CLASSKIT_ACC_PRIVATE` pour un accès privé, `CLASSKIT_ACC_PROTECTED` pour un accès protégé et `CLASSKIT_ACC_PUBLIC` pour un accès public.

Redéfinir une méthode

Vous pouvez, avec Classkit, redéfinir une méthode, c'est-à-dire lui donner de nouvelles instructions et même de nouveaux arguments. La fonction s'appelle

`classkit_method_redefine()` et elle effectue exactement la même action que `classkit_method_add()` à la différence que `classkit_method_add()` crée une nouvelle fonction alors que `classkit_method_redefine()` récupère une fonction existante.

- `bool classkit_method_redefine(chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux])`

Le rôle des arguments de la fonction `classkit_method_redefine()` est exactement le même que pour ceux de la fonction `classkit_method_add()`.

Supprimer une méthode

Avec Classkit, il est possible de supprimer une méthode, que celle-ci soit définie dynamiquement par `classkit_method_add()` ou non. Il suffit d'utiliser la fonction `classkit_method_remove()`. Cela peut s'avérer nécessaire pour sécuriser le code. Ainsi, vous pouvez supprimer dynamiquement une fonction dangereuse pour le script courant et pourtant indispensable pour d'autres scripts du projet.

- `bool classkit_method_remove(chaîne $nom_classe, chaîne $nom_methode)`

Le premier argument est le nom de la classe dans laquelle se trouve la méthode qui est elle-même définie en second paramètre de la fonction.

7.2 Passer la vitesse supérieure avec Runkit

Runkit est une extension PHP plus évoluée que Classkit. D'ailleurs, Runkit possède ses propres constantes prédéfinies ainsi que les constantes prédéfinies de Classkit. Runkit agit sur les classes en cours d'exécution mais pas seulement. Elle agit également sur les fonctions. Vous allez découvrir qu'avec Runkit, il est possible d'exploiter le code PHP de façon plus évoluée qu'avec Classkit.

Pour commencer, vous allez découvrir les directives de configuration de Runkit à placer dans le *php.ini*. Ensuite, nous aborderons en premier lieu les fonctions de Runkit similaires à celles étudiées dans la première partie avec Classkit. Enfin, l'étude portera sur les fonctions de Runkit qui permettent d'effectuer des manipulations dynamiques sur le code PHP et qui sont impossibles avec l'extension Classkit.

De même que pour Classkit, installer Runkit via XAMPP est très simple. Il suffit d'éditer le *php.ini* et de décommenter la ligne `;extension=php_runkit.dll` en supprimant le point-virgule (`;`) qui se trouve en début de ligne.

Directives de configuration

Runkit possède deux directives de configuration (à placer dans le fichier *php.ini*) qui sont `runkit.superglobal` qui a pour valeur par défaut une chaîne de caractères nulle et `runkit.internal_override` qui a la valeur numérique 0 par défaut.

`runkit.superglobal` permet de définir les variables qui vont être reconnues par le moteur PHP lui-même. En fait, elles ont la même portée que des variables superglobales bien plus connues qui sont `$_GET`, `$_POST`, `$_SERVER`, etc.

`runkit.internal_override = 1` permet de surcharger des fonctions internes à PHP en plus des fonctions définies par l'utilisateur. Par défaut, `runkit.internal_override = 0` et permet uniquement la surcharge de fonctions écrites par le programmeur et non les fonctions internes à PHP.

Fonctions similaires à Classkit

Renommer une méthode

La fonction qui renomme une méthode (fonction située dans une classe) s'appelle `runkit_method_rename()`. Son fonctionnement est identique à `classkit_method_rename()` et ses argument sont les mêmes et ont les mêmes rôles.

- `bool runkit_method_rename(chaîne $nom_classe, chaîne $nom_methode, chaîne $nouveau_nom)`

Il ne faut pas confondre `runkit_method_rename()` qui agit sur une fonction définie dans une classe avec `runkit_function_rename()` qui agit sur une fonction définie hors de toute classe et qui sera étudiée un peu plus loin dans ce même chapitre. Le script *rk_renommer_methode.php* montre un exemple très simple de l'utilisation de `runkit_function_rename()` :

rk_renommer_methode.php

```
<?php
// On inclut le fichier dans lequel on a défini la classe
// et la méthode que l'on va renommer
require_once('classe_UneClasse.php');

// On instancie un objet de la classe
$CCK = new UneClasse();

// Avant de renommer, on appelle avec le nom d'origine
echo '<b>appel avec le nom d\'origine :</b> ';
```

```
echo $CCK->uneMethode('David', 'DRAPEAU') .'<br />';

// On renomme la méthode uneMethode()
// qui va s'appeler maintenant laNouvelleMethode()
$renommer = runkit_method_rename('Uneclasse',
            'uneMethode', 'laNouvelleMethode');
if($renommer){
  echo '<b><font color="blue">
        méthode renommée avec succès
        </font></b><br/>';
} else {
  echo '<b><font color="red">
        impossible de renommer la méthode
        </font></b><br/>';
}

// Puis on utilise la méthode renommée précédemment
// en utilisant son nouveau nom
echo '<b>appel avec le nouveau nom : </b>';
echo $CCK->laNouvelleMethode('David', 'DRAPEAU') .'<br/>';

// Puis on appelle à nouveau par le nom d'origine
echo '<b>appel à nouveau avec le nom d\'origine :</b> ';
echo $CCK->uneMethode('David', 'DRAPEAU') .'<br />';
?>
```

Ajouter une méthode

La fonction qui ajoute une méthode (fonction située dans une classe) s'appelle `runkit_method_add()`. Son fonctionnement est identique à `classkit_method_add()` et ses argument sont les mêmes et ont les mêmes rôles.

- `bool runkit_method_add(chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux])`

Il ne faut pas confondre `runkit_method_add()` qui agit sur une fonction définie dans une classe avec `runkit_function_add()` qui agit sur une fonction définie hors de toute classe et qui sera étudiée ultérieurement dans ce même chapitre. Le script *rk_ajouter_methode .php* montre un exemple très simple de l'utilisation de `runkit_add_method()` :

rk_ajouter_methode.php

```
<?php
require_once('classe_UneClasse.php');

// On écrit dans le heredoc les instructions
```

```
// qui seront situées dans le corps de la fonction
$code =<<<codemethodeajoutee
\$sum = \$num1 + \$num2;
return \$sum;
codemethodeajoutee;

// On ajoute une nouvelle méthode dans la classe UneClasse
// avec un accès public (CLASSKIT_ACC_PUBLIC)
// la fonction contient deux arguments
runkit_method_add('UneClasse', 'methode_ajoutee',
     '$num1, $num2', $code, CLASSKIT_ACC_PUBLIC);

// création d'un objet Example
$oCCK = new UneClasse();
echo $oCCK->methode_ajoutee(12, 14);
?>
```

⚠ Majuscule vs minuscule

Si vous analysez le script *ck_ajouter_methode.php*, vous vous rendrez compte que le nom de la fonction ajoutée est `methodeAjoutee()`. Ce nom de méthode avec Runkit générera un message d'erreur, lors de son appel, comme quoi PHP ne trouve pas la fonction demandée et dira que celle-ci n'existe pas. En effet, Runkit n'accepte que les noms de fonctions avec lettres minuscules et underscore (_). C'est pour cette raison que dans le script *rk_ajouter_methode.php*, le nom de la fonction à ajouter est `methode_ajoutee()` au lieu de `methodeAjoutee()`.

Ce bug n'existe pas avec la fonction `runkit_method_rename()` puisqu'il a été possible de renommer la fonction `uneMethode()` par le nom `laNouvelleMethode()` dans le script *rk_renommer_methode.php*.

Redéfinir une méthode

ⓘ Chapitre 1

Voici, comme promis, le bon login et le bon mot de passe (*password*) du cas pratique du chapitre 1 au sujet de la compression :

```
$bonlogin = sha1('yeahhh');
$bonpassword = sha1('tropfort');
```

La fonction de l'extension Runkit qui redéfinit une méthode située dans une classe s'appelle `runkit_method_redefine()`. Son fonctionnement est identique à la fonction de l'extension Classkit `classkit_method_redefine()` et ses argument sont les mêmes et ont les mêmes rôles.

- `bool runkit_method_redefine(chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux])`

Il ne faut pas confondre `runkit_method_redefine()` qui agit sur une fonction définie dans une classe avec `runkit_function_redefine()` qui agit sur une fonction définie hors de toute classe et qui sera étudiée ultérieurement dans ce même chapitre. Le script *rk_modifier_methode.php* montre un exemple concret de ce que fait la fonction `runkit_method_redefine()` :

rk_modifier_methode.php

```
<?php
require_once('classe_UneClasse.php');

// On écrit dans le heredoc les instructions
// qui seront situées dans le corps de la fonction
$code =<<<codemethodeajoutee
\$sum = \$num1 + \$num2;
return \$sum;
codemethodeajoutee;


// On ajoute une nouvelle méthode dans la classe UneClasse
// avec un accès public (RUNKIT_ACC_PUBLIC)
// la fonction contient deux arguments
runkit_method_add('UneClasse', 'une_somme',
          '$num1, $num2', $code, RUNKIT_ACC_PUBLIC);

// instanciation de la classe UneClasse
$oUC = new UneClasse();
echo 'La somme est : '. $oUC->une_somme(12, 14);

// On redéfinit la fonction une_somme
// qui va s'appeler un_produit
// et qui va calculer le produit
// des deux nombres $num1, $num2.

// On renomme la méthode une_somme
// qui va s'appeler maintenant un_produit
$renommer = runkit_method_rename('Uneclasse',
            'une_somme', 'un_produit');
if($renommer){
  echo '<br/><b><font color="blue">
        méthode renommée avec succès
        </font></b><br/>';
} else {
  echo '<br/><b><font color="red">
        impossible de renommer la méthode
```

```
        </font></b><br/>';
}

// On écrit dans le heredoc les instructions
// qui seront situées dans le corps de la fonction
$code =<<<codemethodeajoutee
\$sum = \$num1 * \$num2;
return \$sum;
codemethodeajoutee;

// Puis on redéfinit la méthode
$redefinir = runkit_method_redefine('UneClasse',
             'un_produit', '$num1, $num2',
              $code, RUNKIT_ACC_PUBLIC);

if($redefinir){
  echo '<br/><b><font color="blue">
        méthode redéfinie avec succès
        </font></b><br/>';
} else {
  echo '<br/><b><font color="red">
        impossible de redéfinir la méthode
        </font></b><br/>';
}

// On affiche le résultat du produit
echo 'Le produit est : '. $oUC->un_produit(12, 14);

?>
```

CLASSKIT_ACC_* vs RUNKIT_ACC_*

Pour définir le type d'accès à la méthode, avec Classkit, il y a `CLASSKIT_ACC_PRIVATE`, `CLASSKIT_ACC_PROTECTED` et `CLASSKIT_ACC_PUBLIC`. Avec Runkit, il y a `RUNKIT_ACC_PRIVATE`, `RUNKIT_ACC_PROTECTED` et `RUNKIT_ACC_PUBLIC`.

Runkit a également la possibilité d'accepter les constantes `CLASSKIT_ACC_*` à la place des constantes `RUNKIT_ACC_*`, ce qui peut faire gagner énormément de temps le jour où, par exemple, vous récupérez un projet qui contient beaucoup de code dynamique écrit avec Classkit et que vous comptez l'améliorer en utilisant Runkit.

Supprimer une méthode

La fonction qui supprime dynamiquement une méthode (fonction située dans une classe) s'appelle `runkit_method_remove()`. Son fonctionnement est identique à `classkit_method_remove()` et ses argument sont les mêmes et ont les mêmes rôles.

- `bool runkit_method_remove(chaîne $nom_classe, chaîne $nom_methode)`

Il ne faut pas confondre `runkit_method_remove()` qui agit sur une fonction définie dans une classe avec `runkit_function_remove()` qui agit sur une fonction définie hors de toute classe et qui sera étudiée ultérieurement dans ce même chapitre.

rk_supprimer_methode.php

```
<?php
class UneClasse {
  public function methodeMortelle(){
    return 'Je veux vivre!!!';
  }
}

$oUC = new UneClasse();
echo 'avant suppression dynamique de la méthode : ';
echo $oUC->methodeMortelle();
echo '<br/><br/>';

runkit_method_remove('UneClasse', 'methodeMortelle');
echo 'après suppression dynamique de la méthode : ';
echo $oUC->methodeMortelle();

?>
```

Dépasser les possibilités de Classkit

Avec Classkit et Runkit, vous avez jusqu'à présent compris comment faire pour manipuler des fonctions définies dans des classes. PHP permet également de définir des fonctions directement dans des scripts sans pour autant les placer dans une classe. Reprenez les quatre fonctions précédentes et remplacez dans le nom de la fonction le mot `method` par le mot `function` et vous aurez mémorisé en deux secondes les noms des nouvelles fonctions que vous allez voir dans cette seconde partie. Vous l'avez compris, les quatre fonctions de Runkit qui vont suivre sont : `runkit_function_add()`, `runkit_function_redefine()`, `runkit_function_rename()` et, pour terminer cette partie, `runkit_function_remove()`. Supprimez ensuite le premier paramètre (le nom de la classe) et le dernier (le type d'accès en ce qui concerne la fonction d'ajout et de redéfinition) et vous aurez mémorisé, encore en deux secondes, la syntaxe de chacune de ces nouvelles fonctions.

Ajouter une fonction

La fonction `runkit_function_add()` permet d'ajouter une nouvelle fonction dans le script courant.

- `bool runkit_function_add(chaîne $nom_fonction, chaîne $liste_arguments, chaîne $code)`

Il s'agit bien de fonction et non de méthode. Autrement dit, celle-ci (la fonction) ne sera pas définie dans une classe. Le script *rk_php_statique.php* (premier exemple) montre l'utilisation d'une fonction telle que cela se fait en dur dans le script tandis que le script *rk_php_dynamique.php* (second exemple) montre l'utilisation de la même fonction générée dynamiquement.

Premier exemple

rk_php_statique.php

```
<?php
function somme($a, $b){
 return $à + $b;
}

echo somme(3, 7);
?>
```

Second exemple

rk_php_dynamique.php

```
<?php
runkit_function_add('somme', '$a, $b', 'return $a + $b;');

echo somme(12, 14);
?>
```

Modifier une fonction

La fonction `runkit_function_redefine()` permet de redéfinir une fonction située dans le script courant.

- `bool runkit_function_redefine(chaîne $nom_fonction, chaîne $liste_arguments, chaîne $code)`

Renommer une fonction

La fonction `runkit_function_rename()` permet de renommer une fonction située dans le script courant.

- `bool runkit_function_rename(chaîne $nom_fonction, chaîne $nouveau_nom)`

Supprimer une fonction

La fonction `runkit_function_remove()` permet de supprimer une fonction située dans le script courant.

- `bool runkit_function_remove(chaîne $nom_fonction)`

Jouer avec les constantes

Tout comme pour les fonctions et les méthodes, Runkit gère aussi les constantes. Vous pouvez créer une constante, la modifier et/ou la supprimer.

- `bool runkit_constant_add(chaîne $nom_constante, mixte $valeur)`
- `bool runkit_constant_redefine(chaîne $nom_constante, mixte $nouvelle_valeur)`
- `bool runkit_constant_remove(string $nom_constante)`

La différence entre `runkit_constant_add()` et `runkit_constant_redefine()` est que la première crée une constante qui n'existe pas avant l'appel de la fonction et que la seconde fonction récupère une constante existante afin de lui affecter une nouvelle valeur.

Le script *rk_constantes.php* montre un exemple des fonctions `runkit_constant_add()`, `runkit_constant_redefine()` et `runkit_constant_remove()`.

rk_constantes

```
<?php
runkit_constant_add('NOMBRE', 3);
echo 'La constante créée a pour valeur '. NOMBRE;
// affiche: la constante créée a pour valeur 3

runkit_constant_redefine('NOMBRE', 12);
echo '<br/>La constante a été redéfinie
          avec la valeur '. NOMBRE;
// affiche: la constante a été redéfinie avec la valeur 12

runkit_constant_remove('NOMBRE');
echo '<br/>La constante a été supprimée.';
```

```
echo ' Elle a pour valeur '. NOMBRE;
// affiche: La constante a été supprimée.
// Elle a pour valeur NOMBRE

?>
```

Le premier `echo` affiche bien la valeur de la constante définie avec `runkit_constant_add()` ainsi que le deuxième `echo` qui affiche tout autant la nouvelle valeur de la constante redéfinie avec `runkit_function_redefine()`. Cependant, le troisième `echo` affiche `NOMBRE` parce que la constante `NOMBRE` a été supprimée grâce à la fonction `runkit_constant_remove()` et `echo` considère donc `NOMBRE` comme une chaîne, bien que celle-ci ne soit pas placée entre guillemets, et non comme une constante.

Agir sur les classes

Deux fonctions permettent d'agir sur les classes et comme toujours, leur nom est très explicite. La première permet de créer un héritage et se nomme `runkit_class_adopt()` et la seconde fait l'action inverse et s'appelle `runkit_class_emancipate()`. Voici un exemple hors informatique. Vous êtes une personne adulte avec une certaine situation stable et vous désirez adopter un enfant ; celui-ci héritera de vous alors qu'à l'inverse vous êtes un père (ou une mère) et vous ne supportez plus votre enfant et vous ne voulez pas qu'il hérite de vos richesses : alors vous l'émancipez. C'est exactement la même chose avec les deux fonctions citées précédemment. Le script *rk_adopter_emanciper.php* montre un exemple très simple et très concret de l'adoption et de l'émancipation dynamique des classes :

- `bool runkit_class_adopt(chaîne $nom_classe_enfant,`

  `chaîne $nom_classe_parent)`
- `bool runkit_class_emancipate(chaîne $nom_classe_parent)`

rk_adopter_emanciper.php

```
<?php
class ClasseAdulte {
..public function messageImportant(){
....return "J'ai adopté un enfant";
..}
}

class ClasseEnfant {
..
```

```
}

$oEnfant = new ClasseEnfant();
//echo $oEnfant->messageImportant();

runkit_class_adopt('ClasseEnfant', 'ClasseAdulte');
echo $oEnfant->messageImportant();


runkit_class_emancipate('ClasseParent');
echo $oEnfant->messageImportant();
?>
```

À la ligne 13, il y a une instruction mise en commentaire. Cette ligne appelle la fonction `messageImportant()` de la classe `ClasseAdulte` avant que celle-ci ait adopté la classe `ClasseEnfant` : cela génère une erreur fatale et stoppe le script *adopter_emanciper.php* aussitôt après cette instruction lorsque cette ligne est décommentée.

7.3 Conclusion

Vous voici maintenant armé pour utiliser correctement Runkit afin de générer du code dynamique et faire évoluer considérablement vos connaissances et votre façon de programmer. La seule limite au codage dynamique en PHP est votre imagination. Le code PHP, qui n'a plus rien à envier à Java ou à C++ en termes de possibilités, est devenu tellement puissant que les limites seront celles que vous vous imposerez (consciemment ou inconsciemment).

7.4 Check-list

Avec Classkit, nous avons appris à :

- ✔ ajouter des méthodes de manière dynamique ;
- ✔ les renommer ;
- ✔ les redéfinir ;
- ✔ les supprimer.

Avec Runkit, nous avons appris à :

- ✔ effectuer le même travail qu'avec Classkit ;
- ✔ manipuler les fonctions ;
- ✔ créer/détruire des héritages de classes.

8

Dynamiser le code avec Reflection

Cet article fait exception à la règle du livre, car il ne s'agit pas là d'une extension mais d'une API (*Application Programming Interface*) faisant partie de la version objet de PHP5. Cette API a pour nom de baptême `Reflection`, aussi appelée introspection. Utilisée correctement, elle peut vous permettre de vous documenter sur une librairie PHP, sur des fonctions et des extensions internes ou externes à PHP.

8.1 Les classes de Reflection

L'API de réflexion est principalement utilisée pour la POO (Programmation orientée objet). Les classes sont les suivantes :

- `class Reflection{}`
- `interface Reflector{}`
- `class ReflectionException extends Exception{}`
- `class ReflectionFunction implements Reflector{}`
- `class ReflectionParameter implements Reflector{}`
- `class ReflectionMethod extends ReflectionFunction{}`
- `class ReflectionClass implements Reflector{}`
- `class ReflectionObject extends ReflectionClass{}`
- `class ReflectionProperty implements Reflector{}`
- `class ReflectionExtension implements Reflector{}`

8.2 Reflection et ReflectionClass : disséquer une classe en une ligne de code

Comme nous venons de l'écrire, utiliser la réflexion est un excellent moyen pour étudier une classe PHP dans sa totalité. `Reflection` et `ReflectionClass` permettent de disséquer une classe et de savoir très précisément comment elle est constituée. La fonction qui récupère toutes les données de la classe et les affiche dans le navigateur se nomme `export()`.

- `Reflection::export(mixte $nom_classe, bool $retour)`

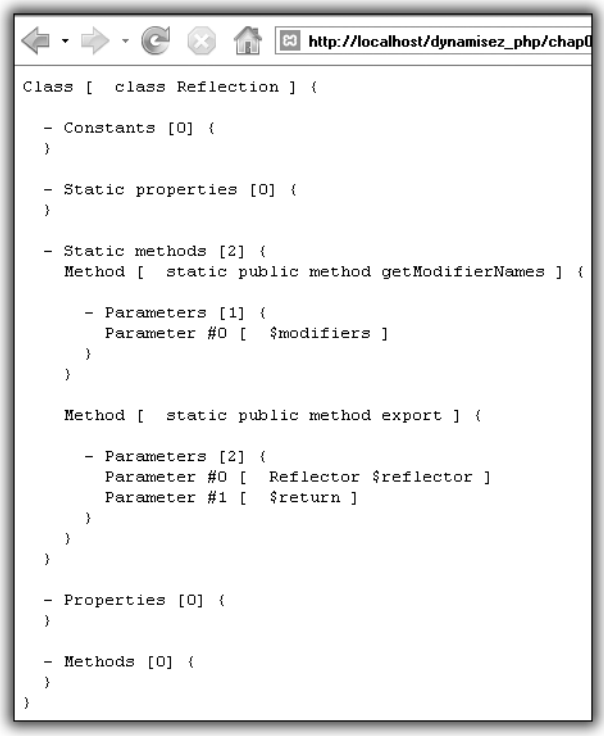
En cas de succès, `export()` retourne une chaîne, ou plutôt un tableau, que vous pouvez très facilement formater en utilisant les balises `<pre></pre>`. On pourrait considérer que `export()` pour la classe `Reflection{}` est l'équivalent de `print_r()` pour les tableaux.

methode_export.php

```
<?php
// Choisir une classe
$classname = 'Reflection';

echo '<pre>';
// Voici LA ligne qui affiche
// toutes les informations de la classe
```

```
Reflection::export(new ReflectionClass($classname));
echo '</pre>';
?>
```

Le résultat de l'exécution dans un navigateur de *methode_export.php* est visible sur la figure suivante :



► Fig. 8.1 : La classe Reflection disséquée

Voici quelques explications sur le résultat affiché dans le navigateur. La première ligne `Class[classe Reflection]` indique le nom de la classe. La ligne `Constants[0]` indique que la classe `Reflection` ne contient aucune constante dans son corps. C'est la même chose pour les propriétés statiques : `Static properties[0]`. En ce qui concerne les méthodes statiques, la classe `Reflection` en possède deux qui ont toutes deux un accès public (`public`). La quantité de paramètres d'une fonction est signalée par la ligne `Parameters[0]`. Et la description du premier argument est signalée par la ligne `Parameter #0 […]`. La suite est tout aussi simple à comprendre.

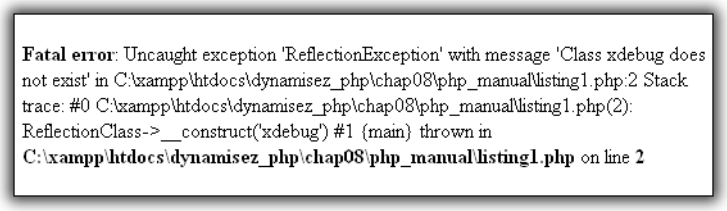
Si la classe n'existe pas, un message d'erreur fatale est affiché dans le navigateur et le script est stoppé immédiatement après la génération de l'erreur. Le script

err_appel_xdebug.php montre un exemple avec Xdebug appelée par `Reflection` et qui génère un message d'erreur puisque Xdebug est le nom d'une extension mais ce n'est pas le nom d'une classe reconnue par PHP.

err_appel_xdebug.php

```
<?php
echo '<pre>';
Reflection::export(new ReflectionClass('xdebug'));
echo '</pre>';
?>
```

Ce qui a pour résultat l'affichage dans le navigateur du message suivant :



▶ Fig. 8.2 : Erreur : Xdebug n'est pas programmée en POO.

Vous voulez tester pour une autre classe ? Reprenez le script *methode_export.php* et modifiez la valeur de la variable `$classname = 'Reflection';` par `$classname = 'ReflectionClass';` et vous connaîtrez toutes les fonctions de la classe `ReflectionClass{}` que vous avez à votre disposition. Observez le script *reflexion_sur_ReflectionClass.php* :

reflexion_sur_ReflectionClass.php

```
<?php
// Choisir une classe
$classname = 'ReflectionClass';

echo '<pre>';
// Voici LA ligne qui affiche
// toutes les informations de la classe
Reflection::export(new ReflectionClass($classname));
echo '</pre>';
?>
```

```
Class [  class ReflectionClass implements Reflector ] {

  - Constants [3] {
    Constant [ integer IS_IMPLICIT_ABSTRACT ] { 16 }
    Constant [ integer IS_EXPLICIT_ABSTRACT ] { 32 }
    Constant [ integer IS_FINAL ] { 64 }
  }

  - Static properties [0] {
  }

  - Static methods [1] {
    Method [  static public method export ] {

      - Parameters [2] {
        Parameter #0 [  $argument ]
        Parameter #1 [  $return ]
      }
    }
  }

  - Properties [1] {
    Property [  public $name ]
  }

  - Methods [40] {
    Method [  final private method __clone ] {
    }

    Method [  public method __construct ] {

      - Parameters [1] {
        Parameter #0 [  $argument ]
      }
    }

    Method [  public method __toString ] {
    }

    Method [  public method getName ] {
    }

    Method [  public method isInternal ] {
    }

    Method [  public method isUserDefined ] {
```

Fig. 8.3 : Un extrait de ReflectionClass par réflexion

Quand vous exécutez le script dans votre navigateur, vous avez une description très précise de ce que vous pouvez utiliser comme fonctions et comme attributs. Vous savez déjà que la classe `ReflectionClass{}` possède trois constantes de type entier (`int`), une méthode statique nommée `export()` (celle que vous avez utilisée dans le script précédent) et quarante méthodes dont seules celles qui ont un accès public (`public`) sont utilisables. Les méthodes d'accès protégé (`protected`) ne sont utilisables que pour les classes qui sont des classes étendues de la classe `ReflectionClass{}` et les méthodes, comme la méthode `__clone()`, qui ont un accès privé (`private`) ne sont accessibles que par les méthodes incluses dans la même classe. Quant aux méthodes `__construct()` et

__toString(), elles sont également appelées, en PHP, des méthodes magiques et sont utilisées implicitement. Autrement dit, vous n'avez pas à vous en occuper.

8.3 Tout connaître d'une fonction

Une fois la réflexion faite sur la classe, vous pouvez pousser l'expérience encore plus avant en effectuant la réflexion sur chaque fonction de la classe. La classe qui s'en charge est ReflectionFunction{}. Et si vous effectuez une réflexion sur cette classe (script *reflexion_sur_ReflectionFunction.php*), un extrait du résultat est visible à la figure suivante :

reflexion_sur_ReflectionFunction.php

```
<?php
echo '<pre>';
Reflection::export(new ReflectionClass('ReflectionFunction'));
echo '</pre>';
?>
```

```
Class [  class ReflectionFunction extends ReflectionFunctionAbstract implements Reflector ] {

  - Constants [1] {
    Constant [ integer IS_DEPRECATED ] { 262144 }
  }

  - Static properties [0] {
  }

  - Static methods [1] {
    Method [  static public method export ] {

      - Parameters [2] {
        Parameter #0 [  $name ]
```

► Fig. 8.4 : Un extrait de ReflectionFunction par réflexion

La première ligne indique que la classe ReflectionFunction{} est une classe étendue de la classe ReflectionFunctionAbstract{} et qu'elle implémente aussi l'interface Reflector{}. Vous pouvez aussi voir dans les méthodes statiques que la fonction export() s'y trouve à nouveau. Le script *fonction_calculerSomme.php* utilise la méthode export() de la classe ReflectionFunction{} qui retourne sous forme de chaîne tous les renseignements de la fonction calculerSomme() :

fonction_calculerSomme.php

```
<?php
```

```
// On définit une fonction toute simple

/**
 * Ici la description de la fonction qui sera
 * récupérable par getDocComment() de la classe
 * ReflectionFunction
 *
 * @param int $arg1
 * @param int $arg2
 * @return unknown
 */
function calculerSomme(int $arg1, int $arg2){
  return $arg1 + $arg2;
}

echo '<pre>';
// LA ligne qui dissèque la fonction
echo ReflectionFunction::export(calculerSomme);
echo '</pre>';
?>
```

Le résultat est visible sur la figure suivante.

```
/**
 * Ici la description de la fonction qui sera
 * récupérable par getDocComment() de la classe
 * ReflectionFunction
 *
 * @param int $arg1
 * @param int $arg2
 * @return unknown
 */
Function [   function calculerSomme ] {
  @@ C:\Program Files\xampp\htdocs\dynamisez_php\chap08\dissequer_fonction.php 13 - 15

  - Parameters [2] {
    Parameter #0 [  int $arg1 ]
    Parameter #1 [  int $arg2 ]
  }
}
```

► Fig. 8.5 : Réflexion de la fonction calculerSomme

Comme d'habitude, la première ligne donne le nom de la fonction. La deuxième ligne donne le chemin complet et le nom du fichier dans lequel est définie la fonction. La deuxième ligne donne également d'autres informations : le numéro de ligne où commence la fonction et le numéro de ligne où elle finit. Dans cet exemple, il est indiqué 13 – 15. Si vous analysez le script *fonction_calculerSomme.php*, il est aisé de se rendre compte que la définition de la fonction commence à la treizième ligne et se termine à la quinzième ligne.

Voici les fonctions de la classe `ReflectionFunction{}` :

- `chaîne getName()` retourne le nom de la fonction.
- `bool isInternal()` retourne `true` si une fonction est interne à PHP ou `false` si elle l'a été par le programmeur.
- `bool isUserDefined()` effectue l'inverse de `isInternal()` et retourne `true` si une fonction a été définie par le programmeur ou `false` si elle est interne à PHP.
- `chaîne getFileName()` retourne le nom du fichier dans lequel se trouve la définition de la fonction. Si la fonction est interne à PHP, elle n'est donc pas définie dans un fichier, `getFileName()` retourne une chaîne vide.
- `entier getStartLine()` retourne le numéro de ligne du début de la définition de la fonction.
- `entier getEndLine()` retourne la numéro de ligne de la fin de la définition de la fonction.
- `chaîne getDocComment()` retourne le commentaire associé à la fonction à condition que celui-ci (le commentaire) soit formaté de la façon suivante :

```
/**
 * description de la fonction
 * @nom_predefini
 */
```

  `nom_predefini` peut avoir pour valeur : `author`, `copyright`, `name`, `param`, etc.

- `tableau getStaticVariables()` retourne sous forme de tableau les variables statiques définies dans la fonction.
- `entier getNumberOfParameters()` retourne le nombre de paramètres de la fonction (les variables situées entre les parenthèses de la fonction placée en réflexion).
- `entier getNumberOfRequiredParameters()` retourne le nombre de paramètres obligatoires lors de l'appel de la fonction. Si une fonction comporte trois paramètres dont un optionnel alors `getNumberOfParameters()` retournera la valeur `3` et `getNumberOfRequiredParameters()` retournera la valeur `2`.

Le script *dissequer_function.php* montre un exemple d'utilisation de chacune des fonctions que nous venons de décrire :

dissequer_fonction.php

```
<?php
// Utilisation des méthodes de la classe ReflectionFunction

$F = new ReflectionFunction('printf');
```

```
echo $F->isInternal()
 ? 'printf est une fonction interne à PHP'
 : 'printf est une fonction créée par un programmeur';
echo '<br/>';

/**
 * Ceci est une fonction qui n'est pas interne à PHP
 *
 * @author David DRAPEAU
 *
 * @param int a
 * @param int b
 *
 * @return int sum
 *
 */
function calculerSomme($a, $b, $msg = 'la somme est '){
  $c = $a + $b;
  return $msg . $c;
}

$nomFonction = 'calculerSomme';

$F = new ReflectionFunction($nomFonction);
echo $F->isInternal()
 ? 'testFonction est une fonction interne à PHP'
 : 'testFonction a été créée par un programmeur';
echo '<br/>';
// Restons toujours sur la fonction printf
//

// nom de la fonction
$nom = $F->getName();
echo 'le nom de la fonction : '. $nom .'<br/>';

// nom du fichier dans lequel se trouve
// la définition de la fonction
$filename = $F->getFileName();
echo 'le nom du fichier où est écrite la fonction : '.
      $filename .'<br/>';

// ligne de début
$ligneDeb = $F->getStartLine();
echo 'à quel endroit du fichier démarre
      la fonction : ligne '. $ligneDeb .'<br/>';

// ligne de fin
```

```
$ligneFin = $F->getEndLine();
echo 'à quel endroit du fichier se termine
      la fonction : ligne '. $ligneFin .'<br/>';

// les lignes de commentaires qui documentent la fonction
$comment = $F->getDocComment();
echo 'commentaires sur la fonction :<br/>';
echo '<pre>';
echo $comment;
echo '</pre>';

// valeurs de retour
$returns = $F->returnsReference();
echo $returns .'<br/>';

// paramètres
$param = $F->getParameters();
echo $param .'<br/>';

// nombre de paramètres
$nbParams = $F->getNumberOfParameters();
echo $nbParams .'<br/>';

// nombre de paramètres requis
$nbRqParams = $F->getNumberOfRequiredParameters();
echo $nbRqParams .'<br/>';

echo 'utilisation de export()<br/>';
echo '<pre>';
ReflectionFunction::export('calculerSomme');
echo '</pre>';

?>
```

8.4 ReflectionMethod : on entre dans la classe

La différence entre la classe `ReflectionFunction{}` et la classe `ReflectionMethod{}` est que la première concerne les fonctions écrites hors de toutes classes alors que la seconde concerne les méthodes définies à l'intérieur d'une classe. Il semble donc logique que les informations qu'il est possible d'y récupérer soient différentes. Par exemple, s'agissant d'une méthode, vous pouvez récupérer le nom de la classe dans laquelle elle est définie. C'est une information qu'il est impossible de connaître d'une fonction puisque celle-ci n'appartient à aucune classe. Une autre évidence est qu'une méthode est aussi une fonction dans le sens où, tout comme une fonction, la méthode possède un nom, est

définie par l'utilisateur ou est interne à PHP, etc. La classe `ReflectionMethod{}` possède donc des méthodes héritées de la classe `ReflectionFunction{}` qui sont les suivantes : `getName()`, `isInternal()`, `isUserDefined()`, `getFileName()`, `getStartLine()`, `getEndLine()`, `getDocComment()`, `getStaticVariables()`, `returnsReference()`, `getParameters()`, `getNumberOfParameters()`, `getNumberOfRequiredParameters()`.

Parmi les nouvelles méthodes, vous avez les suivantes :

- `public __construct(mixte $nom_classe, chaîne $nom_methode)` qui est automatiquement utilisée lors de l'instanciation de l'objet de la classe.
- `public chaîne __toString()` et également une méthode magique (comme `__construct()`) dont vous n'avez pas à vous préoccuper. Elle jouera son rôle le moment venu.
- `public static chaîne export(mixte $nom_classe, chaîne $nom_methode, bool $retour)`.
- `public mixte invoke(stdclass $object, mixte $args)`.
- `public mixte invokeArgs(stdclass $objet, tableau $args)`.
- `public bool isFinal()` permet de vérifier si une classe peut être dérivée ou non.
- `public bool isAbstract()` vérifie si une classe peut être utilisée directement ou s'il faut passer par une autre classe qui en dérive.
- `public bool isPublic()` détermine si une méthode possède un accès public.
- `public bool isPrivate()` détermine si une méthode possède un accès privé.
- `public bool isProtected()` détermine si une méthode possède un accès protégé.
- `public bool isStatic()` détermine si une fonction peut être redéfinie (ou non) dans une classe qui en hérite.
- `public bool isConstructor()` retourne `true` si la méthode est un constructeur et `false` sinon.
- `public bool isDestructor()` retourne `true` si la méthode est un destructeur et `false` sinon.
- `public entier getModifiers()`.
- `public ReflectionClass getDeclaringClass()` crée une réflexion sur la classe dans laquelle la méthode (sur laquelle vous faites la réflexion) est définie.

8.5 Cas pratique : un système de plug-ins

Vous venez d'avoir une idée géniale (et rentable) et vous désirez créer une application complète en PHP5. Vous avez pour objectif de faire évoluer le projet en ajoutant, dans un futur plus ou moins proche, de nouvelles fonctionnalités et ce, sans avoir à retourner dans

le code déjà écrit. Et quand cette fonctionnalité est en place, un lien sur la page d'accueil, sur laquelle vous n'avez rien modifié, affiche le lien comme par magie. Car, une chose est sûre, quand vous n'avez pas touché au code d'un projet durant des mois, se replonger dedans, même quand vous en êtes le concepteur, n'est pas toujours aisé quand le projet fait plusieurs dizaines (centaines ?) de milliers de lignes de codes et quelques centaines (milliers ?) de fichiers. Vous allez découvrir dans ce cas pratique, grâce à la réflexion, que vous pourrez ajouter de nouvelles fonctionnalités sans avoir à retoucher le code déjà écrit dans le projet. Et l'avantage, c'est que l'inverse est tout aussi vrai. Vous pourrez supprimer une fonctionnalité juste en supprimant les fichiers concernés sans avoir à modifier quoi que ce soit dans le reste du projet. Et mieux encore, si vous désirez garder les fichiers présents sur votre serveur, vous n'aurez qu'à placer le code du plug-in et de la fonctionnalité en commentaire.

Dans les différentes méthodes étudiées, il y en aura trois. L'exemple est montré pour une application web quelle qu'elle soit : site Internet, application intranet/extranet. Il s'agit de partir d'un noyau en MVC (*Model View Controller*) qui sera le cœur du projet. Plus significativement, il s'agit de la partie du programme qui prendra en compte toutes les nouvelles fonctionnalités sans que rien ne soit modifié dans ce projet. Il existe donc en racine un fichier *index.php* qui appelle le contrôleur. Également en racine, il y a un répertoire *models* qui contient tous les fichiers nommés *Model*.php* et un répertoire *views* qui contient tous les fichiers nommés *View*.php* où * représente le nom de l'action à effectuer (par exemple, si vous désirez ajouter un client, les fichiers à créer seront *ModelAjouterClient.php* et *ViewAjouterClient.php*). Évidemment, il est également nécessaire de créer un répertoire *include* qui contient des classes dont les fichiers sont nommés *Class*.php*. Puis, vient le répertoire le plus important, toujours en racine, qui est le fameux (indispensable !) répertoire nommé *plug-ins* dans lequel vous allez placer des modules, vos fameux plug-ins qui seront nommés **.php*, qui ne sont ni des modèles, ni des vues mais des fonctionnalités automatiquement récupérées par le projet et rendront ainsi actifs les modèles et les vues dès que le fichier *.php* sera ajouté dans le répertoire *plug-ins* (en gardant le même exemple, pour que la fonctionnalité `ajouter un client` soit prise en compte, vous ajouterez le fichier *AjouterClient.php*). En clair, toute la partie MVC est le noyau qui va récupérer les plug-ins afin de permettre à l'application de récupérer les fonctionnalités au fur et à mesure que celles-ci sont installées.

Première méthode : la plus simple pour comprendre

Dans le projet, vous décidez de consacrer une petite partie de la page d'accueil à l'affichage de liens vers des sites Internet distants, ce qui pourrait être par exemple vos sites favoris, ceux que vous fréquentez le plus. Probablement qu'au commencement du projet, il n'y aura aucun lien. Ce n'est pas la priorité du site. Donc, il ne faut pas de

messages d'erreurs si le répertoire *plug-ins* ne contient aucun fichier et en même temps il faut prévoir le fait qu'il en possèdera bientôt afin de ne pas avoir à revenir dans le code du noyau ultérieurement excepté pour optimiser celui-ci.

index_methode1.php

```
<?php
$dir = opendir('./plug-ins'); // On scanne le répertoire

// On scanne le répertoire plug-ins
while(($file = readdir($dir)) !== false){
  //Si le nom du fichier est différent de . et de ..
  if($file != '.' && $file != '..'){
    // on appelle le fichier
    require_once('plug-ins/'.$file);

    // Puis on sépare le nom du fichier de son extension
    // afin de récupérer le nom de la classe
    // se trouvant dans le fichier
    list($className, $extensionFile) = explode(".", $file);

    // On appelle la classe qui se trouve dans le fichier
    $oRC = new ReflectionClass($className);

    // On vérifie que la classe possède bien
    // la méthode 'ajouterLien()'
    $oRC->hasMethod('ajouterLien')
 ? $lien = 'true' : $lien = 'false';

    if($lien == 'true'){ // Si c'est le cas
      $oCN = new $className;

      // On ajoute le lien dans la page web
      // en fonction des informations
      // récupérées dans la classe
      // par les propriétés de classe
      echo '<a href="'. $oCN->leLien .'" />'.
         $oCN->laDescriptionDuLien .'</a><br/>';
    }
  }
}

closedir($dir);
?>
```

Tel que programmé dans cet exemple, il est primordial que le nom de la classe soit identique au nom du fichier. Voici la définition de la classe *MicroApplications.php* à insérer dans le répertoire *plug-ins* qui permet d'ajouter dans la page d'accueil de l'application un lien externe (à l'application) qui pointe vers le site Internet de Micro Application :

MicroApplications.php

```
<?php
/**
 * Cette classe est en fait un plug-in
 * IMPORTANT : Il est PRIMORDIAL que le nom de la classe
 * soit identique au nom du fichier sans l'extension .php
 * et en respectant la même casse
 *
 * Cette classe est appelée par un fichier qui
 * utilise la méthode de réflexion pour récupérer
 * les informations nécessaires
 *
 */
class MicroApplications {
  public $leLien = 'http://www.microapp.com/';
  public $laDescriptionDuLien = 'MicroApplications';

  public function __construct(){}

  public function __destruct(){}

  public function ajouterLien(){}
}

?>
```

Ce premier exemple est une solution mais elle n'est pas très efficace. Imaginez le nombre de fois que vous allez appeler la fonction `require_once()` dans la boucle si vous avez près d'une centaine de plug-ins d'installés. La quantité de ressources consommée serait très grande et le temps d'affichage de la page s'en trouverait nettement augmenté. Pourquoi cet exemple alors ? Pour vous rappeler que la première bonne solution trouvée n'est pas forcément la meilleure. Elle est certainement valable puisque cela fonctionne et appelle bien les bons plug-ins. Cependant, elle est loin d'être la meilleure. Voyez tout de suite la deuxième méthode.

Deuxième méthode : visez l'efficacité

Dans l'exemple qui suit, dès que le script trouve un fichier, il utilise la fonction `explode()` afin de séparer dans le nom du fichier l'action à effectuer du nom de la classe qui décrit précisément ce que va faire le fichier. Et c'est ce préfixe qui va dire quelle est l'action que va effectuer le plug-in : ajouter un lien externe, un lien interne, manipuler des données SQL, et/ou toute autre manipulation que vous aurez définie dans votre projet. Si le nom du préfixe correspond à l'action `ajouterLien`, alors on exécute l'action qui va ajouter dans la partie de la page le lien qui va pointer vers le site Internet distant. Voici le script qui appelle les plug-ins concernés :

index_methode2.php

```
<?php
$file = '';
$dir = opendir('./plug-ins'); // On scanne le répertoire

// On scanne les répertoires
while(($file = readdir($dir)) !== false){
  //Si le nom du fichier est différent de . et de ..
  if($file != '.' && $file != '..'){
    // On décompose le nom de fichier pour connaître l'action à
    ➥ effectuer
    list($action, $filename) = explode("_-_", $file);
    //echo $action .'<br/>';
    if($action == 'ajouterLien'){

      require_once('plug-ins/'. $file);

      // Puis on sépare le nom du fichier de son extension
      // afin de récupérer le nom de la classe se trouvant dans le
      ➥ fichier
      list($className, $extensionFile) = explode(".", $filename);

      // On appelle la classe qui se trouve dans le fichier
      $oRC = new ReflectionClass($className);

      $oCN = new $className;

      // On ajoute le lien dans la page web
      // grâce à la méthode invoke()
      $method = new ReflectionMethod($className, 'ajouterLien');
      echo $method->invoke($oCN, 'ajouterLien', true);
    }
  }
}
```

```
closedir($dir);
?>
```

Pour que ce script fonctionne, il faut que le fichier placé dans le répertoire *plug-ins* respecte une norme de nom qui est *action_-_nomclasse.php* (exemple : *ajouterLien_-_MicroApplications.php*). Voici le fichier *ajouterLien_-_MicroApplications .php* qui contient la classe qui va ajouter dans la page web le lien qui pointera vers le site distant Micro Application :

ajouterLien_-_MicroApplications.php

```
<?php
class MicroApplications {
  public $leLien = 'http://www.microapp.com/';
  public $laDescriptionDuLien = 'Micro Applications';

  public function __construct(){}

  public function __destruct(){}

  public function ajouterLien(){
    return '<a href="'. $this->leLien .'" />'.
       $this->laDescriptionDuLien .'</a>';
  }
}

?>
```

Les autres méthodes

En fait, vous avez le choix entre créer un fichier par lien ou placer tous les liens dans une table avec pour clé l'adresse Internet et pour valeur la description du lien que vous récupérerez grâce à la méthode `ajouterLien()` dans laquelle vous écrivez les instructions. L'avantage de cette méthode est que vous n'ouvrez qu'un seul fichier, ce qui rend la procédure beaucoup plus rapide. Seul inconvénient, quand vous désirez ajouter un lien supplémentaire, vous devez retourner dans le code du fichier placé dans le répertoire *plug-ins*.

Il existe bien d'autres méthodes pour créer des systèmes de plug-ins. Il vous revient d'optimiser cet exemple, voire de le traiter de façon complètement différente. Les deux méthodes que nous vous avons présentées n'utilisent pas les bases de données. Cela peut être une solution pourtant bien efficace si les tables SQL sont bien pensées. Vous pouvez

également mélanger tout un ensemble de techniques comme la réflexion avec le codage dynamique via Runkit et des interfaces graphiques via XSLT. Cela permet de mettre deux équipes sur un gros projet : l'une se consacrera aux interfaces graphiques et l'autre à l'évolution de la programmation des fonctionnalités supplémentaires à venir dans le projet via le système de plug-ins qui récupère également les fichiers XSLT pour l'affichage des données et ce, sans qu'aucune des deux équipes influe négativement sur l'autre. Et ce qui fonctionne pour deux équipes peut fonctionner pour trois ou plus. Développer cette idée nécessiterait un ouvrage à part probablement bien plus volumineux que celui-ci. Un jour peut-être…

8.6 Check-list

Dans ce chapitre, nous avons vu :

- ✔ ce qu'est la réflexion ;
- ✔ comment sont constituées l'API de réflexion et les classes qui la composent ;
- ✔ comment analyser du code avec les différentes classes de l'API de réflexion ;
- ✔ les différentes classes disponibles avec l'API de réflexion ;
- ✔ comment créer un système de plug-ins via l'API de réflexion.

9

Soulager le serveur avec le cache

APC, pour *Alternative PHP Cache* (Cache PHP alternatif), va nous permettre d'alléger considérablement les tâches du serveur en conservant en mémoire (cache) différentes variables, ce qui nous permettra de stocker les résultats des requêtes SQL. Le serveur n'aura donc pas à effectuer ces requêtes et se contentera de générer la page. Bref, c'est un gain de temps (et de puissance) considérable.

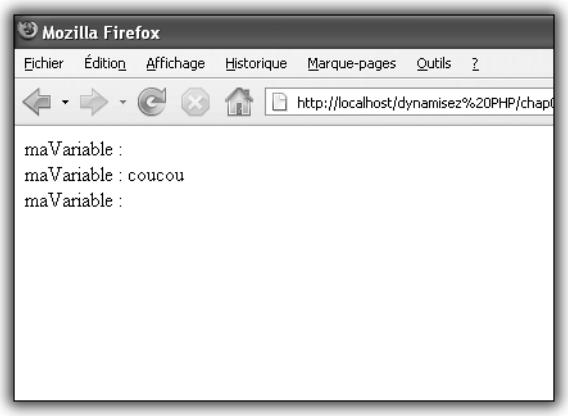
9.1 APC, tout simplement

Afin d'entrer en contact avec APC, nous allons rédiger un petit script qui stockera une variable dans le cache, qui l'affichera, puis l'effacera.

Dans un dossier nommé *chap09*, créez le fichier *apc_test.php* et remplissez-le comme suit :

apc_test.php

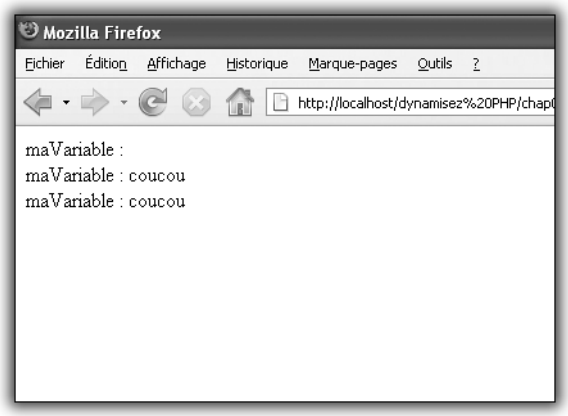
```
<?php
$maVariable = 'coucou';
echo "maVariable : ";
echo(apc_fetch('maVariable'));//n'écris rien car
     //la variable n'est pas encore stockée.
echo "<br/>";

apc_add('maVariable', $maVariable);//stocke la variable

echo "maVariable : ";
echo(apc_fetch('maVariable'));//écris le contenu
     //de la variable

apc_clear_cache ( "user" );//vide le cache

echo "<br/>";
echo "maVariable : ";
echo(apc_fetch('maVariable'));//n'écris rien car
     //la variable a été effacée du cache.
?>
```



▶ Fig. 9.1 : Seule la variable qui existe est affichée.

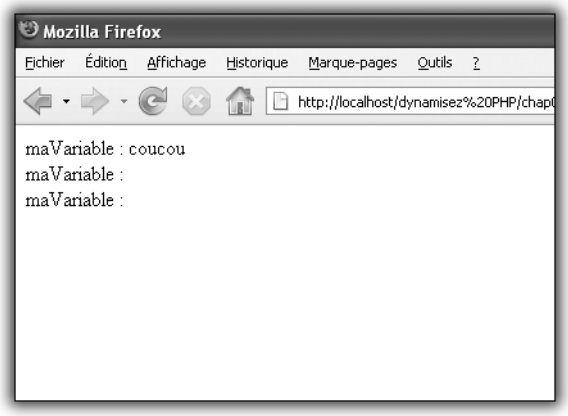
Dans ce premier exemple, nous utilisons trois fonctions : `apc_fetch()`, `apc_add()` et `apc_clear_cache()`. Il s'agit là des trois fonctions les plus utilisées lorsque l'on travaille avec APC. Leur fonctionnement précis sera détaillé dans la suite de ce chapitre. Pour le moment, nous allons améliorer le fonctionnement du système de cache.

En effet, dans notre court exemple, nous effaçons immédiatement la variable stockée dans le cache. Que se passe-t-il si nous supprimons de notre code l'appel à la fonction `apc_clear_cache()` ?



► Fig. 9.2 : Seul le premier appel (fonction apc_fetch()) ne renvoie rien.

Comme prévu, le premier appel de la fonction `apc_fetch()`ne renvoie rien, la variable `$maVariable` n'étant pas encore stockée dans le cache. Mais maintenant, cette variable restera stockée dans le cache indéfiniment. Chargez une nouvelle fois cette page.



► Fig. 9.3 : Seul le premier appel affiche quelque chose.

Le premier appel à `apc_fetch()`affiche bien le contenu de notre variable, mais qu'en est-il des deux autres appels ? Pourquoi n'affichent-ils rien ?

En fait, la fonction `apc_add()` met une variable en cache uniquement si celle-ci n'est pas déjà stockée. Dans le cas où cette variable existerait déjà dans le cache, la fonction retourne `false`. La variable n'est pas supprimée mais, à l'utilisation, cela revient pratiquement au même. Remplaçons donc la fonction `apc_add()` par la fonction `apc_store()` comme suit :

```
apc_store('maVariable', $maVariable);//stock la variable
```

Et voilà, le tour est joué !

Un système de cache évolué

Dans l'introduction de ce chapitre, nous avons pris l'exemple d'un forum. C'est exactement ce que nous allons faire maintenant afin de pénétrer plus avant dans l'utilisation d'APC. Nous n'allons pas, bien évidemment, en créer un entièrement, nous allons seulement nous intéresser à la génération des pages.

Pour ce faire, nous allons créer une base de données, que nous appellerons `forum`, et n'y ajouter qu'une seule table (nommée `fil`) que voici :

```
CREATE TABLE 'forum'. 'fil' (
'id' INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
'message' TEXT NOT NULL
)
```

Voici maintenant, d'un bloc, le fichier *apc_forum.php*, que nous allons modifier petit à petit tout au long de ce chapitre :

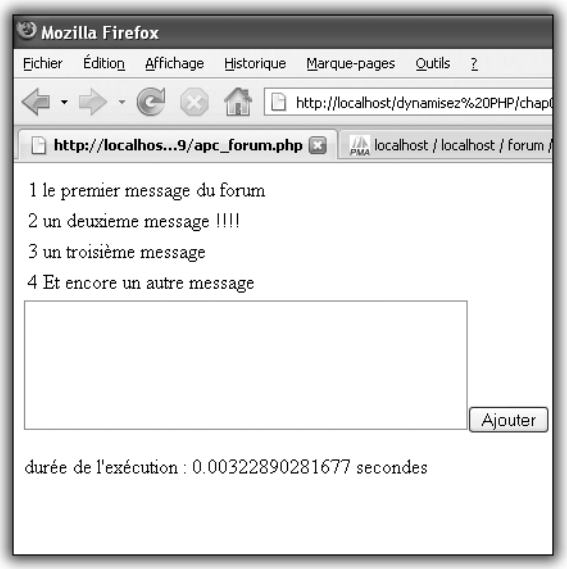
apc_forum.php

```
<?php
$tempsDebut = microtime(true);

// Parametres mysql à remplacer
define('DB_SERVER', 'localhost'); // serveur mysql
define('DB_SERVER_USERNAME', 'root'); // nom d'utilisateur
define('DB_SERVER_PASSWORD', ''); // mot de passe
define('DB_DATABASE', 'forum'); // nom de la base

$connect = mysql_connect(DB_SERVER,
                         DB_SERVER_USERNAME,
                         DB_SERVER_PASSWORD)
  or die('Impossible de se connecter : ' . mysql_error());
mysql_select_db(DB_DATABASE, $connect);
```

```
if($_GET['action'] == "ajouter"){
  $sql = "INSERT INTO 'forum'.'fil' ('id', 'message')";
  $sql .= "VALUES (NULL , '".$_POST['message']."')";

  if (!mysql_query($sql)) {
    echo "le message n'a pas été ajouté...";
  }
}

afficheFil();
afficheFormulaire();

function piedHTML(){
  echo "\n</body>\n</html>";
}

function afficheFil(){
  $sql = "SELECT * FROM 'fil' ";

  $resultat = mysql_query($sql);

  if (!$resultat) {
  echo "La requête a échoué.";
  } else {
    afficheMessage($resultat);
  }
}

function afficheMessage($r){
  $resultat = "<table>";
  while ($line = mysql_fetch_assoc($r)) {
    $resultat .= "<tr>";
    foreach ($line as $col_value) {
      $resultat .= "<td>$col_value</td>";
    }
    $resultat .= "</tr>";
  }
  $resultat .= "</table>";
  echo utf8_decode($resultat);
}

function afficheFormulaire(){
  echo '<form method="post" action="apc_forum.php?action=ajouter">';
  echo '<textarea name="message" rows="5" cols="40"></textarea>';
  echo '<input type="submit" value="Ajouter" />';
  echo '</form>';
```

```
}

$tempsFin = microtime(true);
$temps = $tempsFin - $tempsDebut;

echo utf8_decode("durée de l'exécution : ".$temps." secondes");
?>
```

Comme vous pouvez le constater, ce fichier n'est pas un modèle de programmation pour la communication avec mySQL, mais ce n'est pas le but ni de ce chapitre ni de cet ouvrage.



► Fig. 9.4 :
Un minuscule extrait d'un petit forum

Notez la présence d'un chronomètre qui va nous permettre d'évaluer APC.

Créons maintenant un autre fichier que nous appellerons *apc_temoin.php*, puis copions-y le contenu du fichier *apc_forum.php*. De cette manière, nous pourrons toujours comparer les performances du fichier *apc_forum.php*, que nous modifierons en utilisant avec APC, celles du fichier *apc_temoin.php*.

Mettre des réponses à des requêtes SQL dans le cache

La première modification que nous allons effectuer concerne le résultat de la requête SQL. Avant d'appeler la fonction `afficheFil()`, la fonction qui exécute la requête SQL, notre fichier se doit de vérifier qu'il n'y a pas déjà un résultat dans le cache. Modifions-le donc ainsi :

```
...
if(apc_fetch('stock') != 'en stock'){
  afficheFil();
} else {
  $html = apc_fetch('resultat');
  echo utf8_decode($html);
}
afficheFormulaire();
...
```

Grâce à la fonction `apc_fetch()`, notre programme commence par regarder si la variable de cache `stock` existe. Il faut maintenant que notre fichier créé dans le cache contienne le résultat de la requête SQL. Nous allons donc rajouter quelques lignes à notre fonction `afficheMessage()` :

```
…
function afficheMessage($r){
  $resultat = "<table>";
  while ($line = mysql_fetch_assoc($r)) {
    $resultat .= "<tr>";
    foreach ($line as $col_value) {
      $resultat .= "<td>$col_value</td>";
    }
    $resultat .= "</tr>";
  }
  $resultat .= "</table>";
  echo utf8_decode($resultat);
  apc_store('resultat', $resultat);//stocke la variable
  apc_store('stock', 'en stock');
}
…
```

La fonction `apc_store()` stocke nos deux variables de cache `resultat` et `stock`. Cela fait, notre programme stocke la réponse de la requête SQL dans le cache, et ne l'exécutera plus. Nous remarquons pourtant que, même lorsque notre programme ne fait pas de requête, il se connecte quand même à la base de données. Nous réglons donc ce léger problème comme suit :

```
…
// Paramètres mysql à remplacer
define('DB_SERVER', 'localhost'); // serveur mysql
define('DB_SERVER_USERNAME', 'root'); // nom d'utilisateur
define('DB_SERVER_PASSWORD', ''); // mot de passe
define('DB_DATABASE', 'forum'); // nom de la base
```

```
if($_GET['action'] == "ajouter"){
  connexion();
  $sql = "INSERT INTO 'forum'.'fil' ('id', 'message')";
  $sql .= "VALUES (NULL , '".$_POST['message']."')";

  if (!mysql_query($sql)) {
    echo "le message n'a pas été ajouté...";
  }
}

if(apc_fetch('stock') != 'en stock'){
  afficheFil();
} else {
  $html = apc_fetch('resultat');
  echo utf8_decode($html);
}
…
…
function connexion(){
$connect = mysql_connect(DB_SERVER,
                         DB_SERVER_USERNAME,
                         DB_SERVER_PASSWORD)
  or die('Impossible de se connecter : ' . mysql_error());
mysql_select_db(DB_DATABASE, $connect);
}
…
```

Essayons maintenant, grâce à `apc_forum()`, de rajouter un message à notre fil de discussion. Comme vous pouvez le vérifier, notre nouvelle entrée n'apparaît pas dans notre mini-forum. Pourtant, notre nouveau message a correctement été enregistré dans la base de données… C'est normal. Notre variable étant déjà dans le cache, la requête SQL n'est pas effectuée. Nous avons plusieurs alternatives pour régler ce problème.

Nous pouvons par exemple vider le cache au moment de la requête SQL d'enregistrement, grâce à la fonction `apc_clear_cache()` :

```
…
if($_GET['action'] == "ajouter"){
connexion();
  $sql = "INSERT INTO 'forum'.'fil' ('id', 'message')";
  $sql .= "VALUES (NULL , '".$_POST['message']."')";

  if (!mysql_query($sql)) {
    echo "le message n'a pas été ajouté...";
  }
  apc_clear_cache ( "user" );//vide le cache
```

```
}
…
```

Ou alors, nous pouvons ajouter un troisième argument à la fonction `apc_store()` :

```
…
function afficheMessage($r){
  $resultat = "<table>";
  while ($line = mysql_fetch_assoc($r)) {
    $resultat .= "<tr>";
    foreach ($line as $col_value) {
      $resultat .= "<td>$col_value</td>";
    }
    $resultat .= "</tr>";
  }
  $resultat .= "</table>";
  echo utf8_decode($resultat);
  apc_store('resultat', $resultat, 5);//stocke la variable
  apc_store('stock', 'en stock', 5);
}
…
```

En procédant de cette manière, les variables de cache `resultat` et `stock` seront effacées au bout de cinq secondes (ou plutôt au moment de la requête suivante, si celle-ci apparaît après cinq secondes). Ce troisième argument peut aussi être ajouté à la fonction `apc_add()`.

APC et les constantes

Au tout début de notre fichier *apc_forum.php*, nous définissons un certain nombre de constantes que nous utilisons pour la connexion à la base de données. Il n'est cependant pas judicieux de les redéfinir à chaque chargement de notre programme.

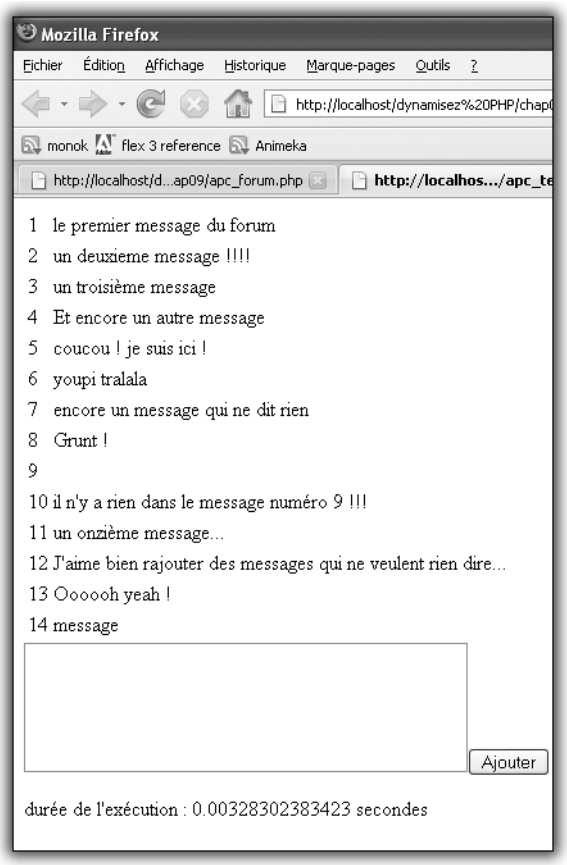
APC permet la mise en cache des constantes et est même un peu plus rapide que `define()`. Nous allons donc modifier notre code afin qu'il intègre cette fonctionnalité. Modifiez-donc *apc_forum.php* de la manière suivante :

```
…
if(!apc_load_constants('constantes')){
  // Parametres mysql à remplacer

  $constantes = array(
    'DB_SERVER' => 'localhost',
    'DB_SERVER_USERNAME' => 'root',
    'DB_SERVER_PASSWORD' => '',
```

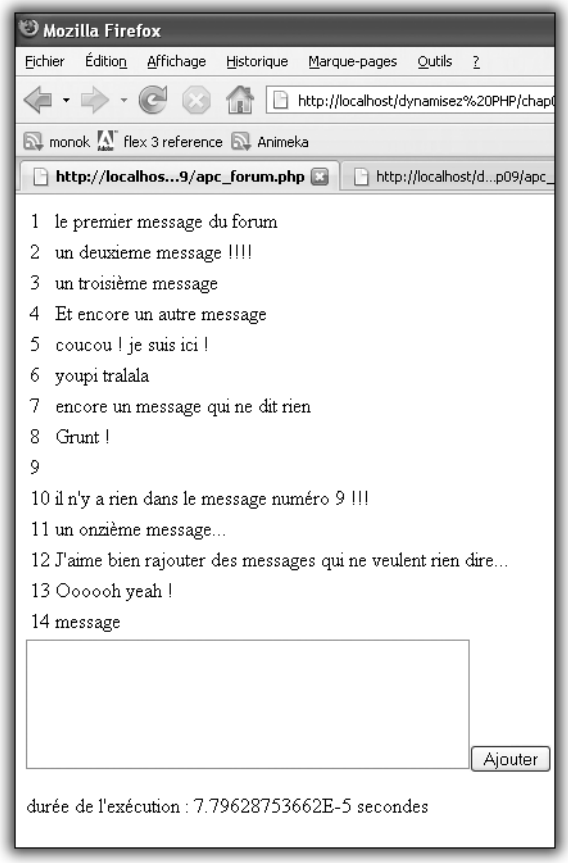
```
    'DB_DATABASE', 'forum'
  );
  apc_define_constants('constantes', $constantes);
}

if(apc_fetch('stock') != 'en stock'){
…
```

En passant un tableau contenant toutes les constantes et une clé d'indexation à la fonction `apc_define_constants()`, nous les stockons dans le cache. Pour les appeler depuis le cache, nous utilisons simplement la fonction `apc_load_content()`…

Voyez maintenant la différence de performance entre les fichiers *apc_temoin.php* et *apc_forum.php*.



► Fig. 9.5 : Le fichier apc_temoin.php met un certain temps à s'afficher.



► Fig. 9.6 : Le fichier apc_forum.php que nous avons modifié est beaucoup plus rapide, même avec seulement 14 messages.

9.2 Check-list

Après avoir lu ce chapitre, utiliser le cache avec APC est devenu chose aisée.

Nous avons vu :

- ✔ comment charger des variables en mémoire cache grâce aux fonctions `apc_add()` et `apc_store()` ;
- ✔ comment accéder à des variables disponibles en mémoire cache ;
- ✔ comment enregistrer et lire des constantes depuis la mémoire cache.

10

Compiler le code PHP

Bcompiler permet de compiler votre code PHP. Mais à quoi cela sert-il ? Tout simplement à protéger vos lignes de code si durement créées à la sueur de votre front. En effet, les fichiers compilés par Bcompiler sont illisibles pour un être humain, mais parfaitement utilisables par un ordinateur. Vous pouvez ainsi diffuser vos fichiers PHP compilés sans craindre pour leur intégrité.

10.1 Créer du bytecode

Compiler un fichier

Afin de s'initier à Bcompiler, nous allons commencer par le plus simple : la compilation d'un fichier PHP complet.

Dans un dossier intitulé *chap10*, nous allons créer un dossier *bytecode* ainsi que les fichiers *bcompiler_fichier.php* et *bcompiler_fichier_proprietaire.php*.

Voici le contenu de *bcompiler_fichier_proprietaire.php* :

bcompiler_fichier_proprietaire.php

```
<?php
$secret = "mot de passe hyper_secret";
echo "mon fichier PHP<br/>\n";
?>
```

Si l'on invoque ce fichier via la commande `include`, la phrase "mon fichier PHP" apparaîtra simplement à l'écran.

Le second fichier, *bcompiler_fichier.php* va seulement nous servir à compiler le fichier *bcompiler_fichier_proprietaire.php*, puis à afficher son contenu :

bcompiler_fichier.php

```
<?php
// adresse du futur fichier compilé :
$bytecode = "bytecode/bcompiler_fichier_proprietaire.phb";

// adresse du code source :
$codeSource = "bcompiler_fichier_proprietaire.php";

// création du fichier compilé :
$fichierBytecode = fopen($bytecode, "w");

// écriture de l'en-tête du fichier :
bcompiler_write_header($fichierBytecode);

//écriture du corps du fichier :
bcompiler_write_file($fichierBytecode, $codeSource);

// écriture du pied de page du fichier :
bcompiler_write_footer($fichierBytecode);
```

```
// fermeture du fichier :
fclose($fichierBytecode);

// appel du fichier compilé :
include 'bytecode/bcompiler_fichier_proprietaire.phb';
?>
```

Lorsque vous lancez ce fichier, la phrase "mon fichier PHP" apparaît comme convenu à l'écran : le fichier de bytecode a bien été créé. Vous pouvez remarquer l'extension de ce nouveau fichier : *.phb*, comme "PHp Bcompiler" ; mais il ne s'agit là que d'une convention. Nous aurions pu choisir n'importe quelle autre.

Les étapes pour créer un fichier de bytecode sont assez simples. D'abord, nous créons le fichier lui-même, via la fonction `fopen()`Ensuite, nous écrivons le contenu du fichier via les fonctions `bcompiler_write_header()`, `bcompiler_write_file()` et `bcompiler_write_footer()`. Nous en déduisons donc qu'un fichier de bytecode, comme n'importe quel fichier, dispose d'un en-tête, d'un corps et d'un pied de page.

Attardons-nous un peu sur ce fichier de bytecode : ouvrons-le avec un éditeur de texte, tel que Notepad. Que voyons-nous ? Des carrés et des espaces à foison. Ainsi, notre code est protégé. Et pourtant… Si nous regardons un peu plus attentivement, nous voyons très lisiblement écrite la chaîne de caractères "mon fichier PHP", ainsi que le contenu de la variable `$secret`. C'est tout à fait normal : une chaîne de caractères n'étant pas, à proprement parler, du code, il serait absurde de tenter de la compiler… Il ne faut donc pas laisser de données sensibles, comme un mot de passe, par exemple, dans un fichier de bytecode, à moins de les crypter au préalable.

Comparons maintenant le poids du fichier de bytecode et celui du fichier source :

► Fig. 10.1 :
Notre fichier source

► Fig. 10.2 :
Notre fichier compilé

Vous vous rendez compte que le fichier de bytecode pèse environ cinq fois plus lourd que le fichier source ! C'est hélas le prix à payer pour protéger ses algorithmes.

Test de vitesse

Nous allons maintenant comparer la vitesse d'exécution d'un fichier de bytecode et du fichier source. Pour ce faire, nous allons créer trois fichiers : *bcompiler_vitesse.php*, *bcompiler_vitesse_ouvert.php* et *bcompiler_vitesse_secret.php*.

bcompiler_vitesse_ouvert.php

```
<?php
function ouvert(){
  for($i = 0 ; $i < 100000 ; $i++){
    $a = $i." : ".sqrt($i*5)."<br/>\n";
  }
}
?>
```

Ce fichier ne contient qu'une seule fonction dont le but avoué est de faire travailler le processeur un certain temps.

bcompiler_vitesse_secret.php

```
<?php
function secret(){
  for($i = 0 ; $i < 100000 ; $i++){
    $a = $i." : ".sqrt($i*5)."<br/>\n";
  }
}
?>
```

Le fichier *bcompiler_vitesse_secret.php* a exactement le même but que le fichier *bcompiler_vitesse_ouvert.php*, à ceci près qu'il va être compilé.

Enfin, voici le code du troisième fichier (*bcompiler_vitesse.php*) :

bcompiler_vitesse.php

```
<?php
// adresse du futur fichier compilé :
$bytecode = "bytecode/bcompiler_fonctions_secret.phb";

// adresse du code source :
$codeSource = "bcompiler_fonctions_secret.php";
```

```
// création du fichier compilé :
$fichierBytecode = fopen($bytecode, "w");

// écriture de l'en-tête du fichier :
bcompiler_write_header($fichierBytecode);

//écriture du corps du fichier :
bcompiler_write_file($fichierBytecode, $codeSource);

// écriture du pied de page du fichier :
bcompiler_write_footer($fichierBytecode);
fclose($fichierBytecode);

echo "poids avant compilation : <b>".filesize($codeSource)."
➥ octets</b><br/>";

include 'bcompiler_fonctions_ouvert.php';
include 'bytecode/bcompiler_fonctions_secret.phb';

$tempsDebut = microtime(true);
ouvert();
$tempsFin = microtime(true);
$temps = $tempsFin - $tempsDebut;

echo "Nous avons mis <b>$temps</b> secondes avec le code
➥ source.<br/>\n";

echo "poids après compilation : <b>".filesize($bytecode)."
➥ octets</b>.<br/>";

$tempsDebut = microtime(true);
secret();
$tempsFin = microtime(true);
$temps = $tempsFin - $tempsDebut;

echo "Nous avons mis <b>$temps</b> secondes avec le bytecode.<br/>\n";

?>
```

Ce dernier fichier va compiler *bcompiler_vitesse_secret.php*, puis exécuter les deux fonctions et comparer les vitesses d'exécution de l'une et de l'autre, ainsi que le poids du fichier source et du fichier de bytecode.

Voici les résultats obtenus lorsque nous lançons ce script depuis la machine sur laquelle nous écrivons :

▶ Fig. 10.3 :
Résultat du script

Nous pouvons voir qu'un fichier de bytecode n'est pas exécuté plus rapidement que s'il n'avait pas été compilé.

Compiler des classes et des fonctions

Les fonctions

Bcompiler, en plus de permettre la compilation complète d'un fichier, permet de n'en compiler que certaines fonctions. Cela est très pratique pour obtenir un fichier compilé contenant quelques utilitaires.

Nous allons donc, toujours dans notre dossier *chap10*, créer deux nouveaux fichiers. Le premier, *bcompiler_fonction_source.php*, contiendra le code source de quelques fonctions et les compilera. Le second, *bcompiler_fonction.php*, nous servira à tester le fichier de bytecode généré par le premier fichier.

Voici le code de notre premier fichier :

bcompiler_fonction_source.php

```
<?php

// adresse du futur fichier compilé :
$bytecode = "bytecode/bcompiler_fonction.phb";

// Création du fichier compilé
$fichierBytecode = fopen($bytecode,"w");

// écriture de l'en-tête du fichier :
bcompiler_write_header($fichierBytecode);

// écriture des fonctions au sein du fichier compilé
bcompiler_write_function($fichierBytecode,"phraseEnCouleur");
bcompiler_write_function($fichierBytecode,"ajoutPhrase");
bcompiler_write_function($fichierBytecode,"ecritPhrase");
```

```
// écriture du pied de page du fichier :
bcompiler_write_footer($fichierBytecode);
fclose($fichierBytecode);

// les fonctions à compiler :
function phraseEnCouleur($phrase = "Phrase par défaut",
                         $couleur = "#DE0059"){
  return '<div style="color:'.$couleur.'">'.$phrase.'</div>';
}

function ajoutPhrase(&$tab, $phrase){
  $tab[] = $phrase;
}

function ecritPhrase(&$tab){
  for($i = 0 ; $i < count($tab) ; $i++){
    echo $tab[$i];
  }
}
?>
```

Dans ce fichier, nous utilisons différents types de fonctions : avec ou sans valeur de retour, avec des arguments disposant ou non de valeur par défaut ou avec des arguments passés par références.

Afin de les écrire dans le corps du fichier de bytecode, nous utilisons la fonction `bcompiler_write_function()` pour chacune de nos fonctions.

Regardons maintenant le simplissime code du fichier *bcompiler_fonction.php* :

bcompiler_fonction.php

```
<?php
include 'bytecode/bcompiler_fonction.phb';
$tableau = array();
$str = phraseEnCouleur("Salut à vous !<br/>La forme ?", "#455FDB");
ajoutPhrase($tableau, $str);
ajoutPhrase($tableau, phraseEnCouleur());
ecritPhrase($tableau);
?>
```

L'utilisation est plutôt simple, n'est-ce pas ?

Cela dit, imaginez que vous ayez une cinquantaine de fonctions à compiler : il serait vraiment fastidieux d'écrire cinquante fois la fonction `bcompiler_write_function()`. Si toutes vos fonctions se trouvent au sein d'un même fichier, vous pouvez utiliser la

fonction `bcompiler_write_functions_from_file()`. En une seule ligne, toutes les fonctions d'un même fichier seront compilées et ajoutées au bytecode.

Ouvrons à nouveau le fichier *bcompiler_fonction_source.php*, et remplaçons les trois appels à la fonction `bcompiler_write_function()`par cette seule ligne :

```
Bcompiler_write_functions_from_file($fichierBytecode,
➥ 'bcompiler_fonction_source.php');
```

Le tour est joué. Il ne reste plus qu'à compiler notre fichier de bytecode (en lançant le fichier *bcompiler_fonction_source.php*), et à exécuter de nouveau le fichier *bcompiler_fonction.php* pour s'assurer que tout fonctionne bien.

Les classes

Bcompiler met aussi à notre disposition un outil pour compiler des classes. Afin d'illustrer notre propos, nous allons créer deux fichiers : *bcompiler_class_source.php* et *bcompiler_class.php*. Le premier contiendra le code de quelques classes ainsi que les quelques lignes nécessaires à leur compilation. Le second servira bien sûr à tester le fichier de bytecode.

bcompiler_class_source.php

```
<?php

// adresse du futur fichier compilé :
$bytecode = "bytecode/bcompiler_class.phb";

// Création du fichier compilé
$fichierBytecode = fopen($bytecode,"w");

// écriture de l'en-tête du fichier :
bcompiler_write_header($fichierBytecode);

// écriture des classes au sein du fichier compilé
bcompiler_write_class($fichierBytecode, "Animal");
bcompiler_write_class($fichierBytecode, "Mammifere");

// écriture du pied de page du fichier :
bcompiler_write_footer($fichierBytecode);
fclose($fichierBytecode);

// les classes à compiler :
class Animal {

  public $nom;
```

```
  function __construct($nom){
    $this->nom = $nom;
  }

  function presentation(){
    return ' Je suis : <b>'.$this->nom.'</b>';
  }

  function seFaitManger($animal){
    return $this->nom." : \"Rosebud !\"<br/>";
  }
}

class Mammifere extends Animal{

  public $regime;

  function __construct($nom, $regime){
    parent::__construct($nom);
    $this->regime = $regime;
  }

  function presentation(){
    return parent::presentation().
        '. Mon régime : <b>'.
        $this->regime.'</b>';
  }

  function mange($animal){
    return $this->nom.
           ' est en train de manger '.
           $animal->nom."<br/>".
           $animal->seFaitManger($this);
  }
}

?>
```

Dans cet exemple, nous utilisons deux classes : `Animal` et `Mammifere`. La classe `Mammifere` héritant de la classe `Animal`, il est nécessaire de compiler d'abord `Animal`, puis `Mammifere`. Essayons de compiler de cette manière :

```
bcompiler_write_class($fichierBytecode, "Mammifere");
bcompiler_write_class($fichierBytecode, "Animal");
```

Ce code génère une erreur car la classe `Mammifere` ne trouve pas sa classe mère dans le bytecode !

Une fois que nous avons exécuté ce fichier (avec les classes dans le bon ordre), le fichier *bcompiler_class.phb* a été créé. Testons-le donc :

bcompiler_class.php

```
<?php
include 'bytecode/bcompiler_class.phb';

$bestiole = new Animal("lapin");
echo $bestiole->presentation()."<br/>";
$bestiole2 = new Mammifere("tigre", "carnivore");
echo $bestiole2->presentation()."<br/>";
echo $bestiole2->mange($bestiole);
?>
```

Comme pour la compilation de fonctions, l'usage est relativement simple. Cependant, Bcompiler n'offre pas de fonction permettant en une seule ligne de code la compilation de toutes les classes d'un fichier : en effet, rien ne lui indiquerait quelle classe descend de quelle autre, et Bcompiler ne peut pas le déduire tout seul.

10.2 Lire le bytecode

Jusqu'à présent, nous avons toujours utilisé `include` pour lire le contenu d'un fichier de bytecode. Les commandes `include` et `require` resteront les manières les plus simples et les plus portables d'utiliser un fichier compilé. Mais il existe d'autres méthodes.

Par exemple, renommons le fichier *bcompiler_fichier_proprietaire.phb* (qui devrait se trouver dans votre répertoire *bytecode*) en *bcompiler_fichier_proprietaire.php* : nous changeons seulement l'extension. Ouvrez ensuite ce fichier depuis votre navigateur préféré. Comme il s'agit là d'un simple script pouvant fonctionner seul, le navigateur affiche bien la phrase "mon fichier PHP".

Un fichier compilé, pour peu qu'il porte l'extension *.php*, peut donc être lu directement.

La fonction bcompiler_read()

Nous pouvons accéder au contenu d'un fichier compilé grâce à la fonction `bcompiler_read()`. Cette fonction, contrairement aux structures `include` ou `require`, n'exécute pas le code contenu dans le fichier.

Créons donc un fichier *bcompiler_lire.php*. Ce fichier devra ouvrir trois fichiers de bytecode (l'un contenant un script, l'autre des fonctions et le dernier des classes. Si vous avez fait chaque exemple de ce chapitre, vous disposez déjà de ces fichiers : *bcompiler_fichier_proprietaire.phb* pour le fichier de script, *bcompiler_fonction.phb* pour

le fichier de fonctions et *bcompiler_class.phb* pour le fichier de classes), puis tenter d'en exécuter le contenu. Voici le code de *bcompiler_lire.php* :

bcompiler_lire.php

```
<?php
// ouverture du fichier de bytecode contenant un script
$fichierBytecode = fopen("bytecode/bcompiler_fichier_proprietaire.phb","r");
bcompiler_read($fichierBytecode);
fclose($fichierBytecode);

// essai de lecture d'une variable contenue dans le script
echo $secret;


// ouverture du fichier de bytecode contenant des fonctions
$fichierBytecode = fopen("bytecode/bcompiler_fonction.phb","r");
bcompiler_read($fichierBytecode);
fclose($fichierBytecode);

// essai d'utilisation de ces fonctions
$tableau = array();
$str = phraseEnCouleur("Salut à vous !<br/>La forme ?", "#455FDB");
ajoutPhrase($tableau, $str);
ajoutPhrase($tableau, phraseEnCouleur());
ecritPhrase($tableau);

// ouverture du fichier de bytecode contenant des classes
$fichierBytecode = fopen("bytecode/bcompiler_class.phb","r");
bcompiler_read($fichierBytecode);
fclose($fichierBytecode);

// essai d'utilisation de ces classes
$bestiole = new Animal("lapin");
echo $bestiole->presentation()."<br/>";
$bestiole2 = new Mammifere("tigre", "carnivore");
echo $bestiole2->presentation()."<br/>";
echo $bestiole2->mange($bestiole);
?>
```

Lorsque nous lançons ce fichier, nous nous rendons compte que le fichier de script ne s'exécute pas (la phrase "mon fichier PHP" n'apparaît pas à l'écran). De plus, la variable `$secret` ne s'affiche pas non plus. Et pourtant, aucune erreur ne nous est renvoyée. Par contre, les fichiers de fonctions et de classes s'exécutent correctement.

Quels avantages y a-t-il à utiliser cette fonction ? Si nous comparons les temps d'exécution, nous ne pouvons noter de différences significatives. Le seul avantage est de pouvoir charger ces fichiers compilés en un seul et même endroit du code, pour plus de clarté.

La fonction bcompiler_load()

Comme nous l'avons vu précédemment, un fichier compilé pèse environ cinq fois plus lourd que son alter ego non compilé. Heureusement, l'extension Bcompiler nous propose une petite fonction afin de résoudre ce problème de poids. En effet, `bcompiler_load()` permet de charger un fichier de bytecode compressé avec BZIP2 !

> ***Vous pouvez vous reporter au chapitre* Comprimez vos données*, traitant de la compression, pour plus de détails sur la manière d'utiliser BZIP2.***

Nous allons, en un seul petit fichier (placé dans votre répertoire *chap10*), nommé *bcompiler_bzip.php*, compresser le fichier *bcompiler_class.phb* (si vous avez effectué les exemples de ce chapitre, il devrait déjà exister dans le dossier *bytecode*) puis utiliser les classes contenues dans notre fichier compilé-compressé.

Voici le code de *bcompiler_bzip.php* :

bcompiler_bzip.php

```
<?php
$fichierSource = 'bytecode/bcompiler_class.phb';

// récupération, sous la forme d'une chaîne
// de caractères, du contenu du fichier de bytecode :
$contenu = file_get_contents($fichierSource);

// on écrit le contenu du fichier de bytecode
// à l'écran :
echo "<p>";
echo $contenu;
echo "</p>";
// On s'aperçoit bien, grâce à tous les caractères
// bizarres inscrits à l'écran, qu'il s'agit effectivement
// du contenu du fichier de bytecode !

// ouverture (ou création) du fichier compressé :
$bzo = bzopen('bytecode/compile_compresse.bz2', "w");

// Ecriture du code compilé dans le fichier compressé :
bzwrite($bzo, $contenu, strlen($contenu));

// Fermeture du fichier compressé :
bzclose($bzo);

// chargement des classes contenues
// dans notre fichier compilé-compressé :
```

```
bcompiler_load('bytecode/compile_compresse.bz2');

// Exécution de ces classes :
$bestiole = new Animal("lapin");
echo $bestiole->presentation()."<br/>";
$bestiole2 = new Mammifere("tigre", "carnivore");
echo $bestiole2->presentation()."<br/>";
echo $bestiole2->mange($bestiole);
?>
```

Et voilà ! Nos classes ont été d'abord compilées en un fichier de bytecode, puis compressées grâce à BZIP2. À cet effet, le fichier *compile_compresse.bz2* pèse environ quatre fois moins lourd que *bcompiler_class.phb*, son homologue non compressé.

Malheureusement, `bcompiler_load()`ne s'utilise qu'avec des classes. Si vous espériez compresser du script ou des fonctions compilées, il va falloir passer votre chemin.

Enfin, si nous gagnons en place, nous perdons en vitesse d'exécution. C'est évidemment normal, puisque nous subissons une étape de décompression. Un fichier compilé et compressé met environ deux fois plus de temps qu'un autre à s'exécuter.

Maintenant, c'est à vous de déterminer quelle méthode de compilation va être la plus adaptée à votre projet : faut-il compiler du script ? Des fonctions ? Des classes ? Tout à la fois ? Faut-il compresser ?

Et surtout, n'oubliez pas de vous poser la plus importante de toutes les questions : quelles parties de mon programme sont suffisamment sensibles pour justifier leur compilation et la perte de performances (en termes de poids ou de rapidité d'exécution) qui va avec ?

10.3 Check-list

Dans ce chapitre, nous avons découvert ce qu'était Bcompiler, ses avantages et ses inconvénients.

Nous avons vu comment :

- ✔ compiler vos fichiers PHP ;
- ✔ compiler des classes et des fonctions ;
- ✔ lire des fichiers, des classes ou des fonctions compilés ;
- ✔ lire des fichiers compilés et compressés.

11

Méthodologie

Ce chapitre a pour objectif de vous donner des pistes afin d'étudier des extensions que vous ne connaissez pas encore, et ce par vous-même. Il est facile de travailler une extension lorsque celle-ci est documentée. Mais comment faire lorsqu'elle ne l'est pas ?

11.1 Avant de commencer…

Chacun est unique. Les auteurs sont aussi uniques que vous l'êtes vous-même. Ces deux premières phrases sont là pour prévenir que, même si les différentes écoles essayent plus ou moins et tant bien que mal d'enseigner une pensée assez similaire et formatée, vous n'en restez pas moins des êtres humains. D'ailleurs, pourquoi avez-vous choisi l'informatique et pas la psychologie, l'histoire ou l'ethnologie ? Tout simplement parce que vous avez un passé, un vécu, avec des souvenirs, des réflexes conditionnés et inconscients. Et c'est sans compter le sujet de la mémoire cellulaire et des héritages transgénérationnels. Voilà pourquoi vous avez choisi l'informatique au point d'en faire une profession ou si ce n'est pas encore le cas, de vouloir en faire une profession ou encore un simple loisir. Donc, récupérez parmi les explications qui seront données dans ce chapitre tout ce qui vous semble être le meilleur pour vous et laissez de côté ce qui vous semble inutile, inefficace, voire même ridicule. Chacun possède son mode de pensée, sa logique et son ressenti. Et il y a autant de bonnes façons de faire pour atteindre un même résultat escompté qu'il y a d'êtres vivants.

Connaître ses motivations

Connaître ses besoins est la première chose à faire. En effet, comment peut-on évoluer si on ne sait pas où l'on va ? Rappelez-vous votre passé d'écolier, de collégien, de lycéen, rappelez-vous ces longs cours de mathématiques où l'on vous apprenait à calculer une intégrale, sans même savoir à quoi cela pouvait servir, ou si vous alliez utiliser cette connaissance à l'avenir. Mais peut-être ce jeu sans but vous amusait…

Quoi qu'il en soit, sans motivations, vous n'irez jamais bien loin dans vos découvertes. Alors, pourquoi apprenez-vous à utiliser telle ou telle extension ? Est-ce pour répondre à un besoin précis d'un projet ? Est-ce pour élargir le champ de vos connaissances et de vos compétences ? Pour épater quelqu'un ? Pour vous épater vous-même ? Est-ce tout simplement par jeu ? Il n'y a pas de mauvaises réponses. L'important, ce n'est même pas forcément de savoir pourquoi mais de savoir comment. Comment allez-vous trouver la bonne extension ? Comment allez-vous trouver les bonnes documentations et ne pas vous perdre dans un flot de données inutile et consommateur de temps ? Comment allez-vous intégrer et mémoriser les informations nécessaires à l'utilisation de façon naturelle, sans aide constante, la nouvelle extension ou API ou autre ? Comment ?

Savoir ce que vous ne savez pas

C'est une réflexion qui a toute son importance. Car s'il est très facile de savoir ce que l'on sait, surtout consciemment, il est beaucoup plus difficile de savoir ce que l'on ne sait pas.

Et quand vous vous rendez compte d'une non-connaissance, cela s'ensuit d'un acte encore plus difficile, celui d'admettre et de se confronter à la réalité du fait. Et plus le niveau du programmeur est élevé, plus il a du mal à se remettre en question. Beaucoup croient se remettre en question. Mais ne seraient-ils pas en train de se donner une bonne raison de ne pas le faire ? Se remettre en question ne signifie pas forcément avoir conscience de ce que l'on ne sait pas. Ne pas savoir… À partir du moment où vous prenez conscience et que vous admettez en votre for intérieur que vous ne savez pas, alors vous êtes sur le bon chemin pour apprendre.

Quand est-ce que c'est le plus difficile d'admettre, de nous avouer que l'on ne sait pas ? Quand on a pris une habitude qui fonctionnait jusque-là mais qui aujourd'hui montre ses limites… Et plus l'habitude dure depuis longtemps, moins on a envie d'admettre que c'est une mauvaise habitude. On a cru qu'elle était bonne pendant des années. Pourquoi aujourd'hui serait-elle mauvaise ? Parce que la vie est mouvement et que tout change à chaque milliardième de seconde. Pour ceux parmi vous qui modélisez des bases de données, tous savent que la première solution, aussi bonne soit-elle, n'est jamais la meilleure. Tout comme pour la modélisation objet.

Apprendre une extension pour les besoins d'un projet

Si vous devez apprendre une extension pour le bien d'un projet, la première question à se poser est celle de l'absolue nécessité de l'extension en question.

Par exemple, vous devez réaliser un export de certaines données contenues dans une base au format Microsoft Excel. La réponse est déjà claire : il va vous falloir apprendre COM. En êtes-vous bien sûr ? N'y aurait-il pas d'autres solutions plus simples, ne vous imposant pas un temps d'apprentissage ?

La réponse est oui : créez un fichier CSV (*Comma-Separated Value*, pour plus de renseignements, rendez-vous à l'adresse suivante : **http://tools.ietf.org/html/rfc4180**). Vous vous contenterez de faire de la simple concaténation de chaînes de caractères, et votre export sera compris par l'immense majorité des tableurs du marché. Ce qui représente un gain de temps phénoménal. Sachez qu'en plus, il existe une fonction interne au moteur PHP nommée `fgetcsv()` pour exploiter au mieux et très facilement tout fichier CSV.

La réponse est non : vous avez besoin d'utiliser certaines fonctions propres à Microsoft Excel. Et en plus de cela, votre client vous a clairement signifié qu'il désirait un fichier au format Excel et rien d'autre.

Apprendre une extension pour soi-même

Vous avez été exilé dans une ville lointaine, où vous ne connaissez personne. Qu'à cela ne tienne, vous allez approfondir vos connaissances en PHP car vous avez le matériel requis à disposition. Vous n'avez pas de cahiers des charges, encore moins d'objectifs ; il n'y a que vous face à votre liberté. Eh bien, il est vivement conseillé de chasser ces pensées de votre esprit. C'est une erreur qu'il vous faut corriger. Et si PHP est ce qu'il est aujourd'hui, c'est grâce à la communauté qui s'est très rapidement formée autour de ce langage. La communauté de PHP est l'une des plus puissantes du monde informatique avec quelques autres comme Linux.

Vous allez travailler une demi-heure en suivant un tutoriel quelconque, le temps de voir comment fonctionne votre extension. Et puis après ? Il vous faut un projet, quelque chose, même complètement inutile, au sein duquel vous allez exploiter votre extension, faire des expériences, tout planter, tout réussir, etc. Allez-y, faites fonctionner vos méninges, explorez les méandres de votre imagination. Non seulement vous allez apprendre ce que vous souhaitiez, mais vous allez aussi réviser vos acquis, ancrer un peu plus en vous les réflexes de programmation, trouver de meilleurs algorithmes que ceux que vous utilisiez déjà. Et qui sait… peut-être allez-vous avoir l'idée du siècle, celle qui changera votre vie (en mieux, du moins c'est ce que nous vous souhaitons) !

11.2 Différentes méthodes

Extension documentée

Le terme "extension documentée" veut dire que l'extension est documentée dans le manuel officiel de PHP. Vous pouvez avoir accès à ce manuel sur le site **http://php.net/**.

En ce qui concerne les extensions documentées, la méthodologie reste à peu près la même. Apprendre par cœur les syntaxes de chaque fonction. Certains se disent déjà : à quoi ça sert ?

Pensez-vous que vous pourriez parler français aujourd'hui sans avoir eu à apprendre par cœur un certain nombre de mots de telle sorte que vous savez les employer au moment opportun ? Il est fort probable que non et il en est de même des règles syntaxiques et grammaticales. Pensez-vous que vous pourriez courir, ou sauter des obstacles si vous n'aviez pas appris à marcher avant ? Tout ce qui est devenu naturel pour vous a forcément été appris par cœur afin de devenir inné.

C'est exactement pareil pour la programmation et donc pour la programmation PHP. Apprenez par cœur, encore et encore. Puis pratiquez, testez durant des jours, des semaines. Trouvez un projet utile à programmer avec cette nouvelle extension que vous

êtes en train d'étudier. Un projet que vous pourrez réutiliser afin de l'intégrer dans un autre projet. Par exemple, si vous étudiez les sockets, programmez un chat. Si vous étudiez l'extension Crack, programmez un vérificateur de mots de passe. Cherchez toujours un projet qui vous permet de rentabiliser le temps passé à étudier la nouvelle extension.

Extension non documentée

En ce qui concerne les extensions non documentées, il n'y a pas de méthode définie. Tout dépend de l'extension. Est-elle programmée en C puis compilée afin d'être intégrée dans les extensions PHP ? Est-elle programmée en PHP comme le sont les paquets PEAR ? Est-elle programmée en POO (Programmation orientée objet) ou non ? Tout cela a son importance. Le premier réflexe est d'utiliser l'introspection (ou réflexion) que vous avez étudiée dans le chapitre 8 avec les classes `Reflection*`. Puis, la plupart du temps, l'étape suivante est de se plonger dans les sources de l'extension, d'où l'intérêt de connaître également la programmation C.

Si l'introspection donne une bonne description des fonctions, alors la méthode devient la même que pour les extensions documentées : apprendre par cœur les fonctions et leur structure. Sinon, à l'étude des sources, toutes les fonctions de types publics sont répertoriées et sont à mémoriser, par cœur.

Dans les deux cas (réflexion ou non), les auteurs d'une extension lui cherchent une logique et se posent les questions suivantes :

- Comment structurer l'extension ?
- Quel peut être son objectif ?
- Que peut-on en faire ?
- Pour quel projet, quelle application, cette extension peut s'avérer utile ?

À vous de trouver les bonnes réponses. Si vous comprenez la logique d'une extension, vous avez déjà effectué la moitié du travail.

Et si je ne veux pas apprendre par cœur ?

Vous estimez que vous avez passé l'âge de déclamer des poésies devant le tableau noir ? Vous vous refusez à vérifier n'importe où n'importe quand que vous vous rappelez bien du fonctionnement de telle extension, de telle fonction ? Que le travail, c'est le travail, et que l'apprentissage d'une nouvelle extension n'a pas à gâcher votre journée ?

Vous avez choisi la voie la plus longue. Car, pour être certain de vos connaissances, il va vous falloir enchaîner quelques centaines de petits projets personnels, et mettre à profit le

maximum de temps libre pour cela. Oui : du temps libre, car ce ne sont pas ces petits projets personnels qui, bien que pédagogiques, vous nourriront. Vous allez passer trois jours chez votre belle-famille que vous ne supportez pas ? Parfait, vous savez ce qu'il vous reste à faire…

11.3 Quelques conseils d'ordre général

Apprendre de ses erreurs

Lorsque vous travaillez sur une extension, sur un de ces petits projets qui n'ont que l'apprentissage pour vocation, vous allez, en plus d'apprendre de nouvelles choses, réviser. Vous n'allez pas utiliser que les fonctions de votre extension. Vous allez chercher des données dans une base ou dans un fichier XML, concaténer des chaînes, élaborer des algorithmes, etc.

Toutes les manipulations que vous allez effectuer peuvent être source d'erreurs : une fonction mal utilisée avec le mauvais nombre d'arguments, un algorithme sans fin, des erreurs de casse et ainsi de suite.

Essayez de repérer quel est le genre d'erreurs que vous faites le plus souvent, et demandez-vous pourquoi. Est-ce parce que vous avez de mauvaises habitudes de programmation ? Ou parce que vous confondez deux mots sémantiquement proches ? Ou bien, saturez-vous la mémoire de votre serveur, jusqu'à sa capitulation ? De nombreuses raisons sont possibles, et il existe une solution pour chacune d'entres elles. À vous de la trouver afin de corriger vos erreurs tout en vous rendant plus performant, plus sûr de vous-même.

Voici un bon exercice de fond, à pratiquer régulièrement : reprenez vos anciens travaux et améliorez leurs algorithmes, épurez votre code, raccourcissez-le le plus possible ! Vous ne corrigerez pas d'erreurs, mais vous vous améliorerez dans votre manière de concevoir vos algorithmes. Vous les ferez ainsi plus courts dans vos projets suivants, et donc à la fois plus performants et plus rapides ; et, comme il y a moins de code, vous limiterez aussi le nombre d'erreurs possibles.

Utiliser toute l'aide disponible

Lorsque vous découvrez une nouvelle extension, il est illusoire de penser que vous vous en sortirez seulement avec la documentation fournie (encore faut-il qu'elle existe). Plus vous multiplierez vos sources de documentations, plus vous découvrirez des manières de procéder différentes. Vous découvrirez des algorithmes puissants auxquels vous n'aviez pas pensé, des façons originales d'utiliser certaines fonctions…

N'ayez pas peur d'écumer les forums de développeurs pour trouver la réponse à une question précise, ou à poser vous-même votre question. Il y aura toujours quelqu'un pour vous répondre, vous aider dans la résolution de votre problème, ou vous proposer une solution de secours.

Quand vous démarrez l'apprentissage d'une extension (ou de n'importe quoi d'autre, d'ailleurs), recherchez des tutoriels sur Internet et suivez-les. Vous verrez ainsi différentes manières d'utilisation, ouvrant de nouveaux horizons.

Avant même de commencer à apprendre, consultez les blogs de développeurs : il y en a forcément qui ont déjà pris la route que vous vous apprêtez à suivre. Vous trouverez des retours d'expériences, ce qui vous permettra de savoir si, oui ou non, telle extension mérite d'être apprise ou non… De plus, sur ces blogs, vous trouverez des informations que vous n'aviez pas prévues de chercher et qui vous ouvriront peut-être de nouvelles voies.

Tous les livres peuvent vous apprendre quelque chose en la matière : vous y trouverez les avis et les conseils de quelqu'un qui a vraiment exploré un sujet à fond. Ne dédaignez pas non plus les magazines spécialisés.

Et enfin, certainement la méthode la plus efficace et la plus agréable pour obtenir de l'aide, c'est de la demander à vos amis développeurs (si vous en avez). Avez-vous déjà passé un après-midi en compagnie d'un ami, à tripoter des lignes de codes, tous les deux face à la même machine ? Chacun apprend de l'autre et, surtout, vous partagez un agréable moment ensemble. Et tout le monde sait que l'on apprend mieux et plus vite, et que l'on est globalement plus performant quand on se sent bien.

11.4 Check-list

Ce chapitre, indispensable, vous a permis d'aborder les points suivants :

- ✔ se connaître ;
- ✔ savoir ce que l'on ne sait pas ;
- ✔ ce qu'il faut apprendre et pourquoi ;
- ✔ comment apprendre ;
- ✔ l'intérêt de collaborer (programmer à deux ou plus) ;
- ✔ l'intérêt de profiter de tous les médias mis à disposition.

12

Annexes

12.1 Webographie

PHP

- http://php.net/ : le site officiel de PHP toutes versions confondues ;
- http://nexen.net/ : toute l'actualité de PHP ;
- http://afup.org/ : Association française des utilisateurs de PHP ;
- http://developpez.com/ : le site des développeurs francophones ;
- www.developpez.net/forums/ : les forums du site developpez.com/ ;
- http://www.developpez.net/forums/showthread.php?t=6088 : optimisation PHP/MySQL ;
- http://snaps.php.net/ : les versions non stables de PHP dont PHP6 ;
- www.corephp.co.uk/ : l'équipe qui travaille sur PHP ;
- www.phpmvc.net/ : tout sur le MVC (*Model View Controller*) en PHP.

SQL

Normes

- http://savage.net.au/SQL/sql-92.bnf.html : la norme SQL92 version HTML ;
- http://savage.net.au/SQL/sql-99.bnf.html : la norme SQL99 version HTML ;
- http://savage.net.au/SQL/sql-2003-1.bnf.html : la norme SQL2003 version HTML (part 1) ;
- http://savage.net.au/SQL/sql-2003-2.bnf.html : la norme SQL2003 version HTML (part 2) ;
- http://savage.net.au/SQL/sql-2003-core-features.html : pour les perfectionnistes ;
- http://savage.net.au/SQL/sql-2003-noncore-features.html : et les insomniaques.

MySQL

- www.mysql.com/ : le site officiel du SGBD en anglais ;
- www-fr.mysql.com/ : le même que le précédent et en français ;
- http://dev.mysql.com/ : la zone des développeurs avec MySQL6 en téléchargement.

PostgreSQL

- http://www.postgresql.org/ : le site officiel de PostgreSQL ;
- http://www.postgresqlfr.org/ : le site français pour PostgreSQL ;
- http://docs.postgresqlfr.org/ : la documentation française de PostgreSQL.

Linux

- www.debian.org/ : le site français le plus complet sur Debian ;
- www.debian-fr.org/ : le site sur Debian et aussi GNU/Linux

- **www.gentoo.org/** : le site officiel de Gentoo ;
- **www.gentoo.org/doc/fr/** : la documentation Gentoo ;
- **www.gentoofr.org/** : le site Gentoo francophone ;
- **http://fluxbox.sourceforge.net/** : le bureau graphique pour Linux, très léger et très rapide.

Autres

- **www.cru.fr/** : le Comité réseau des universités contient de très nombreuses informations et de liens, notamment en ce qui concerne LDAP (**www.cru.fr/ldap/**) ;
- **www.openldap.org/** : le site officiel OpenLDAP ;
- **http://www.openlaszlo.org/** : Flash version OpenSource, pour les codeurs acharnés ;
- **http://tools.ietf.org/html/rfc4517** : LDAP Schema for User Applications où vous trouverez, entre autres, la signification des abréviations (c pour country, etc.) et comment les utiliser ;
- **http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html** : la liste des abréviations des pays en deux lettres, trois lettres et leur code numérique ;
- **www.fpdf.org/** : le site où télécharger FPDF et se documenter sur le sujet ;
- **http://www.imagemagick.org/script/index.php** : le site où télécharger ImageMagick ;
- **www.magickwand.org/** : le site où télécharger MagickWand ;
- **http://milw0rm.com/** : la description de failles, et une mise à jour hebdomadaire ;
- **http://www.devshed.com/** : un site web très bien fourni pour le webmastering
- **http://www.pantz.org/** : diverses informations sur beaucoup de domaines en informatique.

12.2 Fonctions des extensions

Voici la liste de toutes les fonctions étudiées dans cet ouvrage.

APC

Tableau 12.1 : APC

Les fonctions	Les arguments	Leur description
`bool apc_add`		Met une variable en cache, sauf si elle a déjà été stockée.
	`chaîne $key, mixe $valeur [, entier $duree_de_vie ]`	

Tableau 12.1 : APC

`bool apc_clear_cache`		Efface le cache APC. Si `$cache_type` vaut "`user`", le cache utilisateur sera nettoyé ; sinon, le cache système (les fichiers mis en cache) sera nettoyé.
	`chaîne $cache_type`	
`bool apc_define_constants`		Définit un jeu de constantes. Le paramètre `$key` est utilisé comme identifiant du jeu de constantes et sert à les récupérer en utilisant la fonction `apc_load_constants()`.
	`chaîne $key, tableau $constantes [, bool $case_sensitive ]`	
`bool apc_delete`		Efface une variable du cache. Le paramètre `$key` correspond à celui utilisé pour stocker la variable.
	`chaîne $key`	
`mixe apc_fetch`		Récupère une variable depuis le cache. Le paramètre `$key` correspond à celui utilisé pour stocker la variable.
	`chaîne $key`	
`bool apc_load_content`		Récupère un jeu de constantes depuis le cache.
	`chaîne $key [, bool $case_sensitive ]`	
`bool apc_store`		Met une variable en cache.
	`chaîne $key, mixe $valeur [, entier $duree_de_vie ]`	

Bcompiler

Tableau 12.2 : Bcompiler

Les fonctions	Les arguments	Leur description
`bool bcompiler_load`		Lit les données depuis le fichier compressé bzippé `$filename` et crée les classes depuis le bytecode.
	`chaîne $filename`	
`bool bcompiler_read`		Lit les données depuis un fichier ouvert représenté par le descripteur `$filehandle` et crée les classes depuis le bytecode.
	`resource $filehandle`	

Tableau 12.2 : Bcompiler

`bool bcompiler_write_class`		Lit le bytecode d'une classe existante nommée `$className` depuis PHP et l'écrit dans le fichier ouvert désigné par le descripteur `$filehandle`.
	`resource $filehandle, chaîne $className [, chaîne $extends ]`	
`bool bcompiler_write_file`		Compile le fichier `$filehandle` et écrit le résultat dans le fichier `$filename`.
	`resource $filehandle, chaîne $filename`	
`bool bcompiler_write_footer`		Indique la fin des données à compiler.
	`resource $filehandle`	
`bool bcompiler_write_function`		Écrit la fonction `$functionname` dans le fichier `$filehandle`.
	`resource $filehandle, chaîne $functionname`	
`bool bcompiler_write_functions_from_file`		Recherche toutes les fonctions présentes dans le fichier `$filename` et écrit leur bytecode dans le fichier `$filehandle`.
`bool bcompiler_write_header`		Écrit l'en-tête Bcompiler.
	`resource $filehandle [, chaîne $write_ver ]`	

BZIP2

Tableau 12.3 : BZIP2

Les fonctions	Les arguments	Leur description
`entier bzclose`		Ferme un fichier ouvert avec la fonction `bzopen()`.
	`resource $bzo`	
`mixte bzcompress`		Compresse une chaîne de données en fonction d'un taux de compression et d'un facteur de travail.
	`chaine $donnees [, entier $taux_compression [, entier $facteur_de_travail]]`	

Tableau 12.3 : BZIP2

`mixte bzdecompress`		Décompresse une chaîne de données compressée avec la méthode `bzcompress()`.
	`chaine $donnees_compressees [, entier $small]`	
`tableau bzerror`		Retourne un numéro et un message d'erreur stockés dans un tableau.
	`ressource $bz`	
`entier bzerrno`		Retourne un numéro d'erreur BZIP2.
	`ressource $bz`	
`chaine bzerrstr`		Retourne un message d'erreur BZIP2.
	`ressource $bz`	
`ressource bzopen`		Ouvre un fichier compressé au format BZIP2.
	`chaine $nom_fichier, chaine $mode_ouverture`	
`chaine bzread`		Lit le contenu d'un fichier BZIP2 en fonction d'un nombre de caractères déterminé.
	`ressource $bzo [, entier $nb_caract_max]`	
`entier bzwrite`		Écrit des données dans un fichier BZIP2 ouvert via la fonction `bzopen()`.
	`ressource $bzo, chaine $donnees [, entier $nb_caract_max]`	

Classkit

Tableau 12.4 : Classkit

Les fonctions	Leurs arguments	Leur description
`bool classkit_method_add`		Ajoute dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux]`	

Tableau 12.4 : Classkit

`bool classkit_method_redefine`		Redéfinit dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux]`	
`classkit_method_remove`		Supprime dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode`	
`bool classkit_method_rename`		Renomme dynamiquement une méthode dans une classe.
	`chaine $nom_classe, chaine $nom_methode, chaine $nouveau_nom_methode`	

DOM

Fonctions de l'objet DOMDocument

Tableau 12.5 : DOM

Les fonctions	Les arguments	Leur description
`DOMElement createElement`		Crée une nouvelle instance de `DOMElement`. Pour intégrer ce nœud dans un document, utiliser la fonction `DOMNode::appendChild()`.
	`chaîne $name [, chaîne $value]`	
`DOMElement createElementNS`		Crée un nouveau nœud avec un espace de nom. Pour intégrer ce nœud dans un document, utiliser la fonction `DOMNode::appendChild()`.
	`chaîne $namespaceURI, chaîne $qualifiedName [, chaîne $value]`	
`DOMText createTextNode`		Crée un nouveau nœud texte. Pour intégrer ce nœud dans un document, utiliser la fonction `DOMNode::appendChild()`.
	`chaîne $content`	
`DOMNodeList getElementsByTagName`		Retourne une instance de `DOMNodeList` contenant tous les éléments du document dont le nom de balise vaut `$name`.
	`chaîne $name`	

Tableau 12.5 : DOM

`DOMNodeList getElementsByTagNameNS`		Retourne une instance de `DOMNodeList` contenant tous les éléments du document dont l'espace de nom vaut `$namespaceURI` et le nom vaut `$localName`.
	`chaîne $namespaceURI, chaîne $localName`	
`mixe load`		Charge un fichier XML. Retourne `TRUE` ou `FALSE` en cas de succès ou d'échec. Si appelée statiquement, retourne un objet `DOMDocument`.
	`chaîne $filename [, entier $option]`	
`mixe save`		Enregistre un fichier XML depuis une représentation DOM. Retourne le nombre d'octets écrits ou `FALSE` en cas d'erreur.
	`chaîne $filename [, entier $option]`	

Fonctions de l'objet DOMElement

Tableau 12.6 : Fonctions de l'objet DOMElement

Les fonctions	Les arguments	Leur description
`chaîne getAttribute`		Retourne la valeur de l'attribut dont le nom est `$name`.
	`chaîne $name`	
`bool setAttribute`		Crée un attribut ayant `$name` pour nom et `$value` pour valeur.
	`chaîne $name, chaîne $value`	
`void setAttributeNS`		Crée un attribut avec `$namespaceURI` comme espace de nom, `$qualifiedname` comme nom, et `$value` comme valeur.
	`chaîne $namespaceURI, chaîne $qualifiedname, chaîne $value`	

Fonctions de l'objet DOMNode

Tableau 12.7 : Fonctions de l'objet DOMNode

Les fonctions	Les arguments	Leur description
`DOMNode appendChild`		Ajoute un nœud enfant à la fin de la liste des nœuds enfants déjà existants.
	`DOMNode $nouveauNoeud`	

Tableau 12.7 : Fonctions de l'objet DOMNode

`bool hasChildNodes`		Vérifie si un nœud a des enfants.
	Aucun argument.	

FPDF

Tableau 12.8 : FPDF

Les fonctions	Les arguments	Leur description
`AddPage`		Ajoute une nouvelle page au document. Le paramètre **`$orientation`** permet de choisir le sens de la page : '**`P`**' pour Portrait ou '**`L`**' pour Paysage.
	`[chaine $orientation]`	
`AliasNbPages`		Définit un alias pour le nombre total de pages.
	`[chaine $alias]`	
`Cell`		Écrit une chaîne de caractères dans une cellule et place cette cellule aux coordonnées courantes.
	`flottant $largeur [, flottant $hauteur [, chaine $texte [, mixe $bordure [, entier $interligne [, chaine $alignement [, entier $remplissage [, mixe $lien ]]]]]]]`	
`Close`		Termine le document PDF. Cette fonction est appelée explicitement par la fonction **`Output`**.
`GetStringWidth`		Permet de connaître la longueur d'une chaîne de caractères avec la police de caractères courante.
	`Chaine $chaine`	
`Image`		Place une image dans la page.
	`chaine $fichier, flottant $x, flottant $y [, flottant $largeur [, flottant $hauteur [, chaine $type [, mixe $lien ]]]]`	

Tableau 12.8 : FPDF

`Line`		Trace une ligne entre deux points.
	`flottant $x1, flottant $y1, flottant $x2, flottant $y2`	
`Ln`		Revient à la ligne (retour chariot).
	`flottant $interligne`	
`Multicell`		Permet de créer un bloc de texte.
	`flottant $largeur, flottant $hauteur, chaine $texte [, mixe $bordure [, chaine $alignement [, entier $remplissage ]]]`	
`Output`		Envoie le document vers la sortie standard (le navigateur, le plus souvent) sous forme de chaîne, ou enregistre le fichier.
	`[chaine $nom [, chaine $destination ]]`	
`PageNo`		Renvoie le numéro de page courant.
`Rect`		Dessine un rectangle.
	`flottant $x, flottant $y, flottant $largeur, flottant $hauteur [, chaine $style ]`	
`SetAuthor`		Définit l'auteur du document.
	`chaine $auteur`	
`SetCreator`		Définit le nom du créateur. En général, le nom de l'application qui a généré le nom du document.
	`chaine $createur`	
`SetDrawColor`		Définit la couleur courante des tracés et contours.
	`entier $rouge, entier $vert, entier $bleu`	
`SetFillColor`		Définit la couleur de remplissage courante.
	`entier $rouge, entier $vert, entier $bleu`	

Tableau 12.8 : FPDF

`SetFont`		Définit la police de caractères courante, ainsi que sa taille.
	`chaine $famille [, chaine $style [, flottant $taille ]]`	
`SetKeywords`		Associe des mots-clés au document.
	`chaine $motClef`	
`SetLineWidth`		Fixe l'épaisseur des traits. La valeur par défaut est 0.2.
	`flottant $epaisseur`	
`SetSubject`		Définit le sujet du document.
	`chaine $sujet`	
`SetTitle`		Définit le titre du document.
	`chaine $titre`	
`SetX`		Change la coordonnée x de la position courante.
	`flottant $x`	
`SetY`		Change la coordonnée y de la position courante.
	`flottant $y`	
`SetXY`		Change les coordonnées de la position courante.
	`flottant $x, flottant $y`	

GD2

Tableau 12.9 : GD2

Les fonctions	Les arguments	Leur description
`tableau getimagesize`		Détermine la taille de l'image fournie et en retourne les dimensions, le type d'image et une chaîne type **`height/width`** à placer dans une balise HTML **`IMG`** normale et le type de contenu HTTP correspondant.
	`chaîne $filename [, tableau &$imageinfo]`	

Tableau 12.9 : GD2

`entier imagecolorallocate`		Retourne un identifiant de couleur RVB.
	`resource $image, entier $rouge, entier $vert, entier $bleu`	
`bool imagecopyresized`		Copie une partie de l'image source dans une image de destination.
	`resource $image_destination, resource $image_source, entier $dst_x, entier $dst_y, entier $src_x, entier $src_y, entier $dst_largeur, entier $dst_hauteur, entier $src_largeur, entier $src_hauteur`	
`resource imagecreate`		Crée une nouvelle image à palette.
	`entier $largeur, entier $hauteur`	
`resource imagecreatefromgif`		Crée une nouvelle image GIF à partir d'un fichier.
	`chaîne $filename`	
`resource imagecreatefromjpeg`		Crée une nouvelle image JPEG à partir d'un fichier.
	`chaîne $filename`	
`resource imagecreatefrompng`		Crée une nouvelle image PNG à partir d'un fichier.
	`chaîne $filename`	
`resource imagecreatetruecolor`		Crée une nouvelle image trueColor.
	`entier $largeur, entier $hauteur`	
`resource imagecreatefromwbmp`		Crée une nouvelle image WBMP à partir d'un fichier.
	`chaîne $filename`	
`bool imagedestroy`		Libère la mémoire occupée par l'image `$image`.
	`resource $image`	

Tableau 12.9 : GD2

`bool imagefilledrectangle`		Dessine un rectangle rempli avec la couleur `$couleur`.
	`resource $image, entier $x1, entier $y1, entier $x2, entier $y2, entier $couleur`	
`bool imagegif`		Envoie une image GIF vers un navigateur ou un fichier.
	`resource $image [, chaîne $filename ]`	
`bool imagejpeg`		Envoie une image JPEG vers un navigateur ou un fichier.
	`resource $image [, chaîne $filename ]`	
`bool imageline`		Dessine une ligne de couleur `$couleur`.
	`resource $image, entier $x1, entier $y1, entier $x2, entier $y2, entier $couleur`	
`bool imagepng`		Envoie une image PNG vers un navigateur ou un fichier.
	`resource $image [, chaîne $filename ]`	
`bool imagesetthickness`		Spécifie l'épaisseur d'un trait.
	`resource $image, entier $epaisseur`	
`bool imagechaîne`		Dessine une chaîne de caractères aux coordonnées spécifiées, et de couleur `$couleur`.
	`resource $image, entier $font, entier $x, entier $y, entier $chaîne, entier $couleur`	
`bool imagewbmp`		Envoie une image WBMP vers un navigateur ou un fichier.
	`resource $image [, chaîne $filename ]`	

LDAP

Tableau 12.10 : LDAP

Les fonctions	Les arguments	Leur description
`bool ldap_add`		Ajoute une entrée dans un dossier LDAP.

	`resource $link_identifier,`
	`chaîne $dn,`
	`tableau $entry`	

`bool ldap_bind`		Authentification au serveur LDAP avec le RDN et le mot de passe spécifiés.

	`resource $link_identifier`
	`[, chaîne $bind_rdn`
	`[, chaîne $bind_password]]`	

`entier ldap_close`		Alias de `ldap_unbind()`.
	`resource $id_connexion`	
`resource ldap_connect`		Établit une connexion avec un serveur LDAP.
	`[chaîne $hostname` `[, entier $port]]`	
`bool ldap_delete`		Supprime une entrée du répertoire.
	`ressource $id_connexion,` `chaîne $dn`	
`entier ldap_errno`		Retourne un numéro d'erreur LDAP.
	`ressource $id_connexion`	
`chaine ldap_error`		Retourne une chaîne qui explique l'erreur LDAP générée.
	`ressource $id_connexion`	
`chaine ldap_err2str`		Récupère un numéro d'erreur et le transforme en chaîne.
	`entier $errno`	
`chaîne ldap_first_attribute`		Retourne le premier attribut d'un élément.

	`resource $id_connexion,`
	`resource $result_entry_id,`
	`entier &$ber_identifier`	

Tableau 12.10 : LDAP

`ressource ldap_first_entry`		Retourne la première entrée de l'arbre LDAP.
	`ressource $id_connexion,` `ressource $result`	
`ressource ldap_first_reference`		Retourne la première référence de l'arbre LDAP.
	`ressource $id_connexion,` `ressource $result`	
`bool ldap_free_result`		Libère le résultat afin de libérer de la mémoire RAM.
	`ressource $id_resultat`	
`tableau ldap_get_entries`		Retourne sous forme d'un tableau les entrées d'un résultat de recherche.
	`resource $id_connexion,` `resource $id_resultat`	
`bool ldap_modify`		Modifie les entrées d'un arbre LDAP.

	`ressource $id_connexion,`
	`chaîne $dn,`
	`tableau $entrees`	

`resource ldap_search`		Effectue une recherche dans un arbre LDAP.

	`ressource $id_connexion,`
	`chaîne $base_dn,`
	`chaîne $filtre`
	`[, tableau $attributs`
	`[, entier $attrs_seuls`
	`[, entier $taille_max`
	`[, entier $timelimit`
	`[, entier $deref]]]]]`	

`bool ldap_set_option`		Modifie la valeur d'une option LDAP.

	`ressource $id_connexion,`
	`entier $option,`
	`mixte $nouvelle_valeur_option`	

MagickWand

Tableau 12.11 : MagickWand

Les fonctions	Leurs arguments	Leur description
`void DrawBezier`		Dessine une courbe de Bézier en utilisant la couleur et le style de trait courants. Le tableau `$poentier` doit contenir au minimum six nombres (soit les coordonnées de trois points).
	`DrawingWand $d, Tableau $poentier`	
`void DrawCircle`		Dessine un cercle sur l'image.
	`DrawingWand $d, float $centreX, float $centreY, float $RayonX, float $RayonY`	
`void DrawSetFillAlpha`		Définit l'opacité de la couleur de remplissage.
	`DrawingWand $d, Float $alpha`	
`void DrawSetFillColor`		Définit la couleur de remplissage.
	`DrawingWand $d, PixelWand $p`	
`void DrawSetStrokeColor`		Définit la couleur des traits et des contours.
	`DrawingWand $d, PixelWand $p`	
`void DrawSetStrokeAlpha`		Définit l'opacité des traits et des contours.
	`DrawingWand $d, float $alpha`	
`void DrawSetStrokeWidth`		Définit l'épaisseur des traits et des contours.
	`DrawingWand $d, float $epaisseur`	
`bool MagickAddImage`		Copie l'image `$source` sur l'image `$destination`.
	`DrawingWand $destination, DrawingWand $source`	
`bool MagickBlurImage`		Applique un effet de flou.
	`DrawingWand $d, float $radius, float $sigma [, entier channel_type ]`	

Tableau 12.11 : MagickWand

`bool MagickEchoImageBlob`		Extrait le BLOB (*Binary Large OBject*) dans l'image contenue par `$d` et l'envoie vers la sortie standard qui, sur le Web, devrait être le navigateur de l'utilisateur.
	`DrawingWand $d`	
`bool MagickFlattenImages`		Aplatit la séquence d'image contenue par `$d`.
	`DrawingWand $d`	
`chaîne MagickGetExceptionChaîne`		Renvoie une chaîne de caractères décrivant les erreurs ou exceptions survenues lors de l'exécution d'une méthode de l'API MagickWand.
	`DrawingWand $d`	
`bool MagickGaussianBlurImage`		Applique un effet de flou gaussien à l'image.
	`DrawingWand $d, float $radius, float $sigma [, entier $channel_type ]`	
`float MagickGetImageHeight`		Renvoie la hauteur de l'image.
	`DrawingWand $d`	
`float MagickGetImageWidth`		Renvoie la largeur de l'image.
	`DrawingWand $d`	
`bool MagickMotionBlurImage`		Applique un effet de flou directionnel sur l'image.
	`DrawingWand $d float $radius, float $sigma, float $angle`	
`bool MagickNegateImage`		L'image passe en négatif.
	`DrawingWand $d [, bool $luminanceSeulement [, entier channel_type ]]`	
`bool MagickOilPaentierImage`		Applique un effet de peinture à l'huile sur l'image.
	`DrawingWand $d float $radius`	
`bool MagickRadialBlurImage`		Applique un effet de flou radial à l'image.
	`DrawingWand $d, float $angle`	

Tableau 12.11 : MagickWand

`bool MagickRaiseImage`		Applique un effet dit de biseautage, où des ombres et des lumières sont ajoutées à l'image, donnant vaguement l'impression d'un bouton.
	`DrawingWand $d, float $largeur, float $hauteur, entier $x, entier $y, bool $raise`	
`bool MagickScaleImage`		Redimensionne l'image.
	`DrawingWand $d, float $largeur, float $hauteur`	
`bool MagickSetImageCompressionQuality`		Définit le taux de compression de l'image.
	`DrawingWand $d, float $qualite`	
`bool MagickSwirlImage`		Applique un effet tourbillon sur l'image.
	`DrawingWand $d, float $degres`	
`bool MagickWriteImage`		Enregistre l'image sur le disque.
	`DrawingWand $d, chaîne $filename`	
`DrawingWand NewDrawingWand`		Crée un objet `DrawingWand`.
`MagickWand NewMagickWand`		Crée un objet `MagickWand`.
`PixelWand NewPixelWand`		Crée un objet `PixelWand`.
	`DrawingWand $d,`	
`bool PixelSetColor`		Définit une couleur grâce à une chaîne de caractères (exemples : "`blue`", "`#0000ff`", "`rgb(0,0,255)`", "`cmyk(100,100,100,10)`".
	`DrawingWand $d, Chaîne $couleur`	

PDO (PHP Data Object)

Tableau 12.12 : PDO

Les fonctions	Leurs arguments	Leur description
`PDO __construct`		Construit une nouvelle instance permettant la connexion au SGBD voulu. Exemple : `$oL = new pdo($dsn);`
	`chaîne $dsn [, chaîne $username [, chaîne $password [, tableau $driver_options]]]`	
`entier PDO->exec`		Exécute une requête et retourne le nombre de lignes affectées.
	`chaîne $statement`	
`PDOStatement PDO->query`		Exécute une requête et retourne un jeu de résultats en tant qu'objet `PDOStatement`. Exemple : `$qqch = $oL->exec($req_sql);`
	`chaîne $statement`	
`tableau fetchAll`		Retourne sous forme de tableau les résultats de la recherche après exécution de `query()` Exemple : `$tab = $oL->fetchAll();`
	`[ entier $fetch_style [, entier $column_index]]`	

Reflection

Tableau 12.13 : Reflection

Les fonctions	Leurs arguments	Leur description
`Reflection::export`		Affiche un tableau qui contient toute l'architecture de la classe `$nom_classe`.
	`mixte $nom_classe, bool $retour`	
`chaîne ReflectionFunction::getDocComment`		Retourne le commentaire associé à la fonction sur laquelle est effectuée la réflexion. Le commentaire doit respecter le formatage PHPDoc.
`entier ReflectionFunction::getEndLine`		Retourne le numéro de ligne du fichier sur laquelle se termine la fonction (accolade fermante `'}'`).

Tableau 12.13 : Reflection

`chaîne ReflectionFunction::getFileName`	Retourne le nom du fichier dans lequel se trouve la fonction.
`chaîne ReflectionFunction::getName`	Retourne le nom de la fonction sur laquelle est effectuée la réflexion.
`entier ReflectionFunction::getNumberOfParameters`	Retourne le nombre de paramètres (obligatoires et optionnels) nécessaires à la fonction.
`entier ReflectionFunction::getNumberOfRequiredParameters`	Retourne le nombre de paramètres obligatoires nécessaires à la fonction.
`entier ReflectionFunction::getStartLine`	Retourne le numéro de ligne du fichier sur laquelle commence la définition de la fonction.
`tableau ReflectionFunction::getStaticVariables`	Retourne sous forme de tableau les variables de type statique de la fonction sur laquelle est effectuée la réflexion.
`bool ReflectionFunction::isEntierernal`	Détermine si une fonction a été définie par un utilisateur (retourne `true`) ou si celle-ci est interne à PHP (retourne `false`).
`bool ReflectionFunction::isUserDefined`	Détermine si une fonction est interne à PHP (retourne `true`) ou si celle-ci a été définie par un utilisateur (retourne `false`).
`public ReflectionClass getDeclaringClass`	Crée une réflexion sur la classe dans laquelle la méthode (sur laquelle vous faites la réflexion) est définie.
`public bool isAbstract`	Retourne `true` si la méthode est définie en tant qu'abstraite.
`public bool isConstructor`	Retourne `true` si la méthode est un constructeur.
`public bool isDestructor`	Retourne `true` si la méthode est un destructeur.

Tableau 12.13 : Reflection

`public bool isFinal`	Retourne `true` si une méthode est définie en tant que finale.
`public bool isPrivate`	Retourne `true` si une méthode a un accès privé.
`public bool isProtected`	Retourne `true` si une méthode a un accès protégé.
`public bool isPublic`	Retourne `true` si une méthode a un accès public.
`public bool isStatic`	Retourne `true` si une méthode est définie en tant que statique.

Runkit

Tableau 12.14 : Runkit

Les fonctions	Leurs arguments	Leur description
`bool runkit_class_adopt`	`chaîne $nom_classe_enfant, chaîne $nom_classe_parent`	Permet à une classe enfant d'hériter d'une classe parente. Il s'agit bien d'un héritage dynamique.
`bool runkit_class_emancipate`	`chaîne $nom_classe_parent`	Permet à une classe parente de se libérer de toutes ses classes enfants.
`bool runkit_constant_add`	`chaîne $nom_constante, mixte $valeur`	Ajoute dynamiquement une constante.
`bool runkit_constant_redefine`	`chaîne $nom_constante, mixte $nouvelle_valeur`	Redéfinit dynamiquement une constante.

Tableau 12.14 : Runkit

`bool runkit_constant_remove`		Supprime dynamiquement une constante.
	`chaîne $nom_constante`	
`bool runkit_function_add`		Ajoute dynamiquement une fonction.
	`chaîne $nom_fonction, chaîne $liste_arguments, chaîne $code`	
`bool runkit_function_redefine`		Redéfinit dynamiquement une fonction.
	`chaîne $nom_fonction, chaîne $liste_arguments, chaîne $code`	
`bool runkit_function_remove`		Supprime dynamiquement une fonction.
	`chaîne $nom_fonction`	
`bool runkit_function_rename`		Renomme dynamiquement une fonction.
	`chaîne $nom_fonction, chaîne $nouveau_nom`	
`bool runkit_method_add`		Ajoute dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux]`	
`bool runkit_method_redefine`		Redéfinit dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode, chaîne $args, chaîne $code [, entier $drapeaux]`	
`bool runkit_method_remove`		Supprime dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode`	
`bool runkit_method_rename`		Renomme dynamiquement une méthode dans une classe.
	`chaîne $nom_classe, chaîne $nom_methode, chaîne $nouveau_nom`	

ZIP

Tableau 12.15 : ZIP

Les fonctions	Leurs arguments	Leur description
`bool ZipArchive::addFile`		Ajoute un fichier dans une archive ZIP.
	`chaîne $nom_fichier [, chaine $nom_dans_archive]`	
`bool ZipArchive::addFromChaîne`		Ajoute une chaîne (`$contenu`) dans un fichier (`$nom_dans_archive`) qui sera placé dans l'archive ouverte.
	`chaine $nom_dans_archive , chaine $contenu`	
`bool ZipArchive::close`		Ferme une archive ZIP ouverte via la fonction `addFile()`.
	`void`	
`mixte ZipArchive::extractTo`		Extrait un ou plusieurs fichiers d'une archive ZIP.
	`chaîne $rep_dest [, mixte $entrees]`	
`chaîne ZipArchive::getArchiveComment`		Retourne le commentaire associé à une archive ZIP.
`mixte ZipArchive::locateName`		Localise l'emplacement d'un fichier dans une archive ZIP.
	`chaîne $nom_fichier [, entier $drapeaux]`	
`mixte ZipArchive::open`		Ouvre une archive ZIP.
	`chaîne nom_fichier [, entier drapeaux]`	
`bool ZipArchive::renameIndex`		Renomme l'index d'un fichier situé dans une archive ZIP.
	`entier $index, chaîne $nouveau_nom`	
`bool ZipArchive::renameName`		Renomme le nom d'un fichier situé dans une archive ZIP.
	`chaîne $nom_actuel, chaîne $nouveau_nom`	
`mixte ZipArchive::setArchiveComment`		Dépose un commentaire associé à une archive ZIP.
	`chaîne $commentaire`	

ZLIB

Tableau 12.16 : ZLIB

Les fonctions	Les arguments	Leur description
`bool gzclose`		Ferme un fichier GZIP ouvert via la fonction `gzopen()`.
	`ressource $gzo`	
`chaîne gzcompress`		Compresse les données selon un taux de compression.
	`chaîne $donnees` `[, entier $taux_compression]`	
`chaîne gzdeflate`		Compresse une chaîne via une méthode qui utilise simultanément l'algorithme LZ77 et le codage de Huffman.
	`chaîne $donnees` `[, entier $niveau]`	
`chaîne gzgetc`		Retourne un caractère des données contenues dans le fichier ZLIB.
	`ressource $zp`	
`chaîne gzgets`		Retourne une chaîne de caractères des données contenues dans le fichier ZLIB.
	`ressource $zp` `, entier $length`	
`chaîne gzgetss`		Retourne une chaîne de caractères des données contenues dans le fichier ZLIB et en supprimant les balises HTML.

	`ressource $zp,`
	`entier $length`
	`[, chaîne $balises_permises]`	

`chaîne gzinflate`		Décompresse une chaîne compressée via la fonction `gzdeflate()`.
	`chaîne $data` `[, entier $taille]`	
`ressource gzopen`		Ouvre un fichier compressé avec l'algorithme ZLIB.

	`chaîne $nom_fichier,`
	`chaîne $mode`
	`[, entier $use_include_path]`	

`entier gzpassthru`		Lit les données non lues dans un fichier ZLIB.
	`resource $zp`	

Tableau 12.16 : ZLIB

`chaîne gzread`		Lit le contenu d'un fichier ZLIB.
	`ressource $gzo , entier $taille`	
`bool gzrewind`		Place le curseur au début du fichier ZLIB.
	`ressource $gzo`	
`entier gzseek`		Place le curseur à un endroit précis du fichier ZLIB.
	`ressource $gzo, entier $emplacement`	
`entier gztell`		Retourne la position à laquelle se trouve le curseur dans le fichier ZLIB.
	`resource $zp`	
`chaîne gzuncompress`		Décompresse une chaîne compressée via la fonction `gzcompress()`.
	`chaîne $donnees [, entier $taille]`	
`entier gzwrite`		Écrit dans un fichier ZLIB.
	`ressource $gzo, chaîne $chaine [, entier $taille]`	

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

W

X

Z

Notes

Notes

Notes

Composé en France par Jouve
11, bd de Sébastopol - 75001 Paris