CD Le plus grand magazine sur PHP au monde

COURS VIDÉO SUR L'API GOOGLE MAPS !

# phpsolutions

Nouvelles technologies et solutions pour les développeurs PHP

PHP N° 2/2010 (38) ISSN 1731-4593 Prix 7,5 EUR CD offert France Metro : 7,50 EUR DOM : 8,80 EUR MAR : 80 MAD TOM/S : 990 XPF

## ADAPTEZ VOS SITES PHP SUR MOBILES AVEC HAWHAW

phpsolutions

**OUTILS**

**UTILISER ORACLE AVEC PHP**
INTRODUCTION ET INSTALLATION

**PROJETS**

**RÉALISEZ VOTRE SERVEUR D'OBJETS AVEC PHP**
LA SOLUTION DE COMMUNICATION HAUTE PERFORMANCE

**SUR LE CD**

**EN EXCLUSIVITÉ 2 COURS VIDÉOS !**
**API GOOGLE MAPS**
**PDO AVANCÉ**

**INTRODUCTION AU DOM AVEC PHP**
CONSTRUIRE ET MANIPULER DES DOCUMENTS XML DANS UNE ARCHITECTURE OBJET

**MÉTHODES AGILES**
DÉVELOPPER PLUS EFFICACEMENT VOS PROJETS

**RÉALISER UN INJECTEUR DE DÉPENDANCES**
CONCEPTS ORIENTÉ OBJET DE PHP 5

**STRATÉGIES D'OPTIMISATION POUR PHP**
VOS CODES SANS FAILLES

L 14389 - 38 - F: 7,50 € - RD

# TABLE DES MATIÈRES

## VARIA



## OUTILS

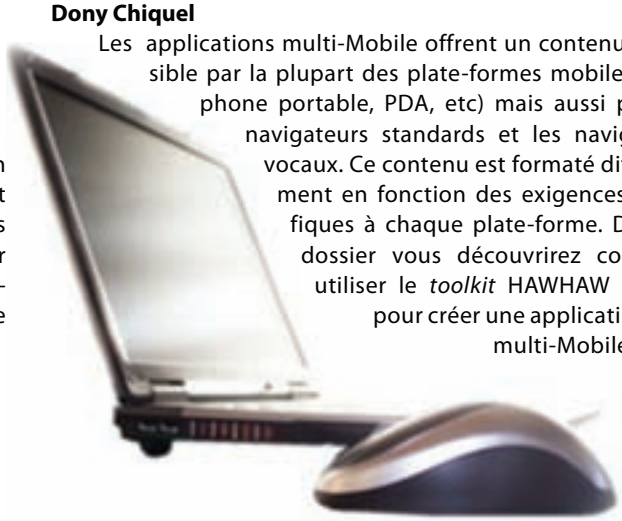


## PROJETS



## DOSSIER





## PRATIQUE



## E-COMMERCE

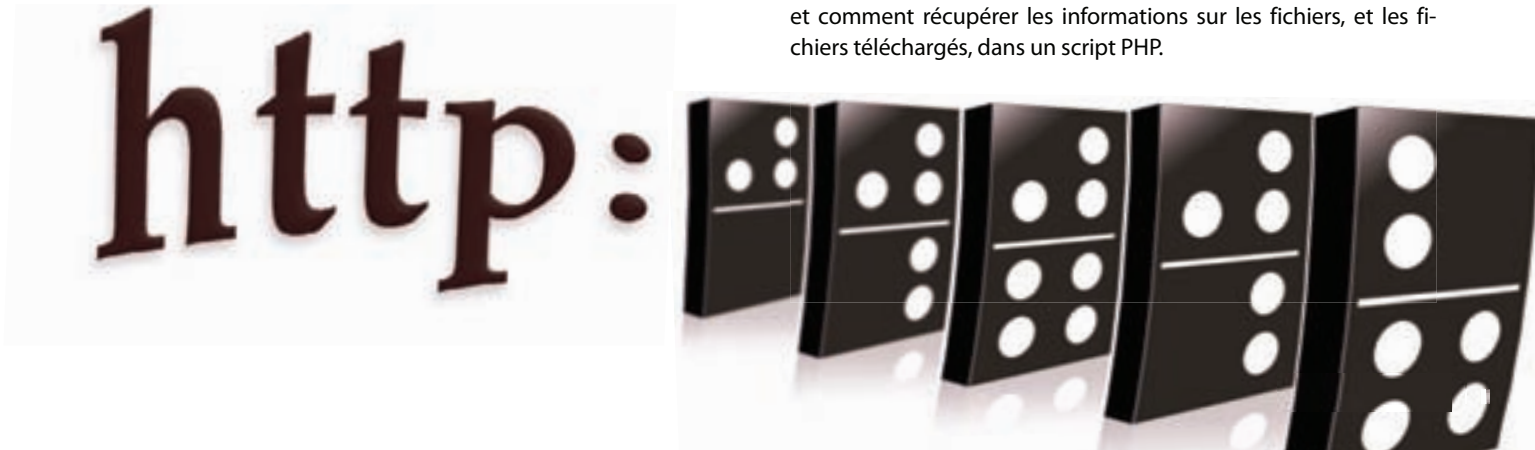


## FICHE TECHNIQUE



## POUR LES DÉBUTANTS





**PHP 5.2.12 & PHP 5.3.1**
Deux nouvelles versions de PHP viennent de sortir. Ces deux versions correspondent à une série de corrections de différents types de *bugs*. Il est important d'effectuer les mises à jours pour garder une version de PHP impeccable.

**CrawlProtect**
*Crawlprotect* est une nouvelle API destinée à protéger votre site internet. Cette API est réalisée en PHP et ne possède pas de base de données. Les caractéristiques proposées vont bloquer les tentatives d'injection de code, les injections SQL, les visites de robots connus comme étant des *Badbots* (robots utilisés par les *hackers*), les aspirateurs de site, les tentatives d'éxécution de commande *shell*.
*http://www.crawlprotect.com/fr*

**GreenSQL**
*GreenSQL* est un pare-feu applicatif SQL pour bases de données MySQL ou PostgreSQL. La distribution est sous licence SQL et va vous protéger contre de nombreux types d'attaques comme les injections SQL, les *Cross-site scripting*... Son fonctionnement est basé en mode *reverse-proxy*.
*http://www.greensql.net*

**Compilation de PHP**
Les expressions régulières sont très utilisées en PHP. Elles peuvent vous permettre d'effectuer par exemple la recherche de caractères alpha-numériques. Cependant lors de l'installation et la compilation de PHP sur votre serveur avec une version de PCRE personnalisée, vous pouvez rencontrer des problèmes avec les expressions régulières si vous désirez utiliser les caractères accentués (caractères français).
*http://blog.astrumfutura.com/archives/430-PCRE-Regex-Word-Matching-w-vs-a-zA-Z0-9_.html*

**PHP et XHProf**
*XHProf* est un profileur hiérarchique pour le langage PHP. Il vous permet d'obtenir des chiffres de temps, temps CPU, utilisation de la mémoire, etc. Le résultat obtenu vous permettra d'analyser des valeurs. Le tutoriel de Lorenzo montre une utilisation assez détaillée avec des exemples et des représentations graphiques, développés pour *Facebook*.
*http://techportal.ibuildings.com/2009/12/01/profiling-with-xhprof/*

**Wordpress 2.9**
La nouvelle version de Wordpress 2.9 vient de sortir après de nombreuses versions béta. Cette nouvelle version est toujours réalisée en PHP et MySQL 4.1.2 minimum, baptisée *Carmen*, en hommage à la chanteuse de jazz Carmen McRae. Elle apporte une série de correctifs, l'apparition d'une corbeille pour le contenu, un mini éditeur d'image pour réaliser des effets en ligne et aussi l'attribution d'une vignette par article. De nombreuses améliorations ont aussi vu le jour comme la gestion des médias, tinyMCE, optimisation des thèmes par défaut.
*http://www.wordpress-fr.net/blog/sortie-de-wordpress-2-9-carmen*

# Uwamp

Un nouveau AMP (Apache MySQL PHP) vient de voir le jour. Il rejoint la liste des environnements déjà existants pour Windows et s'appelle *uWamp*. Ce qui le distingue des autres :

- Il ne possède pas de kit d'installation, permettant ainsi de l'installer sur différents supports comme clé USB ou un autre support modulabe.
- Une interface allégée pour être un maximum intuitive. Elle propose un monitoring CPU.


Cette version propose 3 versions de PHP : 5.2.12, 5.3, 6.0 dev. Dans ce pack se trouve aussi *SQLite Browser*. Après l'installation, vous vous retrouvez dans votre propre environnement de développement.
*http://www.uwamp.com*


Progmatique est un portail sur la programmation et l'informatique vous proposant des cours et aides mémoires sur plusieurs langages de programmation : PHP, XHTML, CSS, mais aussi C/C++, JAVA. Des articles sur l'informatique en général (raccourcis claviers, registre windows, howto linux, ...) complètent le site. Webmaster et/ou développeur, des services gratuits vous sont aussi proposés, comme par exemple *CodesWall* (*http://www.codeswall.info*) qui vous permet de diffuser votre code source avec colorisation syntaxique sur l'irc, un forum, un site, etc...

Pour plus d'informations, n'hésitez pas à vous connecter sur *http://www.progmatique.fr*.

# Un classement FUN

Votre site web, blog ou site personnel peut contenir de nombreuses actualités. Ces actualités peuvent être diverses et variées car vous pouvez proposer des tutoriaux, des actualités, des articles développés. Bien sûr, l'outil que vous utilisez, propose d'afficher un hit parade des dernières lectures ou encore le contenu des dernières publications. Et pour accroitre encore plus de trafic, vous pouvez aussi proposer un classement des articles les plus commentés.

En combinant le langage PHP et le CSS, ce nouveau classement affichera un hit parade FUN.

Le concept de *Liam McCable* développe comment :

- Extraire les commentaires de la base de données.
- Définir les couleurs.
- Mettre en forme chaque ligne associée à un CSS.

*http://blog.creativityden.com/create-a-funky-most-commented-section-for-your-blog*

## Cours vidéo : PDO avancé

Le cours vidéo qui vous est présenté sur le CD-ROM, a été réalisé en exclusivité pour le magazine PHP Solutions par Christophe Villeneuve d'*Alter Way Consulting*.

Le cours va porter sur l'utilisation des marqueurs en PDO. Ce cours vidéo va se décomposer en différentes parties :

- Un rappel.
- Les marqueurs.
- Insertion.
- Débuggage.

### Rappel

Un rappel du chapitre précédent est important pour un lecteur qui n'a pas pu suivre l'utilisation de PDO. Cette petite partie vous permet de revoir comment créer une base de données, de visualiser son contenu avec un filtre SQL.

### Les marqueurs

Il existe différentes approches de marqueurs et de possibilités suivant les cas d'utilisations. Le cours vidéo vous montrera la façon la plus simple possible d'utiliser un seul marqueur, et aussi d'en utiliser plusieurs en même temps.

Ces opérations permettent de voir une utilisation plus poussée de PDO et surtout l'utilisation de la fonction `PDO::BindParam()`.

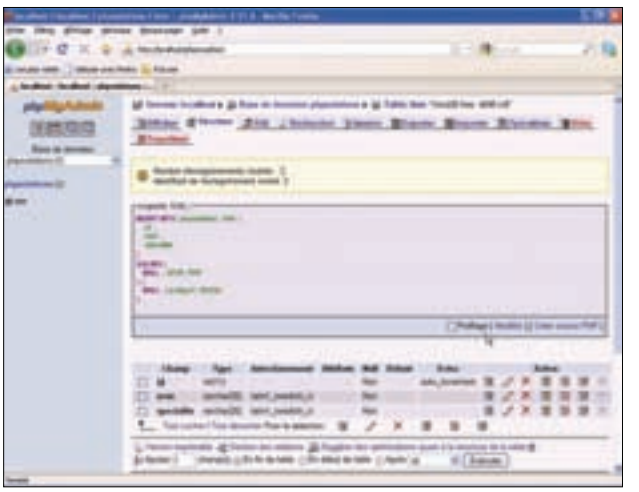
### Insertion

Les marqueurs ne se limitent pas à visualiser des données, ils offrent les même possibilités qu'une méthode classique, c'est-à-dire la possibilité d'insérer, de mettre à jour et de supprimer des données.

Le cours d'aujourd'hui vous montre la possibilité d'insérer des données dans la base en utilisant des marqueurs.

### Débuggage

Lorsque vous manipulez des marqueurs, vous pouvez avoir besoin de consulter ou de débugger les requêtes. En utilisant la fonction `PDO::debugDumpParams()`, vous pouvez obtenir de nombreuses informations pour corriger et faire évoluer vos requêtes.



## Cours vidéo : tour d'horizon de Google Maps en 30 minutes

Nous vous présentons le tutoriel vidéo sur Google Maps, préparé spécialement pour ce numéro par Aymeric Lagier.

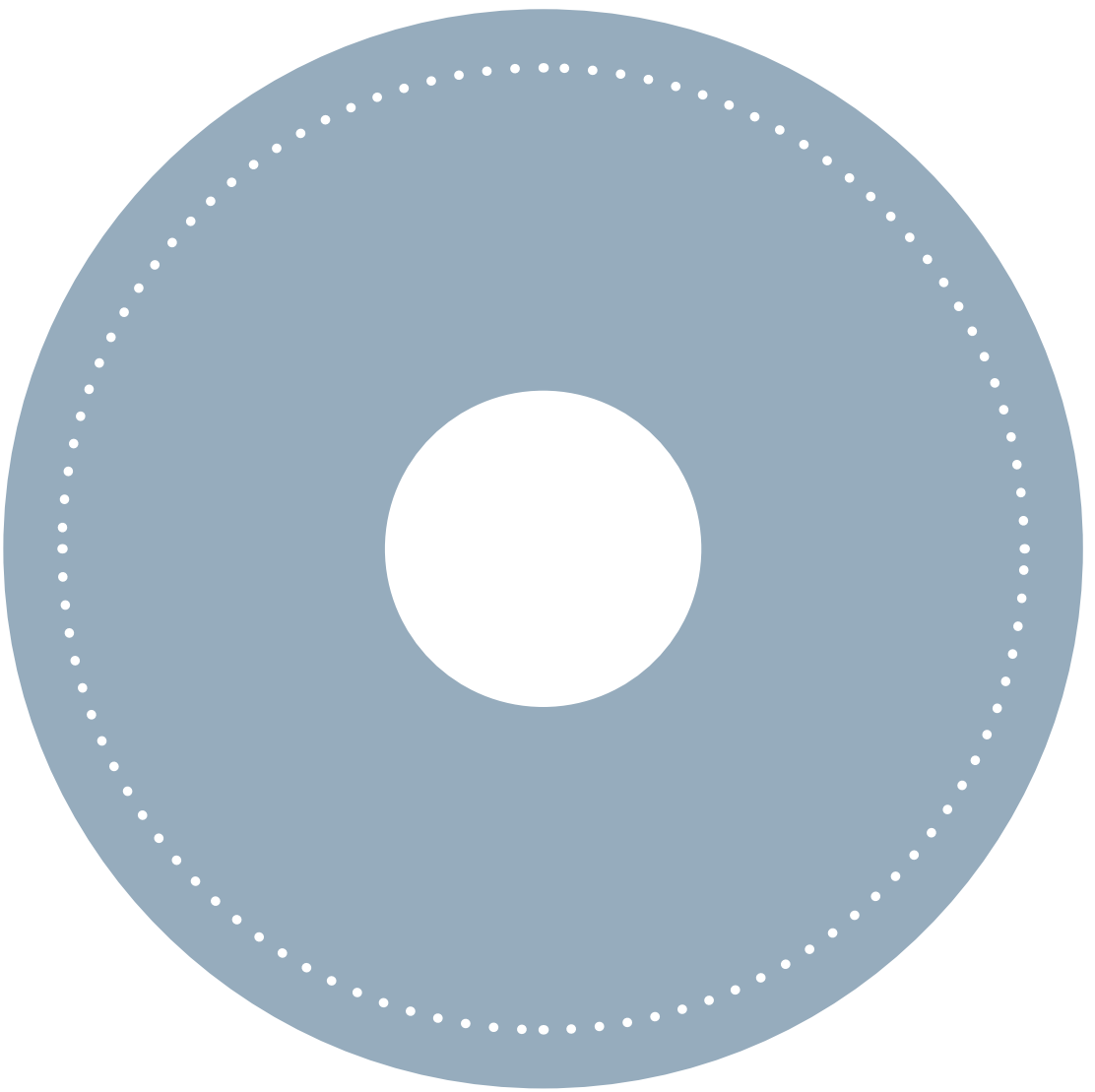
Google fournit une API pour son outil *Google Maps*. Il est donc possible de complètement intégrer *Google Maps* à votre site web et même de personnaliser vos cartes avec un panel d'outils impressionnants. Ce cours vidéo n'a pas la prétention de vous apprendre toute l'API, mais de donner un avant goût de ce qu'il est possible de faire. Les exemples présents dans cette vidéo sont tirés des exemples fournis par Google. Pour en découvrir plus et connaître de nouvelles fonctionnalités, je vous invite à consulter cette page : *http://code.google.com/apis/maps/documentation*.

Dans cette vidéo, vous apprendrez à construire une carte *Google Maps* et à placer des points sur une carte pour les fonctions les plus simples. Pour découvrir la puissance de *Google Maps*, nous dessinerons également des formes sur une carte et nous développerons une carte affichant la météo à différentes échelles de l'Europe.

Pour aborder ce cours, vous n'avez besoin que des bases en JavaScript et HTML.

Bon apprentissage !

**S'il vous est impossible de lire un CD, alors qu'il n'a pas de défaut apparent, essayez de le lire dans un autre lecteur.**



**Pour tout problème concernant les CDs, écrivez-nous à l'adresse : cd@phpsolmag.org**

# Oracle pour PHP : installation et introduction

La professionnalisation de PHP implique une rencontre inévitable et de plus en plus fréquente avec Oracle, le SGBD le plus utilisé en environnement critique ! Pourquoi utiliser Oracle ? Comment l'installer sur Linux ? Quelle extension choisir ? Comment utiliser Oracle avec PHP ? Cet article propose de couvrir ces sujets.

**Cet article explique :**

- Pour quelles raisons choisir Oracle avec PHP, quelle extension utiliser.
- Comment installer Oracle et ses extensions PHP sous Linux.
- Comment accéder à Oracle depuis un programme écrit en PHP.

**Ce qu'il faut savoir :**

- Des rudiments de PHP.
- Utiliser un système Linux.
- Des rudiments de compilation pour l'installation des extensions.

Niveau de difficulté


Le système de gestion de bases de données Oracle est un champion de la haute disponibilité. Les entreprises qui gèrent de gros systèmes le choisissent souvent en priorité pour les performances de son architecture répartie. La solution est robuste, soutenue par une multinationale en pleine expansion qui rassure les acheteurs. En revanche, le SGBDR Oracle n'est pas un logiciel libre. La société Oracle propose aujourd'hui la solution MySQL à ceux qui souhaitent les avantages offerts par une licence différente.

Il existe plusieurs connecteurs Oracle pour PHP. L'ancienne extension Oracle n'est plus utilisée et doit être bannie. Aujourd'hui, trois extensions peuvent être choisies en fonction de vos besoins :

- l'extension oci8 proposée par Zend et Oracle,
- l'extension oci8 libre,
- le pilote oracle pour PDO.

La solution *oci8* proposée conjointement par Zend Technologies et Oracle est fournie dans *Zend Server, Zend Platform* ou *Zend Core pour Oracle*. Elle possède l'avantage d'un support commercial conjoint Oracle et Zend pour les entreprises demandeuses de garanties, et certifie des performances optimales, même pour les fonctionnalités récentes. La version libre d'oci8 est de très bonne facture, il s'agit de la solution la plus utilisée aujourd'hui. Enfin, le pilote Oracle pour PDO fonctionne bien lui aussi, mais il ne faut pas espérer un accès natif aux fonctionnalités avancées. De plus, quelques problèmes d'instabilité mineurs sont notés par les utilisateurs, en particulier si vous souhaitez utilisez les types de données LOB.

## Installations

Cette première partie concerne les installations de Oracle et de PHP de manière à obtenir un environnement de développement ou de production utilisable. Il sera plus facile de mettre en oeuvre cet environnement sous Windows que sous UNIX ou Linux, car l'installation est très guidée et les extensions PHP sont fournies dans des installeurs tels que Wampserver. En revanche, la plupart des installations d'Oracle étant réalisées sous UNIX/Linux, cet article propose la description d'une installation sous Ubuntu Server, une distribution dérivée de Debian.

Si vous êtes sous Windows, vous pouvez suivre ces indications en installant un Ubuntu Server ou une Debian Lenny dans une machine virtuelle, en utilisant *VirtualBox* par exemple. La machine virtuelle de cette

### Machine virtuelle de l'article

Vous pouvez récupérer la machine virtuelle liée à cet article, et visualiser une démonstration de l'installation, à l'adresse suivante :

- *http://www.openstates.com/phpsolutions/*
- *Login et mot de passe : openstates / openstates*

### Travailler sur une machine virtuelle

Depuis qu'elle assure stabilité et performances, la virtualisation est de plus en plus courante et propose de nombreux avantages. On utilise plusieurs programmes tels que *Vmware, VirtualBox, KVM* ou *Xen* pour installer des systèmes virtuels sur un système hôte (machines physiques). Voici les raisons pour lesquelles vous pourriez choisir une machine virtuelle :

- Faciliter la sauvegarde et la restauration d'un système entier.
- Installer un système optimisé pour une utilisation donnée (Oracle avec PHP).
- Dupliquer facilement un environnement pour le partager avec d'autres personnes ou multiplier les noeuds d'un cluster.
- S'épargner l'installation d'Oracle et PHP de cet article si vous la trouvez fastidieuse.

La plupart des processeurs récents prennent en charge la virtualisation afin de rendre le système hébergé performant.

installation est gérée par KVM, mais elle est compatible avec d'autres gestionnaires, en particulier *VirtualBox*, que vous pouvez utiliser sous Windows et Linux.

L'installation de la machine virtuelle n'est pas l'objet de nos propos, nous n'en parlerons donc pas, mais vous trouverez tout ce qu'il faut comme information sur Internet, cette opération étant largement documentée. L'installation qui va suivre va se découper en plusieurs parties :

- l'installation d'Oracle,
- l'installation du client Oracle, Apache, PHP et leurs configurations,
- l'installation des extensions Oracle pour PHP.

Avant toute chose, nous devons faire quelques remarques sur le système d'exploitation qui va supporter l'installation. Oracle n'est pas spécialement léger, il consomme de l'espace disque et des ressources mémoire. C'est pourquoi vous devez veiller à proposer les valeurs minimales suivantes sous peine de se voir refuser l'installation du serveur Oracle :

- Espace disque minimal : 1,5 Go, au moins 3 Go conseillés.
- RAM minimale : 512 Mo, au moins 1 Go conseillé.
- Fichier d'échange minimal (swap) : 1 Go, au moins 2 Go conseillés.

En production ces valeurs doivent être beaucoup plus grandes, surtout si vous utilisez une version récente d'Oracle tel que la 11g.

## Installation d'Oracle

Dans le cadre de cet article, nous avons choisi d'utiliser Oracle XE, une version gratuite du SGBD Oracle, proposée pour les développements. D'un point de vue technique, il n'y aura pas de différence avec une licence commerciale. La plupart des installations d'Oracle se font sous *UNIX/Linux*. La machine virtuelle utilisée est un *Ubuntu Server*, mais il est également possible d'installer Oracle XE sous Windows grâce à un installeur adéquat.

Une fois les pré-requis précédents vérifiés, grâce à Joel Becker, nous pouvons installer Oracle XE avec des paquets Debian. Cela va nous faire gagner du temps et nous épargner un certain nombre de désagréments. Cette procédure est disponible à l'adresse suivante :

*http://www.oracle.com/technology/tech/linux/install/xe-on-kubuntu.html*

Il est important de rappeler que si vous ne disposez pas des pré-requis ci-avant, l'installation échouera. La première étape consiste à ajouter dans le fichier */etc/apt/sources.list* la


**Figure 1.** *Récupération de la clé permettant l'accès au dépôt APT*


**Figure 2.** *Installation des paquetages d'Oracle XE*

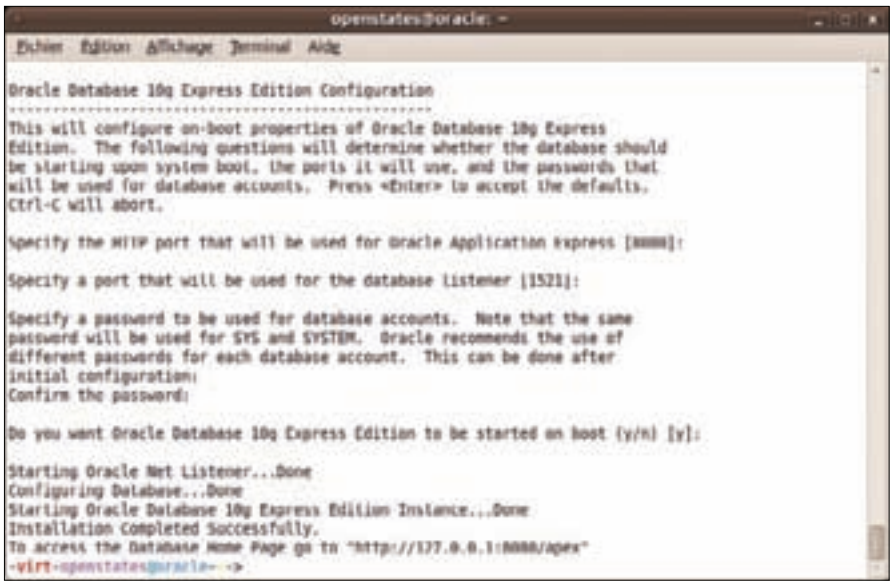



**Figure 3.** *Configuration minimale d'Oracle XE via le paquetage APT*

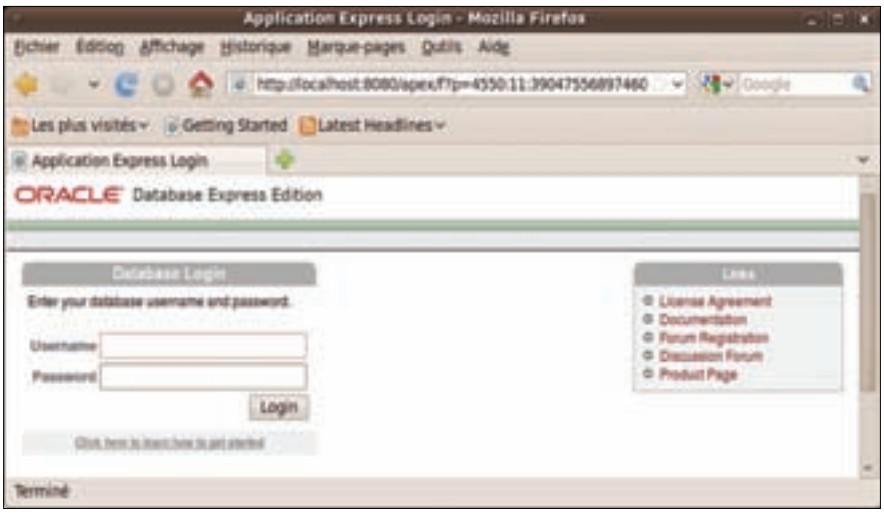



**Figure 4.** *Accès à l'interface APEX d'Oracle XE via le tunnel SSH*

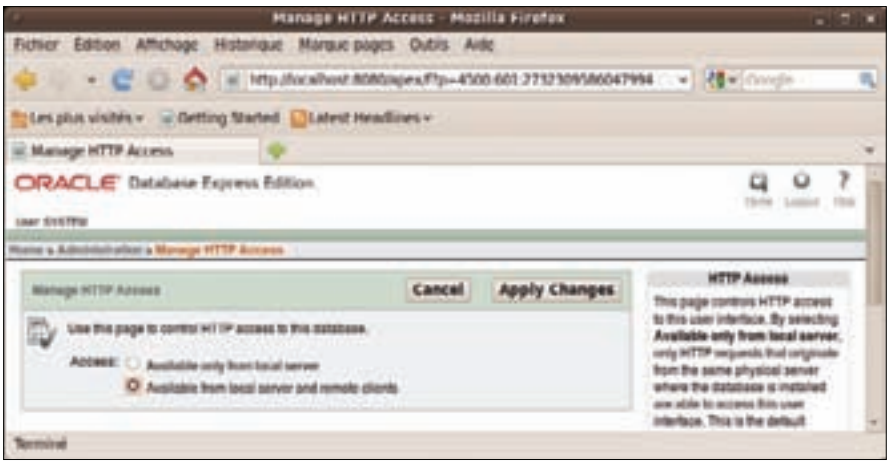



**Figure 5.** *Autorisation des accès depuis l'extérieurs, afin de se passer du tunnel*

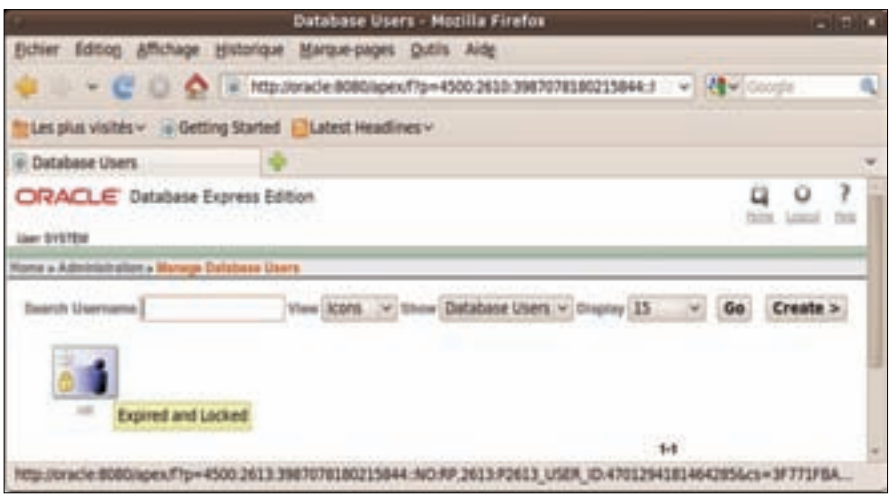



**Figure 6.** *Gestion des utilisateurs : l'utilisateur HR, expiré et verrouillé*

**Listing 1.** *Installation d'oracle avec les paquets Debian*

```
$ wget http://oss.oracle.com/el4/
RPM-GPG-KEY-oracle -O- | sudo apt-
key add -
$ sudo apt-get update
$ sudo apt-get install oracle-xe
$ sudo /etc/init.d/oracle-xe
configure
```

ligne permettant d'accéder aux paquets Oracle. Éditez ce fichier et ajoutez-y la ligne suivante :

```
deb http://oss.oracle.com/debian unstable
main non-free
```

Puis, dans un terminal entrez les commandes décrites dans le Listing 1 et répondez aux questions au fur et à mesure. Les trois premières commandes procèdent à l'inscription des paquets Oracle dans le dépôt APT (cf. Figure 1), puis au téléchargement et à l'installation de Oracle XE (cf. Figure 2). La dernière commande procède à la configuration minimale du SGBD et pose quelques questions tel que l'illustre la Figure 3 :

- Le port de l'interface d'administration : 8080 (valeur par défaut).
- Le port d'écoute du SGBD : 1521 (valeur par défaut).
- Le mot de passe administrateur : "openstates" dans notre exemple.
- Faut-il démarrer Oracle en même temps que le système : oui (défaut).

Suite à cette saisie, la configuration peut prendre un certain temps, puis l'interface d'administration est directement accessible. Par défaut, l'accès est autorisé uniquement en local, c'est pourquoi, compte tenu de notre configuration, nous devons trouver un moyen d'accéder à l'interface depuis l'extérieur. Une solution consiste à établir un tunnel ssh avec la machine virtuelle. Dans un terminal de la machine hôte, connectez-vous à la machine virtuelle avec la ligne de commande suivante (ou openstates est le nom de l'utilisateur du système et oracle un nom de domaine qui pointe vers la machine, que vous pouvez remplacer par une adresse IP) :

```
$ ssh -L8080:localhost:8080 openstates@
oracle
```

Ce tunnel peut être créé depuis une machine Windows grâce à un client SSH comme *putty*, à paramétrer dans l'onglet "tunnel". L'interface d'administration est alors accessible depuis l'extérieur de la machine virtuelle, sur le port 8080, tel que l'illustre la Figure 4.

Pour se passer du tunnel à l'avenir, connectez-vous avec l'utilisateur *system* et cochez *Available from local server and remote clients* (ou l'équivalent en français) dans la *page Home* > *Administration* > *Manage HTTP Access* illustrée sur la Figure 5.

Enfin, il faut que nous disposions d'un utilisateur pour nos connexions depuis PHP. Il en existe un nommé *HR* qui est expiré et désactivé par défaut. Pour l'activer, allez sur la page *Home* > *Administration* > *Manage Database Users* (cf. Figure 6). Dans le formulaire qui s'affiche (cf. Figure 7), spécifiez un mot de passe et mettez le statut du compte sur *Unlocked*. Les privilèges par défaut sont suffisants pour notre démonstration, il ne reste plus qu'a confirmer en cliquant sur *Alter User*.

Déconnectez-vous (bouton en haut à droite) puis re-connectez-vous avec l'utilisateur

*HR*. Dans la page qui s'affiche (cf. Figure 8), la fonctionnalité *Object Browser* propose un outil de gestion pour vos tables, vues, synonymes, séquences et autres objets disponibles dans votre base Oracle : une sorte de *phpMyAdmin* allégé. Enfin, le bouton *SQL* permet d'exécuter des requêtes à la main.

Il nous reste à vérifier que Oracle est bien lancé, ainsi que son listener qui permet l'accès au SGBD par d'autres programmes tels que Apache/PHP. Pour cela, exécutez la commande suivante :

```
$ ps xa | grep xe | grep -v grep
```

Le résultat devrait ressembler à celui de la Figure 9. La première ligne indique que le listener est bien actif et les autres que le SGBD Oracle est bien lancé et attend des connexions.

## Installation du client Oracle, Apache et PHP

Nous avons maintenant un SGBD Oracle qui tourne ! Mais ce n'est pas tout à fait suffisant, il nous faut à présent créer un point de connexion (que nous nommerons XE) et installer le client Oracle (instantclient) nécessaire aux extensions Oracle pour PHP, ainsi qu'une installation de Apache (serveur HTTP) et de PHP.

Commencez par éditer le fichier /etc/tnsnames.ora (cf. Figure 10. Si ce fichier n'existe pas, il faut le créer), puis recopiez-y la déclaration illustrée sur la Figure 11. Ensuite installons le client Oracle, PHP et Apache via `apt-get` :

```
$ sudo apt-get install oracle-xe-client
php5 php5-dev
```

Pour accéder à Oracle, il est nécessaire de définir des variables d'environnement, que ce soit dans le système ou dans la configuration d'Apache. Pour cela vous devez connaître le chemin vers le client Oracle, qui généralement est le suivant :

```
/usr/lib/oracle/xe/app/oracle/
product/10.2.0/client/
```

Pour éviter d'avoir des problèmes d'incompatibilité avec les jeux de caractères et assurer des performances optimales, il est utile de préciser le même jeu de caractères pour les échanges et le stockage interne des données par Oracle. Pour le connaître la valeur de ce jeu, utilisez sqlplus (Figure 12) ou la fonctionnalité *SQL* de l'interface *apex*.

Le Listing 3 spécifie les variables d'environnement du système (cf. Figure 13) et le Listing 4 les déclarations à mettre dans le fichier de configuration d'*Apache*, */etc/*


**Figure 7.** *Formulaire de mise à jour d'un utilisateur Oracle*

**Listing 2.** *Trouver le jeu de caractères de stockage par défaut*

```
$ /usr/lib/oracle/xe/app/oracle/product/10.2.0/client/bin/sqlplus
(...)
Enter user-name: system@XE
Enter password: openstates
SQL> select value from NLS_DATABASE_PARAMETERS where parameter='NLS_
CHARACTERSET';
```

**Listing 3.** *Variables d'environnement à mettre dans /etc/profile*

```
#----------------------------------------
#Variables pour Oracle
#----------------------------------------
export ORACLE_BASE=/usr/lib/oracle/xe/app/oracle
export ORACLE_HOME=$ORACLE_BASE/product/10.2.0/client
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
export PATH=$PATH:$ORACLE_HOME/bin
export NLS_LANG=WE8MSWIN1252
```

**Listing 4.** *Variables à mettre dans le fichier de configuration d'Apache*

```
SetEnv LD_LIBRARY_PATH /usr/lib/oracle/xe/app/oracle/product/10.2.0/client/lib
SetEnv TNS_ADMIN /etc/
SetEnv ORACLE_HOME /usr/lib/oracle/xe/app/oracle/product/10.2.0/client/
SetEnv NLS_LANG WE8MSWIN1252
SetEnv ORACLE_SID XE
```

**Listing 5.** *Récupération, compilation et installation de oci8*

```
sudo apt-get install gcc make autotools
mkdir ~/src
cd ~/src
wget "http://pecl.php.net/get/oci8-1.4.0.tgz"
tar -xzf oci8-1.4.0.tgz
cd oci8-1.4.0
phpize
./configure --with-php-config=/usr/bin/php-config --with-oci8=$ORACLE_HOME
make && sudo make install
sudo -s
echo "extension=oci8.so" >> /etc/php5/cli/php.ini
exit
php -m | grep oci8
```

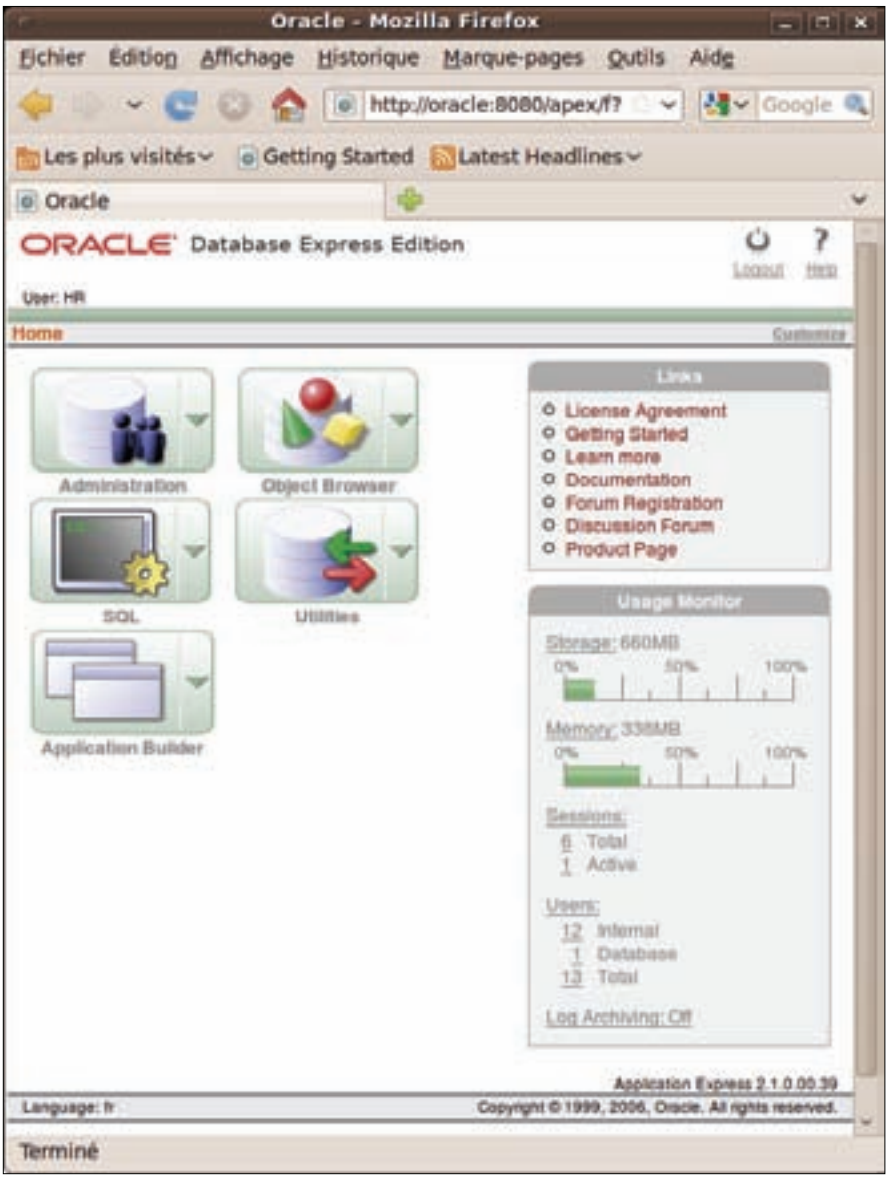



**Figure 8.** *Menu principal de l'interface APEX, utilisateur HR*

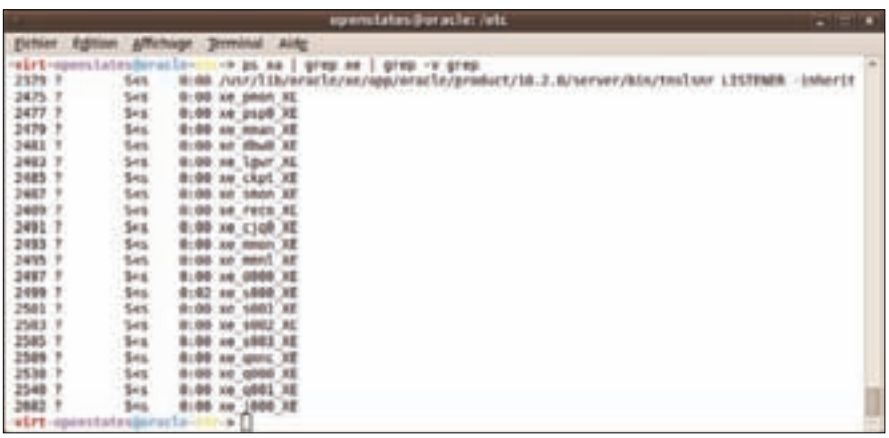


**Figure 9.** *Vérification de la mise en route d'Oracle XE*

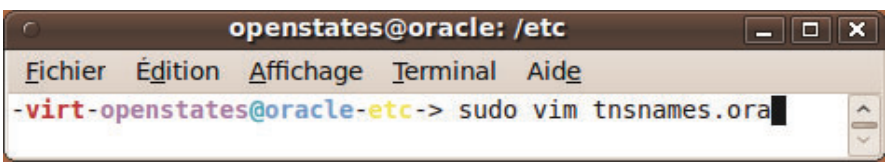



**Figure 10.** *Édition du fichier /etc/tnsnames.ora*

*apache2/apache2.conf* (cf. Figure 14). Voici à quoi correspondent ces variables :

- ORACLE_BASE : le répertoire d'installation d'oracle.
- ORACLE_HOME : le chemin vers l'environnement par défaut (client).
- LD_LIBRARY_PATH : les librairies utiles aux extensions PHP.
- NLS_LANG : le jeu de caractères à utiliser.
- PATH (shell uniquement) : permet l'accès facile aux binaires Apache tel que sqlplus.
- ORACLE_SID : le nom de la connexion à utiliser par défaut.

Une fois ces paramètres inscrits dans les fichiers de configuration, en particulier /etc/profile, rechargez les variables d'environnement du shell. Ceci sera utile pour les compilations à venir :

```
source /etc/profile
```

## Installation des extensions Oracle pour PHP

Nous avons déjà précisé en introduction qu'il existe actuellement deux types d'extensions possibles : *oci8* (en version libre ou livrée avec Zend) et *PDO*. Le driver PDO_OCI est limité aux fonctionnalités proposées par PDO et sera moins adaptée aux versions récentes d'Oracle. Cette extension est encore à ce jour spécifiée comme expérimentale sur la documentation de PHP. L'extension oci8 est en revanche plus stable. Elle permet d'utiliser des fonctionnalités avancées d'Oracle et gère mieux les paramètres de performances. Elle propose en particulier un cache client paramétrable, comme nous allons le découvrir par la suite. En revanche, si vous devez changer de SGBD à terme, la migration sera plus difficile si vous utilisez oci8.

### Compilation et installation de oci8

Il est probable qu'oci8 ne soit pas fourni dans les paquets Debian/Ubuntu. Il vous faudra alors l'installer via PEAR ou par la compilation. Savoir se débrouiller avec cette dernière option est une garantie si le paquetage adéquat n'est pas disponible dans PEAR et, de manière générale, si vous souhaitez une librairie dynamique adaptée à votre besoin. Il s'agit de la méthode que j'ai choisie dans le cadre de cet article. Le Listing 5 décrit les commandes à utiliser pour récupérer, compiler et installer oci8 (cf. Figure 15). Voici, en résumé, ce que fait chaque ligne de ce script :

- Installation des outils de compilation.
- Création d'un répertoire src dans le dossier utilisateur.
- On se déplace dans ce répertoire.
- Récupération des sources d'oci8.

- Décompactage des sources.
- Déplacement dans le dossier de compilation.
- Création des outils de compilation pour un module PHP.
- Création des makefiles.
- Compilation et installation de la librairie oci8.
- On se met en root.
- Activation de l'extension oci8 dans php.ini.
- Sortie du mode root.
- Vérification de la présence d'oci8 dans les modules.

Si la dernière ligne produit le résultat *oci8*, alors l'extension est installée avec succès. En fonction du système que vous utilisez, ces commandes peuvent évoluer, à vous de voir comment adapter votre démarche. Des notions de compilation sont parfois utiles dans ce genre de situation.

### Compilation et installation de pdo_oci

La compilation de pdo_oci se fera de la même manière que pour oci8. Il n'y a que deux changements relatifs à notre configuration :

- Le code source de l'extension pdo_oci est intégré au code source de PHP. Nous devons donc télécharger le code source de PHP pour compiler l'extension.
- D'autre part, la configuration PHP proposée par Debian fait échouer la compilation, mais un simple lien symbolique suffira à régler le problème, tel que le montre le Listing 6.

### Manipulations Oracle avec PHP

Maintenant que notre installation fonctionne, nous allons utiliser les fonctionnalités proposées par nos extensions pour accéder à la base Oracle de démonstration.

## Exemple simple avec oci8

Comme avec MySQL et d'autres SGBD, exécuter des requêtes Oracle se fait au travers d'une connexion déclarée au préalable. Le Listing 7 propose un script procédural qui lit des données dans la base Oracle et les affiche sur la sortie standard (cf. Figure 15). Analysons-le.

Dans un premier temps, on récupère le jeu de caractères à utiliser (depuis l'environnement, sinon en dur) puis on crée une connexion avec `oci_connect()`. Si la connexion échoue, `oci_error()` propose diverses informations dont un descriptif technique de l'erreur par le serveur Oracle. La compilation de la requête `SELECT` se fait avec `oci_parse()`, qui retourne une ressource.

La fonction `oci_set_prefetch()` est optionnelle, mais utile pour disposer de bonnes performances lorsqu'elle est utilisée judicieusement.Mettez une valeur qui correspond

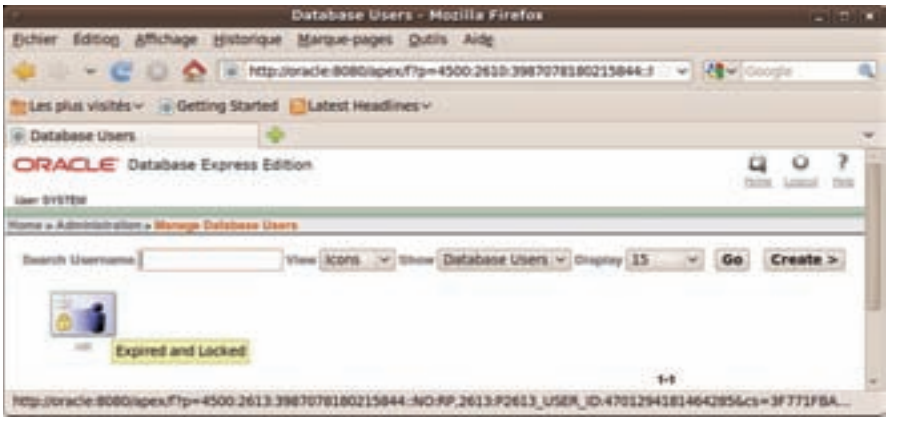



**Figure 11.** *Contenu de /etc/tnsnames.ora*

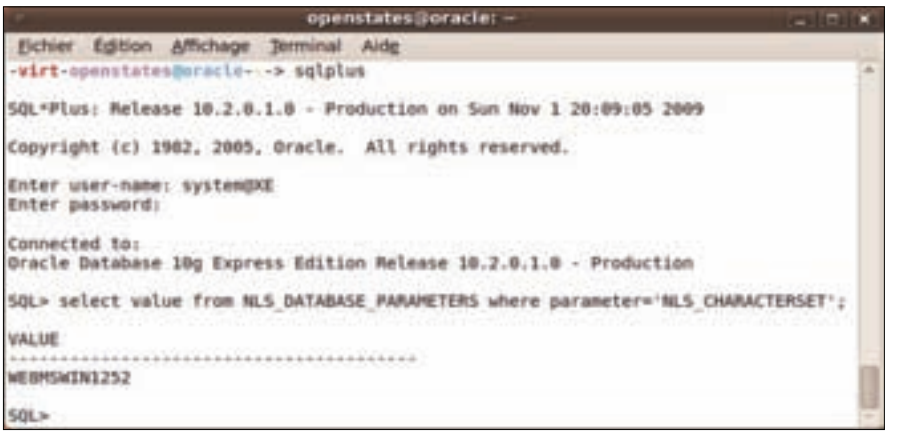



**Figure 12.** *Recherche du jeu de caractères par défaut avec sqlplus*

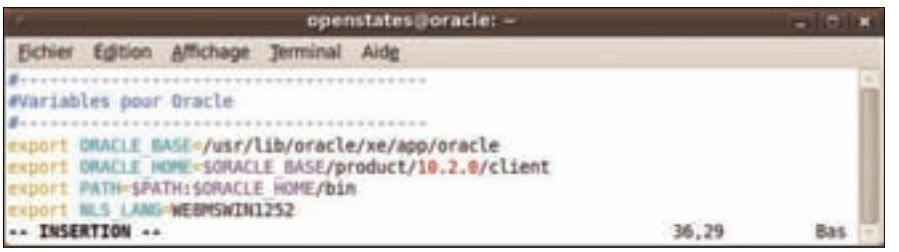



**Figure 13.** *Édition du fichier /etc/profile pour les variables d'environnement*


**Figure 14.** *Édition du fichier apache2.conf pour l'environnement Apache*

**Listing 6.** *Récupération, compilation et installation de pdo_oci*

```
cd ~/src
wget http://museum.php.net/php5/php-5.2.6.tar.gz
tar -xzf php-5.2.6.tar.gz
cd php-5.2.6/ext/pdo_oci/
phpize
cd /usr/include
sudo ln -s php5 php
cd -
./configure --with-pdo-oci=$ORACLE_HOME --with-php-config=/usr/bin/php-config
make && sudo make install
sudo -s
echo "extension=pdo_oci.so" >> /etc/php5/cli/php.ini
exit
php -m | grep -i pdo_oci
```

**Listing 7.** *Accès simple à Oracle avec l'extension oci8*

```
<?php
// Récupération du charset depuis l'environnement
$charset = getenv('NLS_LANG') ? getenv('NLS_LANG') : 'WE8MSWIN1252';

// Connexion à Oracle, en utilisant le bon jeu de caractères
$connexion = oci_connect('HR', 'openstates', 'XE', $charset);
if (!$connexion) {
    print_r(oci_error());
    exit;
}

// Compilation de la requête à exécuter
$query = 'SELECT COUNTRY_ID, COUNTRY_NAME FROM COUNTRIES';
$parsedQuery = oci_parse($connexion, $query);
if (!$parsedQuery) {
    print_r(oci_error($connexion));
    exit;
}

// Pré-chargement de 100 enregistrements avant itération
// pour augmenter les performances (connecteurs récents)
oci_set_prefetch($parsedQuery, 100);

// Exécution de la requête
$result = oci_execute($parsedQuery, OCI_COMMIT_ON_SUCCESS);
if (!$result) {
    print_r(oci_error($parsedQuery));
    exit;
}

// Récupération des résultats
while ($row = oci_fetch_array($parsedQuery, OCI_NUM | OCI_RETURN_NULLS))
{
    vprintf("%2s %s\n", $row);
}

// Fermeture de la connexion
oci_close($connexion);
```

**Listing 8.** *Accès simple à Oracle avec l'extension pdo_oci*

```
<?php
// Récupération du charset depuis l'environnement
$charset = getenv('NLS_LANG') ? getenv('NLS_LANG') : 'WE8MSWIN1252';

// Connexion à Oracle
$dsn = 'oci:dbname=//localhost:1521/XE;charset=' . $charset;
$pdo = new PDO($dsn, 'HR', 'openstates');

// Compilation et exécution de la requête
$query = 'SELECT COUNTRY_ID, COUNTRY_NAME FROM COUNTRIES';
$result = $pdo->query($query, PDO::FETCH_ASSOC);

// Récupération des résultats
foreach ($result as $row) {
    vprintf("%2s %s\n", $row);
}
```


**Figure 15.** *Exécution des commandes de compilation (3 à 7)*

au nombre de lignes que vous souhaitez lire. Ces lignes seront pré-chargées dans le cache client : c'est à dire celui de l'extension.

## Cache client, cache serveur

Pour optimiser les performances d'accès aux données via les requêtes SQL, Oracle propose deux types de cache : le cache client, lié à oci8, apache et PHP, et le cache serveur dans Oracle. Le cache pré-charge les résultats de vos requêtes. Il est persistant entre deux appels, mais supprimé si vous mettez à jour les données entre-temps.

En paramétrant astucieusement ces deux caches dans les configurations PHP et Oracle, vous pouvez améliorer sensiblement vos performances. Ceci est particulièrement recommandé lorsque votre schéma de base de données est très normalisé, c'est à dire divisé en beaucoup de petites tables qui nécessitent de nombreux accès.

Ensuite, `oci_execute()` donne l'ordre d'exécuter la requête. La ressource renvoyée est utilisable par `oci_fetch_array()` qui, dans la boucle `while`, récupère les enregistrements un par un. Vous trouverez dans la documentation PHP la liste des constantes qui définissent la manière dont doit se comporter cette fonction. Enfin, sur la dernière ligne, `oci_close()` ferme la connexion en cours.

## Connexions persistantes

Il est possible, avec oci8, d'utiliser des connexions persistantes avec `oci_pconnect()`. Ce mécanisme maintient les connexions ouvertes entre plusieurs requêtes HTTP. Cela permet d'économiser les ressources nécessaires à l'établissement d'une connexion et donc accélère l'exécution des requêtes. En revanche, de nombreuses connexions ouvertes ne sont potentiellement pas utilisées, ce qui consomme aussi des ressources.

## Bonnes pratiques avec oci8

L'extension oci8 mélange du code procédural et objet. Elle propose des fonctions très spécifiques. Si vous développez en POO, il est recommandé d'encapsuler les fonctions oci8 dans un ou plusieurs objets ou d'utiliser une implémentation existante, telle que celle proposée par `Zend_Db`. Cela permettra d'abstraire la structure de l'extension et d'automatiser certaines tâches, telles que les déconnexions, le maintien d'une seule connexion, etc. Ainsi, vous serez mieux armé pour développer efficacement et durablement.

## Exemple simple avec PDO

Accéder à Oracle via PDO se fait de la même manière qu'avec d'autres SGBD, puisque les classes PDO sont communes quelque soit votre choix. Pour rappel, PDO est un projet développé par Wez Furlong, ayant pour but de

proposer une interface d'accès objet homogène, utilisable avec les SGBD les plus utilisés du marché. Si vous utilisiez déjà PDO, vous ne serez pas dépaysé. Le Listing 8 propose un programme PHP qui fait la même chose que le précédent avec oci8 (Listing 7).

Un des avantages de PDO est sa structure cohérente, 100% orientée objet. Ainsi, certains mécanismes sont effectués de manière implicite : la compilation (prepare), la génération des exceptions ou encore, la fermeture de la base de données. Comme nous pouvons le voir, cela réduit le nombre de lignes de code, bien que dans cet exemple la gestion des erreurs n'est pas assurée.

Les 3 premières lignes établissent la connexion et créent l'objet `$pdo`. Celui-ci possède une méthode `query()` qui compile et exécute la requête, retourne un objet `$result` de type `PDOStatement`, itérable directement dans un `foreach`, comme le serait un tableau PHP.

## Exemple plus avancé avec oci8

Le programme décrit dans le Listing 9 est utilisable en ligne de commandes. Il sert à insérer un ou plusieurs pays dans la table COUNTRIES de la base exemple. Grâce à la gestion des transactions proposées par Oracle, nous pouvons assurer l'atomicité de notre opération. En d'autres termes, si l'une des requêtes échoue, tout est annulé, sinon tout est validé (voir les propriétés ACID dans la remarque ci-jointe). Notre script s'utilise tel que mentionné sur la Figure 17.

## Les propriétés A.C.I.D.

ACID est un acronyme qui fait référence à quatre propriétés, qui peuvent être toutes assurées par Oracle et les SGBD courants du marché : MySQL, PostgreSQL, DB2, MSSQL, etc.

- Atomicité : une transaction, même de plusieurs requêtes, doit être complètement validée ou complètement annulée.
- Cohérence : une transaction ne doit jamais rendre des données incohérentes (rôle des règles d'intégrités).
- Isolation : les exécutions de transactions sont indépendantes, elles n'interfèrent pas entre elles.
- Durabilité : une fois le signal de validation émis, le résultat est écrit sur le disque, il ne peut disparaître par erreur.

Comme dans le premier exemple (Listing 7), notre script se connecte à Oracle et compile une requête avec `oci_parse()`. En revanche, la requête est paramétrée (requête préparée) avec des paramètres nommés : `code`, `:country` et `:idregion`.

## Requêtes préparées

Le principe d'une requête préparée est de séparer la requête SQL des données qu'elle

**Listing 9.** *Exemple plus avancé avec oci8*

```
<?php
// Message d'information pour l'utilisation en ligne de commandes
function usage()
{
    echo "Usage: sudo -u oracle " . basename( _ _ FILE _ _ );
    echo " country _ abbr,country _ name,region _ id [...]\n";
    exit;
}

// Traitement des arguments
if ($ _ SERVER['argc'] < 2) {
    usage();
}
$values = $ _ SERVER['argv'];
array _ shift($values);

// Connexion à oracle
$charset = getenv('NLS _ LANG') ? getenv('NLS _ LANG') : 'WE8MSWIN1252';
$connexion = oci _ connect('HR', 'openstates', 'XE', $charset);

// Compilation de la requête à exécuter
$query = 'INSERT INTO COUNTRIES VALUES (:code, :country, :idregion)';
$stmt = oci _ parse($connexion, $query);

// Affectation des paramètres à des variables PHP
oci _ bind _ by _ name($stmt, ':code',     $code,     2,  SQLT _ CHR);
oci _ bind _ by _ name($stmt, ':country',  $country,  64, SQLT _ CHR);
oci _ bind _ by _ name($stmt, ':idregion', $idregion, -1, SQLT _ LNG);

// Exécution des requêtes
foreach ($values as $row) {
    $tabRow = explode(',', $row);
    if (count($tabRow) != 3) {
        throw new Exception("Bad row syntax [" . $row . "]");
    }
    list($code, $country, $idregion) = $tabRow;
    oci _ execute($stmt, OCI _ DEFAULT);
}

// Validation de la transaction
$committed = oci _ commit($connexion);

// Vérification de la validation : si une erreur est survenue, afficher
// le message d'erreur
if (!$committed) {
    $error = oci _ error($conn);
    echo 'Validation échouée : ' . $error['message'] . "\n";
} else {
    echo "Données insérées\n";
}

// Libération des resources
oci _ free _ statement($stmt);

// Fermeture de la connexion
oci _ close($connexion);
```

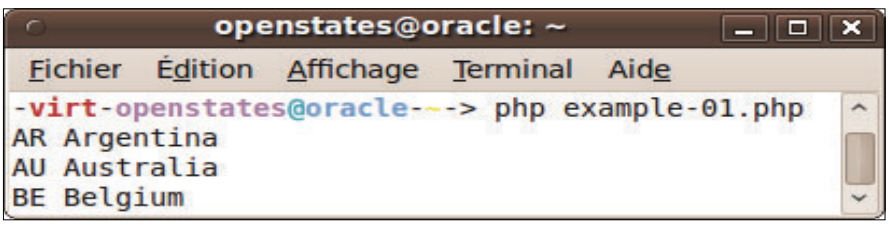



**Figure 16.** *Exécution de l'exemple 1*

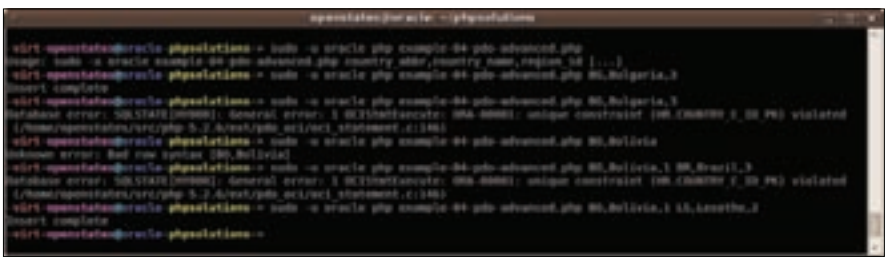


**Figure 17.** *Exécution de l'exemple 4, vérification des règles d'intégrité*

**Listing 10.** *Exemple plus avancé avec pdo_oci*

```
<?php
// Message d'information pour l'utilisation en ligne de commandes
function usage()
{
    echo "Usage: sudo -u oracle " . basename( _ _ FILE _ _ );
    echo " country _ abbr,country _ name,region _ id [...]\n";
    exit;
}

// Traitement des arguments
if ($ _ SERVER['argc'] < 2) {
    usage();
}
$values = $ _ SERVER['argv'];
array _ shift($values);

// Connexion à Oracle
$charset = getenv('NLS _ LANG') ? getenv('NLS _ LANG') : 'WE8MSWIN1252';
$dsn     = 'oci:dbname=//localhost:1521/XE;charset=' . $charset;
$pdo     = new PDO($dsn, 'HR', 'openstates');
$pdo->setAttribute(PDO::ATTR _ ERRMODE, PDO::ERRMODE _ EXCEPTION);

// Début de transaction
$pdo->beginTransaction();
try {

    // Exécution des requêtes
    $query  = 'INSERT INTO COUNTRIES VALUES (?, ?, ?)';
    $stmt   = $pdo->prepare($query);
    foreach ($values as $row) {
        $tabRow = explode(',', $row);
        if (count($tabRow) != 3) {
            throw new Exception("Bad row syntax [" . $row . "]");
        }
        $stmt->execute($tabRow);
    }

    // Tous s'est bien passé : validation
    $pdo->commit();
    echo "Insert complete\n";
}

// Problème dans PDO : message d'erreur + annulation
catch (PDOException $e) {
    echo "Database error: " . $e->getMessage() . "\n";
    $pdo->rollback();
}

// Problème détecté (hors pdo) : erreur + annulation
catch (Exception $e) {
    echo "Unknown error: " . $e->getMessage() . "\n";
    $pdo->rollback();
}
```

doit traiter. Par exemple, au lieu de mélanger la requête `INSERT` avec les données à insérer, on va d'abord créer une requête `INSERT` paramétrée, la compiler puis l'exécuter avec les données, en 2 temps. Les requêtes préparées sont bénéfiques aussi bien en termes de performances que pour se protéger des attaques par injection SQL. Elles sont en particulier très efficace avec un cache de requête, car elle réduisent considérablement le nombre de requêtes différentes à compiler.

Ces paramètres sont ensuite liés à des variables grâce aux 3 appels `oci_bind_by_name()`. Pour chacun d'eux, on mentionne :

- le lien vers la requête SQL en cours (statement),
- le nom du paramètre,
- la variable associée,
- la taille maximale des données
- le type de données.

Le type de données par défaut est `SQLT_CHR`. La boucle `foreach` exécute autant d'insertions qu'il y a de données mentionnées en argument de notre commande. Les données sont découpées avec `explode()`, puis les variables assignées avec `list()` et enfin, `oci_execute()` lance l'opération d'insertion.

## OCI_DEFAULT

Méfiez-vous de cette constante que l'on utilise en deuxième argument de `oci_execute()`. Elle signifie que la transaction doit être validée avec `oci_commit()` dans le cas d'une insertion ou d'une mise à jour. Cela n'est pas très explicite au premier abord. Si cette option n'est pas précisée, la validation (*commit*) sera automatique. Une fois toutes ces opérations effectuées, `oci_commit()` valide l'ensemble des insertions. Et pour finir, `oci_free_statement()` libère les ressources utilisées par la requête en cours, puis `oci_close()` ferme la connexion.

## Exemple plus avancé avec pdo_oci

Intéressons-nous maintenant à la version PDO de la même fonctionnalité (cf. Figure 17). Le Listing 10 propose une implémentation. Une fois la connexion effectuée, la méthode `setAttribute()` impose à PDO de générer des exceptions en cas de problème à l'exécution. Cela facilitera la gestion des erreurs.

La première chose à faire lorsqu'un ensemble de requêtes doivent être vues comme une opération atomique est l'appel `$pdo->beginTransaction()` qui indique à PDO, comme son nom l'indique, qu'on commence une transaction. Dans un bloc `try{}`, nous rassemblons l'ensemble des exécutions jusqu'à la validation finale effectuée par `$pdo->commit()`. Entre-temps, la requête paramétrée est compilée avec `$pdo->prepare()` et exécutée avec `$stmt->execute()`, `$stmt` étant renvoyé par `prepare()`. Ici, nous utilisons des paramètres non nommés. C'est `execute()` qui prend un tableau contenant les valeurs à traiter.

Enfin, l'annulation générale (*rollback*) de la transaction est assurée dans un ou plusieurs bloc(s) catch par l'appel `$pdo->rollback()`. Ici nous en avons deux, qui séparent les problèmes générés par PDO et les autres, mais le dernier catch suffirait. Rappelons qu'avec PDO, il n'y a pas de méthode `close()`, la connexion est automatiquement fermée dans le destructeur de l'objet PDO, donc à la fin de l'exécution du script PHP.

## Conclusion

Cet article est une introduction sur l'utilisation d'Oracle avec PHP. Sur le même sujet, d'autres fonctionnalités sont intéressantes à aborder : la gestion des lobs, les curseurs, les descripteurs, les procédures stockées ou encore les multiples possibilités d'optimisation qu'Oracle propose. À suivre dans un article ultérieur !


## GUILLAUME PONÇON

*Guillaume Ponçon travaille depuis plus de 8 ans avec PHP dans le monde professionnel. Il est aujourd'hui consultant, formateur et entrepreneur. Son parcours s'est enrichi d'une centaine de missions stratégiques auprès d'entreprises et de grands comptes. Il est également spécialiste des outils et méthodes d'industrialisation open source et Zend (Zend Framework, Zend Server...).*

# Introduction au DOM avec PHP

Construire, représenter et manipuler un document XML au sein d'une architecture orientée objet est l'une des tâches les plus courantes des applications modernes. Ainsi dans son souci constant de normalisation le W3C recommande depuis 1998 une méthode unifiée pour manier un document XML : le Document Object Model.

**Cet article explique :**

- Les notions de base du Document Object Model.
- Comment utiliser l'extension DOM de PHP.

**Ce qu'il faut savoir :**

- Connaître la programmation orientée objet.
- De solides notions en XML.

Niveau de difficulté


Contrairement à ce que l'on croit bien souvent le DOM (*Document Object Model*) ne définit pas le document XML ou HTML lui-même. Il s'agit en réalité d'une API (*Application Programming Interface*) qui, comme son nom l'indique propose une interface indépendante de toute plate-forme et de tout langage permettant de représenter et de manipuler un document XML sous la forme d'objets au sein d'un script ou d'un programme orienté objet. Le *Modèle Objet de Document* est donc la représentation de documents XML et par extension HTML, sous la forme d'un arbre aisément transposable en une série d'objets respectant cette API.

Depuis PHP 4 il existe une extension d'abord disponible dans PEAR et désormais distribuée dans le module PHP standard qui propose une implémentation du DOM au sein de PHP. Nous présenterons dans cet article les procédés de base à connaître pour construire, représenter et manipuler un document XML en utilisant le DOM de PHP. Nous aborderons également le fonctionnement de la syntaxe *XPath* qui nous permettra de naviguer plus aisément au sein d'un document XML.

## Principes de base

Le DOM représente donc notre arbre XML sous la forme d'un ensemble d'objets unifiés. En PHP comme dans tous les autres langages implémentant cette API on retrouvera donc une classe `DOMNode`. Chaque point dans notre arbre sera une instance de cette classe. Chacun de ces noeuds pouvant contenir des noeuds enfants ou être qualifié par divers attributs.

## Le document

Même la racine d'un document, représentée alors par la classe `DOMDocument` n'échappe pas à cette règle. La classe `DOMDocument` apportera cependant quelques méthodes supplémentaires à la classe `DOMNode`. Comme par exemple l'ouverture de documents déjà existants ou encore l'ajout d'éléments dans notre arbre.

Une fois votre document chargé en mémoire et représenté par une instance de la classe `DOMDocument`, vous pourrez alors manipuler aisément celui-ci. Si nous savons désormais que chacun des noeuds de notre arbre est représenté par une instance de *DOMNode* et que la racine de celui-ci l'est par une instance de la classe `DOMDocument` il nous reste à voir comment parcourir et manipuler les

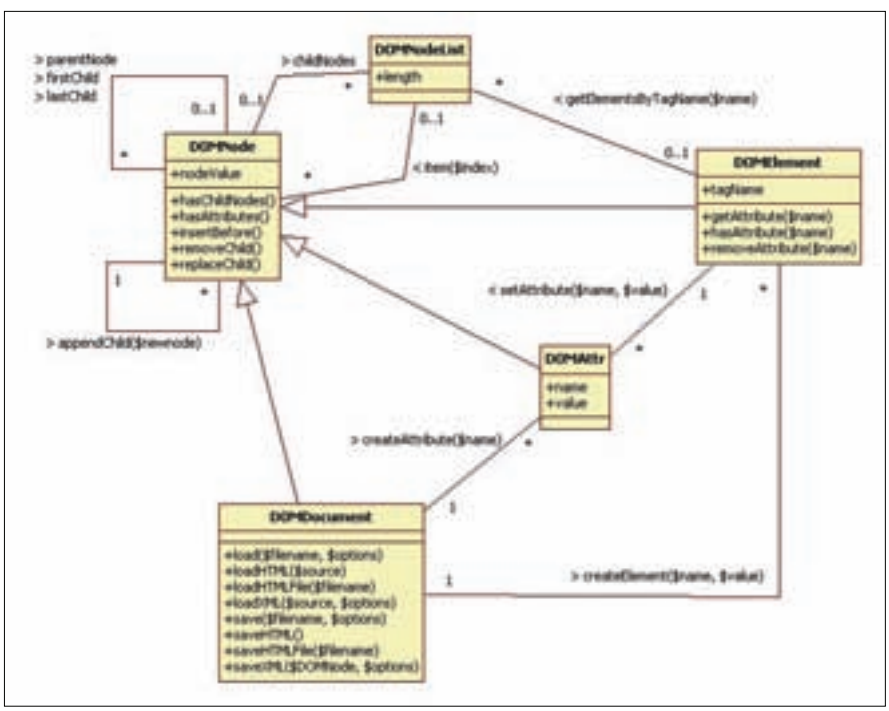



**Figure 1.** *DOM – Diagramme de classe simplifié*

balises et les attributs qui composent votre document.

## Balises et attributs

Le reste des balises de votre document ; qu'elles soient HTML ou plus largement XML seront représentées sous la forme d'un objet dont la classe hérite à son tour de `DOMNode`. Il s'agira alors d'un objet de type *DOMElement*. Les attributs quant à eux seront des instances de *DOMAttr*.

## Premier exemple

Partant de cela nous pouvons alors construire ou parcourir facilement tout document HTML ou XML avec le DOM en PHP. Créons par exemple une première page de type *hello world* dans un fichier *index.php* à la racine de notre site accessible à l'adresse *http://monsite.com/index.php*.

## Lire et écrire des documents

Une des forces de l'extension DOM de PHP est de pouvoir faire ce que l'on veut des documents que l'on manipule. Avec les méthodes `DOMDocument::save()`, `DOMDocument::saveHTMLFile()` et `DOMDocument::saveXML()` on peut soit en afficher le résultat directement vers le flux standard de sortie, soit enregistrer le résultat directement dans un fichier.

De la même manière avec les méthodes `DOMDocument::__construct()`, `DOMDocument::load()`, `DOMDocument::loadHTMLFile()` et `DOMDocument::loadXML()` vous pouvez soit créer vous même vos propres documents en partant de zéro, soit ouvrir un fichier contenant le document sur lequel vous allez travailler. Imaginons par exemple que nous avons notre document html précédemment stocké dans un fichier *hello-world.html* à côté de notre index.php accessible à l'adresse *http://moniste.com/hello-world.html*.

Nous n'aurions alors pas besoin de reconstruire intégralement ce dernier pour le manipuler à nouveau. Dans un fichier *index2.php* accessible à l'adresse *http://monsite.com/index2,php* nous pourrions par exemple récupérer le document précédent et ajouter un paragraphe (voir Listing 2).

**Listing 1.** *Hello World !*

```
// création du document
$document = new DOMDocument();

// création d'un nouvel élément html
$html = $document->createElement('html');

// ajout de l'élément html a la suite du document
$document->appendChild($html);

// création d'un nouvel élément head
$head = $document->createElement('head');

// création d'un nouvel élément title
$title = $document->createElement('title', 'bonjour');

// ajout de l'élément title à la suite, dans le noeud head
$head->appendChild($title);

// ajout de l'élément head à la suite, dans le noeud html
$html->appendChild($head);

// création d'un nouvel élément body
$body = $document->createElement('body');

// création d'un nouvel élément p avec pour valeur Hello World !
$p = $document->createElement('p', 'Hello World !');

// ajout de l'élément p à la suite, dans le noeud body
$body->appendChild($p);

// ajout de l'élément body à la suite, dans le noeud html
$html->appendChild($body);

// on affiche la source de notre document
echo $document->saveHTML();

// on peut même le sauvegarder pour plus tard
$document->saveHTMLFile('hello-world.html');
```

**Listing 2.** *Lecture et écriture dans un fichier*

```
// ouverture du fichier hello-world.html
$document->loadHTMLFile('hello-world.html');

// création d'un nouvel élément p avec pour valeur Bonjour le monde !
$p = $document->createElement('p', 'Bonjour le monde !');

// récupération du premier élément body existant
$body = $document->getElementsByTagName('body')->item(0);

// ajout de l'élément p à la suite, dans le noeud body
$body->appendChild($p);

// on enregistre à nouveau notre document html
$document->saveHTMLFile('hello-world-2.html');
```

## Passage par référence

Les noeuds sont manipulés par référence et non par valeur par l'extension DOM de PHP. Ce qui signifie par exemple que si vous ajoutez un noeud *X* à votre document puis que vous modifiez *X* par la suite en lui ajoutant un attribut par exemple ; les modifications apportées à ce noeud seront reportées au sein de votre document. Bien évidemment comme il s'agit de références, si vous détruisez votre variable *X* le noeud ne sera pas supprimé du document.

Pour mieux comprendre le mécanisme des références, reportez-vous à la documentation de PHP : *http://fr.php.net/manual/fr/language.references.php*.

## Parcourir un document

En plus de permettre la lecture et l'écriture de documents HTML et XML sous la forme d'objets, le *Document Objet Model* est également utile lorsque l'on souhaite parcourir l'arbre de ces derniers. Le DOM permet de connaître aisément tout l'environnement de chaque élément d'un arbre XML. La méthode `DOMElement::getElementsByTagName()` vous permettra par exemple de connaître tous les descendants d'un élément ayant un type donné. De la même façon vous pouvez consulter l'attribut d'un élément avec la méthode `DOMElement::getAttribute()`.

Dans notre Listing 2 nous avons vu par exemple comment récupérer le premier élément `<body>` d'un document. Il est cependant possible de résoudre des problématiques plus complexes. Imaginons par exemple que pour un flux RSS donné nous souhaitions compter le nombre d'éléments `<guid>` avec l'attribut *isPermalink* ayant la valeur `false`. La portion de code résolvant ce problème prendrait alors l'allure de le Listing 3.

## XPath

Si parcourir des documents simples est un jeu d'enfant avec les méthodes proposées par

**Listing 3.** *DOM – Recherche dans le document*

```
// récupération d'un flux rss

$xml_source = file_get_contents('http://www.titaxium.com/feed/rss2');
$document = new DOMDocument();
$document->loadXML($xml_source);

// on compte le nombre d'éléments guid avec l'attribut isPermaLink=false

$permalinks = array();
$guids = $document->getElementsByTagName('guid');
for ($i=0; $i<$guids->length; ++$i) {
    $item = $guids->item($i);
    if ($item->getAttribute('isPermaLink')=='false') {
        $permalinks[] = $item->getAttributeNode('isPermaLink');
    }
}

// on affiche le résultat

echo count($permalinks);
```

**Listing 4.** *XPath – Recherche dans le document*

```
// récupération d'un flux rss

$xml_source = file_get_contents('http://www.titaxium.com/feed/rss2');
$document = new DOMDocument();
$document->loadXML($xml_source);

// on compte le nombre d'éléments guid avec l'attribut isPermaLink=false

$xpath = new DOMXPath($document);
$entries = $xpath->query("//guid[@isPermaLink='false']");

// on affiche le résultat

echo $entries->length;
```

le DOM cela peut vite s'avérer être plus ardu dans le cas de documents complexes. *XPath* peut être alors une alternative à une série de boucles et de conditions imbriquées pour parcourir un arbre.

*XPath* est un petit langage d'interrogation qui s'avère vite très utile. Il s'agit d'une syntaxe non XML créée pour désigner une portion d'un document XML ou par extension HTML. Son approche peut paraître quelque peu semblable à l'adressage utilisé par les systèmes de fichiers. On peut sélectionner des éléments soit en spécifiant leur adresse absolue soit en définissant une série de critères.

## Chemin absolu

Si l'on connait exactement l'emplacement de l'élément que l'on souhaite récupérer, la première des solutions consiste à spécifier son chemin absolu. C'est le type de requête *XPath* le plus simple. Par exemple, pour sélectionner l'élément `<title>` d'une page HTML la requête prendra la forme suivante : */html/head/title*.

## Recherche par critères

En revanche si l'on souhaite récupérer un élément ou une série d'éléments sans connaître par avance leurs emplacements dans l'arbre où simplement trouver tous les éléments ayant un ensemble de caractéristiques : c'est la recherche par critère qu'il faudra utiliser.

Au sein d'une requête *XPath*, le simple slash défini un lien hiérarchique alors qu'un double slash définit un critère. La requête */html/body/p* définie tous les paragraphes d'une page HTML directement enfants de `<body>` lui même directement enfant de `<html>`, à l'inverse la requête *//p* définira tous les paragraphes de votre document.

Une expression entre crochets permet également de spécifier plus précisément un élément. Les attributs peuvent être eux aussi considérés comme des critères de sélection. Par exemple la requête *//a[@rel='nofollow']* sélectionnera tous les liens avec un attribut `rel=nofollow`. Le nombre d'expressions entre crochets n'est pas limité. Ainsi vous pouvez très bien écrire la requête *//a[@rel='nofollow'][@class]* qui correspond à l'ensemble des liens ayant l'attribut rel avec la valeur nofollow et possédant un attribut class.

## Utiliser XPath avec le DOM en PHP

Utiliser *XPath* pour trouver les éléments qui vous intéressent au sein d'un document XML au lieu de parcourir directement ce dernier avec l'API DOM diminuera considérablement le nombre de lignes nécessaires et améliorera de fait la lisibilité de votre code.

Si l'on reprend la problématique de notre exemple précédent qui était de compter le nombre d'éléments ayant un attribut particulier en effectuant cette fois une requête *Xpath* ; le nombre de lignes nécessaires s'en trouvera tout de suite diminué.

*XPath* est donc au XML ce que le SQL est aux bases de données : un allié indispensable pour rédiger rapidement des requêtes claires afin de trouver les informations qui nous intéressent au sein d'un document XML.

## Conclusion

Vous avez désormais entre vos mains l'essentiel pour commencer à créer ou manipuler des documents XML et HTML à l'aide du Document Object Model avec PHP. Bien évidemment ce que nous venons de voir n'est qu'une rapide introduction à ce qu'il est possible de faire avec cette extension. Il existe bon nombre de fonctionnalités que nous n'avons pas abordées ici comme la validation de documents ou l'implémentation de doctypes. Les possibilités offertes sont immenses ; comme par exemple lorsque l'on commence à coupler le DOM avec *XPath*.

De plus, de par sa nature d'API l'apprentissage du *Document Object Model* se révèle très rapidement être un atout car vous retrouverez les mêmes classes et les mêmes méthodes en Java ou en JavaScript par exemple.

### Sur Internet

- *http://jerome.developpez.com/xmlxsl/xpath/* – Tutoriel sur la syntaxe Xpath,
- *http://xmlfr.org/w3c/TR/REC-DOM-Level-1/cover.html* – DOM Spécification de niveau 1, traduction française du document du W3C,
- *http://fr.php.net/manual/fr/book.dom.php* – Documentation de l'extension DOM de PHP.

**THOMAS GASC**

*Thomas Gasc travaille comme développeur web au sein de l'équipe de Thalasso-line (http://www.thalasso-line.com/) . Depuis 2004 il est également développeur php au sein de la communauté TitaXium (http://www.titaxium.com/).*
*Pour contacter l'auteur : thomas@gasc.fr*
*Son blog : http://methylbro.titaxium.org/*

# Joomla!™ Day

## BORDEAUX 21 MARS 2010

# Une journée exceptionnelle !

## Des conférences uniques

Pour la première fois les extensions incontournables de Joomla!™ telles que Joomfish, Community Builder, Sobi2,Jfusion, Flexicontent, JCE, le e-commerce avec Virtuemart seront présentées par leurs développeurs ou par des experts.
Seront aussi traités le référencement avec SEO Camp et la création de template.

## Une journée découverte

À toute personne souhaitant se mettre à niveau sur la création de sites sous Joomla!™, une salle sera dédiée à cette Journée Découverte et tous les sujets seront passés en revue et détaillés pas à pas : du téléchargement à la mise en ligne en passant par la hiérarchisation du contenu, le paramétrage du template, la mise en place des extensions, la sécurisation, la maintenance, etc.

***Une journée complète qui permettra à chacun de repartir avec les meilleures bases possibles pour réaliser son projet***

**Des bons plans sur l'hébergement, le transport, la restauration, info et inscriptions sur :**
**www.joomladay.fr**

## Quelques chiffres

**7000** téléchargements par jour de la version française de Joomla!™.
Près de **45000** personnes enregistrées sur le forum avec environ **1200** inscriptions chaque mois, et plus de **11000** nouveaux messages par mois.
Plus de **260** adhérents à l'AFUJ

## Quelques liens

Le portail francophone : **www.joomla.fr**
Le site des Joomgroupes : **www.joomgroupes.fr**
Le site d'aide : **http://aide.joomla.fr**
Le site dédié aux développeurs : **http://dev.joomla.fr**

**Adhérez à l'association**
**Participez aux projets joomla.fr**
**Devenez acteur du libre**

**www.afuj.fr**

AFUJ
Association Francophone des Utilisateurs de Joomla!™

# Nettoyer du code HTML avec PHP Tidy

HTML Tidy est un outil Open Source de vérification et de correction automatique de documents (X)HTML. Les fonctionnalités de cet outil sont disponibles depuis un script PHP, en utilisant la bibliothèque TidyLib et l'extension Tidy de PHP.

**Cet article explique :**

- Comment installer, configurer et utiliser l'extension Tidy pour PHP.

**Ce qu'il faut savoir :**

- Vous devez connaître les bases du langage PHP.

Niveau de difficulté


La bibliothèque *TidyLib* permet de vérifier et de corriger automatiquement des documents (X)HTML. Elle est disponible pour les systèmes Windows, Linux et Mac OS X. L'extension *Tidy* pour PHP utilise cette bibliothèque pour fournir plusieurs fonctions de nettoyage. Le code (X)HTML nettoyé peut provenir d'un fichier (X)HTML ou d'une chaîne de caractères. Il est également possible d'intercepter et corriger un code (X)HTML produit par un script PHP.

Voici quelques exemples de corrections et modifications réalisées par cet outil :

- ajout de balises fermantes,
- correction de l'imbrication d'éléments,
- ajout de guillemets dans les attributs,
- encodage de certains caractères en entités HTML,
- détection des éléments et attributs propriétaires ou inconnus,
- ajout de la déclaration de document, de l'élément racine `html`, de l'en-tête et du corps,
- modification de la casse,
- indentation,
- un élément `ul` sans élément `li` dans le code est remplacé par un attribut : *style="margin-left: 2em"*,
- transformation d'un document HTML en XHTML.

Vous allez dans un premier temps voir comment installer et configurer l'extension *Tidy*. Puis vous apprendrez à analyser et nettoyer des données provenant d'un fichier ou d'une chaîne de caractères. Vous verrez ensuite comment intercepter du code généré par PHP afin de le nettoyer avant de l'envoyer au navigateur. Vous verrez également comment obtenir des informations sur les erreurs et réparations effectuées. Enfin vous découvrirez comment utiliser la version objet de la bibliothèque.

## Installation de l'extension Tidy

Vous pouvez vérifier que l'extension *Tidy* est activée sur le serveur web en exécutant le script PHP : `<?php phpinfo();?>`. Si la bibliothèque *TidyLib* est installée et l'extension chargée, vous devriez voir apparaître un tableau de spécifications en dessous du titre *tidy* (Figure 1). Si vous obtenez le tableau, vous pouvez passer directement à la section suivante de cet article.

Si vous n'obtenez pas de section *tidy* quand vous exécutez le script PHP, ce peut être parce que l'extension n'est pas chargée dans le fichier de configuration *php.ini*. Si vous êtes sous un système Windows, vérifiez que la ligne ci-après est présente dans le fichier *php.ini* et qu'elle n'est pas commentée :

```
extension=php_tidy.dll
```

Pour les utilisateurs Linux ou Mac OS X, l'extension est `php_tidy.so` :

```
extension=php_tidy.so
```

Une fois la ligne ajoutée ou le commentaire supprimé, redémarrez le serveur *Apache* pour que la modification soit prise en compte. Si vous n'obtenez toujours pas le tableau de la Figure 1, assurez-vous que la bibliothèque *TidyLib* est présente sur votre serveur web. Si ce n'est pas le cas, téléchargez la bibliothèque sur le site officiel du projet *TidyLib*, dont l'adresse est donnée dans le cadre *Sur Internet*. Si vous travaillez sous Linux ou Mac OS X, vérifiez que PHP a été compilé avec l'option `--with-tidy`.

## Configuration de l'extension Tidy

Deux directives du fichier *php.ini* permettent de paramétrer l'extension *Tidy*. La directive

tidy

| Tidy support | enabled |
|---|---|
| libTidy Release | 22 March 2008 |
| Extension Version | 2.0 ($Id: tidy.c,v 1.66.2.8.2.24.2.14 2009/01/06 23:45:16 iliaa Exp $) |

| Directive | Local Value | Master Value |
|---|---|---|
| tidy.clean_output | no value | no value |
| tidy.default_config | no value | no value |

**Figure 1.** *Informations sur l'extension Tidy*

`tidy.clean_output` active ou désactive le nettoyage automatique des documents générés. Il est recommandé de donner la valeur `Off` à cette directive, sinon tout code généré par PHP sera nettoyé automatiquement. Si la valeur `On` est donnée, il devient alors impossible de générer des fichiers PDF, des fichiers pour tableurs ou encore des images. Par exemple, si un script PHP qui crée une image et l'envoie au navigateur, est exécuté, alors le message suivant est affiché :

*The image cannot be displayed, because it contains errors.*

La seconde directive `tidy.default_config` permet de définir un fichier par défaut, contenant des options de configuration, qui seront utilisées lors du nettoyage du code (X)HTML. La valeur donnée à la directive correspond au chemin du fichier sur le disque du serveur. Le fichier de configuration contient une ligne par option, chaque ligne suit le format `nom:valeur`. Par exemple, pour obtenir un code XHTML, indenté avec quatre caractères espaces par indentation, il suffit de définir les options suivantes :

```
indent-spaces:4
output-xhtml:true
indent:true
```

Si un fichier de configuration est fourni à la directive `tidy.default_config`, alors tous les traitements de code (X)HTML réalisés avec l'extension *Tidy* utiliseront ces paramètres. Il est possible de régler des options directement dans un script PHP, qui seront spécifiques à ce fichier. Elles sont alors définies dans un tableau d'options, passé en argument à la fonction qui traite les données dans le script. Les options de l'exemple précédent peuvent être définies sous la forme d'un tableau PHP :

```
$config = array(
    'indent-spaces' => 4,
    'output-xhtml' => true,
    'indent' => true
);
```

Ce tableau `$config` définit des options dans le cas où aucun fichier de configuration n'a été fixé dans la directive `tidy.default_config`. Le paramétrage d'options directement dans le code PHP permet également d'ajouter des options à celles par défaut, ou de redéfinir la valeur de certains paramètres du fichier de configuration.

Dans cet article, seules les trois options présentées précédemment sont utilisées. De nombreuses autres options sont disponibles, vous trouverez la liste complète des paramètres de configuration à l'adresse : *http://tidy.sourceforge.net/docs/quickref.html*.

## Nettoyer des données stockées dans un fichier

Dans cette section vous allez apprendre à nettoyer un fichier (X)HTML depuis un script PHP, en utilisant les fonctions de l'extension *Tidy*. Ceci permet soit d'envoyer un fichier corrigé au navigateur, soit de traiter un ou plusieurs fichiers et de les enregistrer sur le disque après correction.

Une première méthode effectue le nettoyage du code (X)HTML en trois étapes :

- lecture et analyse du fichier avec la fonction `tidy_parse_file`,
- correction du code avec la fonction `tidy_clean_repair`,
- récupération du code nettoyé dans une chaîne de caractères avec la fonction `tidy_get_output`.

La fonction `tidy_parse_file` lit et analyse les données d'un fichier (X)HTML, dont le chemin d'accès sur le disque est passé en premier argument (chemin absolu sur le disque dur, ou relatif). La fonction accepte également de lire des fichiers distants, une URL peut donc être fournie en premier argument. La fonction retourne une instance de la classe `Tidy`. Cet objet est utilisé dans la fonction `tidy_clean_repair` qui nettoie et corrige les données (X)HTML, elle retourne la valeur `true` en cas de succès, la valeur `false` sinon. La fonction `tidy_get_output` prend en paramètre l'objet retourné par `tidy_parse_file` et retourne une chaîne de caractères contenant le code (X)HTML nettoyé.

Les exemples de cet article utilisent tous le code HTML du fichier *test.html*, qui est présenté dans la Figure 2. Comme vous pouvez le constater, ce fichier comporte de nombreuses erreurs (mauvaise imbrication d'éléments, absence de balises fermantes, ...).

Le script PHP du Listing 1 permet de nettoyer ce code. Il lit et analyse le fichier *test.html*, le répare et envoie les données corrigées au navigateur. Le code source de la page générée par le Listing 1 est présenté dans la Figure 3.

Aucune option de configuration n'a été définie dans cet exemple, aucun fichier par

```
<html>
<p>un <i> test <em></i> de</em>
<div class=test> html tidy.
```

**Figure 2.** *Fichier test.html incorrect*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title></title>
</head>
<body>
<p>un <i>test</i> <em>de</em></p>
<div class="test">html tidy.</div>
</body>
</html>
```

**Figure 3.** *Fichier test.html corrigé par l'extension Tidy*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title></title>
    </head>
    <body>
        <p>
            un <i>test</i> <em>de</em>
        </p>
        <div class="test">
            html tidy.
        </div>
    </body>
</html>
```

**Figure 4.** *Fichier test.html corrigé avec des options de configuration*

**Listing 1.** *nettoyage_fichier.php*

```
<?php
$tidy = tidy_parse_file('test.html');
tidy_clean_repair($tidy);
echo tidy_get_output($tidy);
?>
```

**Listing 2.** *nettoyage_chaine.php*

```
<?php
$html = "<html><p>un <i> test <em></i> de</em><div class=test> html
tidy.";
$tidy = tidy_parse_string($html);
tidy_clean_repair($tidy);
echo tidy_get_output($tidy);
?>
```

**Listing 3.** *interception_code_genere.php*

```
<?php
ob_start();
?>
<html><p>un <i>test <em></i> de</em>

<?php
echo '<div class=test>html tidy.';
$page_html = ob_get_clean();
$config = array(
    'indent-spaces' => 4,
    'output-xhtml' => true,
    'indent' => true
);
$tidy = tidy_parse_string($page_html, $config);
tidy_clean_repair($tidy);
echo tidy_get_output($tidy);
?>
```

**Listing 4.** *affichage_erreurs.php*

```
<?php
header('Content-type:text/plain');
$tidy = tidy_parse_file('test.html');
echo "nombre d'erreurs non reparables : ", tidy_error_count($tidy), "\n";
echo "nombre de warning : ", tidy_warning_count($tidy), "\n";
echo "buffer : \n", tidy_get_error_buffer($tidy), "\n";
?>
```

**Listing 5.** *version_objet.php*

```
<?php
$tidy = new Tidy();
$tidy->parseFile('test.html');
$tidy->cleanRepair();
echo $tidy;
?>
```

défaut n'est spécifié dans la directive du fichier *php.ini*. Le code généré est donc du HTML, sans indentation. Il est possible de définir des options de configuration, en deuxième paramètre de la fonction `tidy_parse_file`. Ces options sont stockées soit dans un tableau associatif, comme expliqué dans la section *Configuration de l'extension Tidy*, soit dans un fichier de configuration. Ce dernier suit le même format que le fichier par défaut fourni à la directive `tidy.default_config`. Dans le second cas, le chemin d'accès au fichier de configuration sera donc fourni à la fonction en deuxième argument (chemin complet du fichier sur le disque dur). Un troisième argument facultatif permet de préciser l'encodage.

Par exemple la ligne de code suivante utilise le tableau d'options `$config` défini précédemment, et indique que l'encodage est de l'*UTF-8* :

```
$tidy = tidy_parse_file('test.html',
$config, 'utf8');
```

Une méthode alternative permet d'effectuer les trois étapes précédentes en une seule instruction. La fonction `tidy_repair_file` lit, analyse et répare les données du fichier qu'elle reçoit en premier argument. Elle accepte les mêmes arguments que la fonction `tidy_parse_file` et retourne les données corrigées dans une chaîne de caractères. En appliquant cette méthode, il sera impossible d'utiliser les fonctions de l'extension *Tidy* qui prennent en paramètre un objet *Tidy*. Il ne sera par exemple pas possible d'obtenir des informations sur les erreurs et corrections effectuées, comme vous allez apprendre à le faire dans la section *Afficher les erreurs et réparations*. Le code suivant répare le fichier *test.html* de la Figure 2, en utilisant les options de configuration du tableau `$config` :

```
echo tidy_repair_file('test.html',
$config);
```

Le résultat obtenu est présenté dans la Figure 4.

## Nettoyer des données stockées dans une chaîne

Dans cette section vous allez apprendre à nettoyer une chaîne de caractères contenant un code (X)HTML depuis un script PHP. Comme dans la section précédente de cet article, deux méthodes de nettoyage sont disponibles. La première méthode est effectuée en trois étapes, elle utilise `tidy_parse_string`, `tidy_clean_repair` et `tidy_get_output`.
La fonction `tidy_parse_string` lit et analyse les données de la chaîne de caractères passée en premier argument. Des options de configuration peuvent être fournies en second argument (tableau associatif ou nom du fichier où les options sont stockées). Il est également possible de préciser l'encodage (troisième argument). La fonction retourne une instance de la classe `Tidy`. Cet objet est utilisé par les fonctions `tidy_clean_repair` et `tidy_get_output` qui nettoient les données (X)HTML et les retournent dans une chaîne de caractères.

Le Listing 2 donne un exemple d'utilisation de ces fonctions, le code HTML à nettoyer est placé dans la chaîne `$html`. Le résultat obtenu est le même que celui de la Figure 3. Il est possible d'effectuer toutes ces opérations en une instruction en utilisant la fonction `tidy_repair_string` qui lit, analyse, répare et retourne les données corrigées dans une chaîne de caractères. Elle accepte les mêmes arguments que la fonction `tidy_parse_string`.

## Nettoyer les données envoyées par un script

Si vous devez rendre conforme pour le W3C, ou migrer vers XHTML, un site dont les pages HTML sont générées par PHP, alors l'extension *Tidy* peut vous faciliter la tâche. Elle permet d'obtenir très rapidement des pages (X)HTML conformes. Le nettoyage automatique des données peut être effectué en utilisant un fichier *.htaccess* ou en interceptant le code HTML généré par le script PHP. Ces approches de nettoyage automatique fournissent une solution temporaire de nettoyage à moindre coût, en attendant que la correction de chaque script PHP soit effectuée.

### Utilisation d'un fichier .htaccess

La directive `tidy.clean_output`, qui active le nettoyage automatique des données générées par les scripts PHP, a été présentée dans la section intitulée *Configuration de l'extension Tidy*. Vous avez vu que cette directive ne doit pas être activée dans le fichier de configuration *php.ini*, afin d'éviter que l'exécution de scripts PHP générant des données non HTML (images, fichiers PDF, ...), ne produise des erreurs.

Si le projet à corriger ou migrer ne contient pas de tels scripts, vous pouvez attribuer la valeur `on` à cette directive dans un fichier *.htaccess* situé dans le répertoire racine du projet à corriger ou migrer. Pour fixer cette valeur booléenne, vous devez faire précéder le nom de la directive par la chaîne `php_flag` :

```
php_flag tidy.clean_output on
```

Si certains scripts du projet génèrent des images ou autres documents binaires, vous pouvez localement désactiver la directive, soit en lui attribuant la valeur `off` dans un *.htaccess* d'un sous-dossier qui ne contient que des scripts de génération de documents non HTML, soit en spécifiant les fichiers qui ne doivent pas être corrigés automatiquement. Par exemple, le fichier *.htaccess* ci-après indique qu'il n'y aura pas de correction automatique pour tout fichier commençant (symbole accent circonflexe) par le mot graph, tous les autres scripts PHP seront analysés et corrigés par l'extension *Tidy* :

```
php_flag tidy.clean_output on
<Files ~ "^graph">
php_flag tidy.clean_output off
</Files>
```

Avec l'utilisation d'un *.htaccess* vous pouvez ainsi obtenir la correction automatique intégrale d'un projet, sans modifier une seule ligne de code. Il faut noter que cette méthode n'est possible que si le fichier de configuration du serveur web *Apache* autorise l'utilisation de fichiers *.htaccess*.

### Interception du code

Il est possible de stocker les données envoyées vers la sortie standard par le script PHP et de les nettoyer avec l'extension *Tidy*, avant de les envoyer au navigateur. La fonction PHP `ob_start` permet de stocker les données envoyées vers la sortie standard (*ob* est une abréviation de *output buffer*). Sans l'utilisation de cette fonction, les données sont envoyées directement au navigateur. Une fois toutes les données interceptées, la fonction `ob_get_clean` retourne le contenu du tampon de sortie, vide le tampon et désactive le stockage des données envoyées vers la sortie standard.

### Sur Internet

- *http://tidy.sourceforge.net* – Site officiel de la bibliothèque Tidy,
- *http://www.php.net/manual/en/book.tidy.php* – Extension Tidy pour PHP,
- *http://validator.w3.org* – Outil de validation de documents (X)HTML du W3C.

Le contenu du tampon peut alors être traité, puis envoyé au navigateur.

Le Listing 3 montre comment stopper l'envoi des données vers le navigateur et comment les récupérer à la fin du script, afin de nettoyer le code HTML généré dans le script. Tout le code HTML situé entre les appels aux fonctions `ob_start` et `ob_get_clean` est intercepté et stocké. La fonction `ob_get_clean` retourne les données interceptées, celles-ci sont stockées dans la variable `$page_html`. Le code contenu dans cette chaîne de caractères est nettoyé et envoyé au navigateur en utilisant la première méthode présentée dans la section précédente. Le résultat obtenu est celui de la Figure 4.

## Afficher les erreurs et réparations

Des fonctions fournissent des informations sur les erreurs et alertes obtenues lors de l'analyse et la réparation des données. Les fonctions `tidy_error_count` et `tidy_warning_count` retournent respectivement le nombre d'erreurs non réparables et le nombre d'alertes obtenus lors de l'analyse des données. Elles prennent en paramètre un objet *Tidy* retourné soit par la fonction `tidy_parse_file`, soit par la fonction `tidy_parse_string`. La fonction `tidy_get_error_buffer` retourne une chaîne contenant les numéros de lignes et les explications pour chaque problème rencontré lors de l'analyse. Elle prend elle aussi en paramètre un objet *Tidy*.

Le code du Listing 4 affiche le nombre d'erreurs non réparables et d'alertes rencontrées lors de l'analyse des données du fichier *test.html*. Il indique également le détail des erreurs trouvées et des corrections effectuées. La fonction PHP `header`, en début de Listing 4, indique au navigateur que le contenu envoyé est du texte brut qu'il ne faut pas interpréter. Le résultat affiché par le navigateur est donné ci-après :

```
nombre d'erreurs non reparables : 0
nombre de warning : 7
buffer :
line 1 column 1 - Warning: missing
<!DOCTYPE> declaration
line 2 column 1 - Warning: inserting
implicit <body>
line 2 column 16 - Warning: replacing
unexpected i by </i>
line 2 column 24 - Warning: inserting
implicit <em>
line 3 column 1 - Warning: missing </div>
line 2 column 1 - Warning: inserting
missing 'title' element
line 2 column 16 - Warning: trimming
empty <em>
```

## Version Objet

L'extension Tidy fournit des fonctions procédurales mais également une classe `Tidy` dont les méthodes ont le même comportement que leurs versions procédurales. Les noms de méthodes sont similaires, ce sont les noms des fonctions présentées dans cet article, dans lesquels les caractères soulignés ont été supprimés et la première lettre de chaque mot est passée en majuscule. La documentation officielle de PHP donne la liste des méthodes de la classe.

Le Listing 5 est une réécriture du Listing 1 en PHP objet. Une instance de la classe `Tidy` est créée dans la première instruction. L'invocation des méthodes `parseFile` et `cleanRepair` a pour effet d'analyser et réparer le fichier test.html. L'affichage de l'objet `$tidy` est réalisé simplement avec l'instruction `echo`. Le résultat obtenu est celui de la Figure 3.

## Conclusion

Vous avez appris à générer du code (X)HTML valide pour le W3C depuis un script PHP grâce à l'extension *Tidy*. Celle-ci permet de corriger rapidement et simplement d'anciens projets générant des codes HTML incorrects ou de les transformer en XHTML. Ce type de traitement automatique est utile pour obtenir des résultats valides rapidement, en attendant la correction directe des erreurs (X)HTML dans les scripts PHP.

**CÉCILE ODERO**
**MAGALI CONTENSIN**

*Cécile Odero est spécialisée dans la conception et le développement d'applications web en PHP. Elle travaille au CNRS comme ingénieur en développement et déploiement d'applications.*
*Contact : cecile.odero@gmail.com*
*Magali Contensin, auteur du livre Bases de données et Internet avec PHP et MySQL, est chef de projet en développement d'applications au CNRS. Elle enseigne depuis plus de dix ans le développement d'applications web à l'Université.*
*Contact : http://magali.contensin.online.fr*

# Réalisez votre serveur d'objets avec PHP

Un serveur objet permet plusieurs points à savoir l'allègement du serveur web ainsi que d'éviter la réécriture dans le cas de base de données qui servira pour plusieurs applications.

**Cet article explique :**

- Comment mettre en place un serveur d'objets.

**Ce qu'il faut savoir :**

- Bonnes base de PHP.
- Connaissances en bases de données/SQL.
- Utilisation de PDO.

Niveau de difficulté


Nous sommes à une époque où l'on parle de plus en plus de web-services. Imaginez qu'un jour vous ayez une application web à développer et que l'on mette à votre disposition 3 machines de puissance moyenne. Traditionnellement, l'une servira de serveur web, l'autre de serveur de base de données. Il serait intéressant d'ajouter la 3ème machine aux cotés de ces serveurs pour qu'il traite les plus grosses fonctions et ainsi alléger en fonctions le serveur web. C'est ce que nous allons voir dans cet article d'une manière plus simple qu'on aurait pu le penser!

## Communication avant tout

Si vous regardez la Figure 1, vous remarquerez l'organisation de nos 3 machines : un premier serveur web que nous appellerons *Web-srv* reçoit une requête d'un client (un navigateur internet par exemple). *Web-srv* appelle le 2nd serveur *Obj-srv* pour réaliser les gros calculs et si besoin fait appel lui même à la base de données qui elle, est sur un troisième serveur que nous appellerons *Sql-srv*. Une fois que *Obj-srv* aura sa réponse prête, il la transmettra à *Web-srv* et ce dernier génèrera la page en fonction des données reçues.

Cette architecture possède un avantage indéniable : le serveur le plus exposé, à savoir le serveur web ne possède plus les identifiants de connexion à la base de données. Ainsi, le risque de défaillance suite à des attaques de type *SQL-Injection* visant la base de données se retrouve limité même si cela reste possible. Nous allons prendre pour exemple ici un système de comptes bancaires. La base de données contient une seule table extrêmement simplifiée : comptes.

Chaque compte possède un identifiant, une partie Nom, prénom, mot de passe malgré le fort taux de redondance d'informations, *crédit_compte* contient l'argent sur le compte, et date_activité retient la dernière date d'activité. Le Listing 1 contient la requête Sql permettant la création de la table Comptes. Pour que notre serveur *Obj-srv* discute avec le serveur de base de données (*Sql-srv*), nous utiliserons PDO.

Le Listing 2 présente la construction de notre objet PDO ainsi qu'une requête listant l'ensemble des comptes sans le mot de passe. Ce script serait à placer sur le serveur *Obj-srv*. Le résultat de ce listing pourrait être le suivant :

- Compte n°1 : Pierre Dupond Possède : 1500.
- Compte n°2 : Paul Dupont Possède : 4500.

**Tableau 1.** *Liste des comptes*

| Id_compte | Prenom_user | Nom_user | pass_user | credit_compte | date_activite |
|---|---|---|---|---|---|
| 1 | Pierre | Dupond | K3Xpdb@R | 1500 | 14.02.10 |
| 2 | Paul | Dupont | MAPàuaèQ | 4500 | |

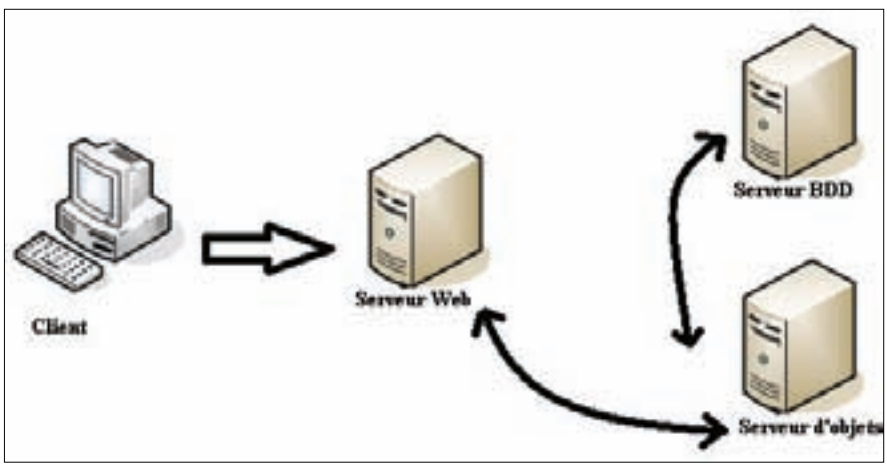



**Figure 1.** *Organisation des serveurs*

**Listing 1.** *Création de la table Comptes*

```
CREATE TABLE 'comptes' (
'id _ compte' INT NOT NULL AUTO _
INCREMENT PRIMARY KEY ,'Nom _
user' VARCHAR( 45 ) NOT NULL
,'Prenom _ user' VARCHAR( 35 ) NOT
NULL ,'Pass _ user' VARCHAR( 20 )
NOT NULL ,'credit _ compte' INT
NOT NULL ,'date _ activite' BIGINT
NOT NULL
);
```

Nous travaillerons avec seulement ces deux comptes pendant la suite de l'article.

Au sein d'une architecture web, chaque machine possède assez souvent un petit nom. Dans notre cas j'ai supposé que notre base de données pouvait être disponible sur la machine *srvsql.monsite.com*. Habituellement, on indique *localhost*, mais comme notre base de données n'est plus sur la même machine, il faut modifier en conséquence.

## Communication

Pour permettre la communication du serveur web (*Web-srv*) à notre serveur objet(*Obj-srv*) il va nous falloir vérifier un paramètre dans *Php.ini*. Celui-ci s'appelle `allow_url_fopen`. En effet, l'astuce de la communication entre serveurs provient de cette option qui, lorsqu'elle est activée, permet de lire le code source d'un page générée. Si dans php.ini, le paramètre est à 0, passez le à 1 et redémarrez PHP. Vous devez dès lors entrevoir le fonctionnement de notre architecture : l'utilisateur va demander une page php sur notre serveur *Web-srv*, celle-ci pourrait avoir besoin d'informations provenant de la base de données, elle fait donc appel à *Obj-srv* en transmettant les paramètres nécessaires. La page appelée sur notre serveur objet va vérifier la cohérence des paramètres reçus. Dans le cas négatif, *Obj-srv* renverra un message d'erreur. Dans le cas positif, il consultera la base de données et renverra le résultat.

Renvoyer est dans notre situation un grand mot car nous devrions dire que si nous appelions le script qui est sur *Obj-srv* via un navigateur, le résultat sera donné par un echo et donc affiché à l'écran. Notre serveur web se contentera donc de lire ces données. Au point où nous en sommes, nous pouvons déjà vérifier le bon fonctionnement de l'architecture : la base de données doit donc comporter notre table 'compte' et notre Listing 2 doit être sur notre serveur objet dans le script test.php par exemple.

Il nous manque plus qu'à appeler ce `'test.php'` via `file_get_contents()` à partir de notre serveur web. Le Listing 3 qui sera sur notre serveur web réalise cette

**Listing 2.** *Récupération des comptes test.php sur Obj-srv*

```
<?php
//on créé notre objet PDO
$pdo = new PDO('mysql:host=srvsql.monsite.com;dbname=base _ test',
'utilisateur','mot _ de _ passe' );

//on affiche la liste des comptes
echo '<ul>';
foreach( $pdo->query('SELECT Id _ compte, Prenom _ user, Nom _ user, credit _
compte FROM comptes') as $row ) {
        echo '<li> Compte n°'. $row['Id _ compte'].': '. $row['Prenom _ user']. '
'. $row['Nom _ user']. ' Possède: '.$row['credit _ compte']. '</li>';
}
echo '</ul>';
?>
```

**Listing 3.** *Script appellant notre serveur objets*

```
<?php
$comptes = file _ get _ contents('http://www.monsite.com/test.php'); echo
$comptes;
?>
```

**Listing 4.** *test.php avec vérification clef*

```
<?php
$verif _ cle = 'sdvbhjcblqdscvHVHçççé756éàé';
if ( isset($ _ GET['cle']) AND ($ _ GET['cle']==$verif _ cle ))
{
        //on créé notre objet PDO
        $pdo = new PDO('mysql:host=srvsql.monsite.com;dbname=base _ test',
'utilisateur','mot _ de _ passe' );

        //on affiche la liste des comptes
        echo '<ul>';
        foreach( $pdo->query('SELECT Id _ compte, Prenom _ user, Nom _ user,
credit _ compte FROM comptes') as $row ) {
        echo '<li> Compte n°'. $row['Id _ compte'].': '. $row['Prenom _ user'].'
'. $row['Nom _ user'].' Possède: '.$row['credit _ compte']. '</li>';
        }
        echo '</ul>';
}
```

**Listing 5.** *Script appellant notre serveur objets avec clé de vérification*

```
<?php
$cle = 'sdvbhjcblqdscvHVHçççé756éàé';
$comptes = file _ get _ contents('http://www.monsite.com/test.php?cle='.$cle);
echo $comptes;
?>
```

**Listing 6.** *Service avec reponse en tableau sérialisé*

```
<?php
$verif _ cle = 'sdvbhjcblqdscvHVHçççé756éàé';
$resultat['exception']=NULL; //on initialise nos erreurs à null.
if ( isset($ _ GET['cle']) AND ($ _ GET['cle']==$verif _ cle ))
{
        try
        {
                //on créé notre objet PDO
                $pdo = new PDO('mysql:host=srvsql.monsite.com;dbname=base _
test', 'utilisateur','mot _ de _ passe' );
                //lève une exception en cas d'erreur.
                $pdo->setAttribute(PDO::ATTR _ ERRMODE, PDO::ERRMODE _
EXCEPTION);
                $sql = 'SELECT Id _ compte, Prenom _ user, Nom _ user,
credit _ compte FROM comptes';
                $resulset = $pdo->query($sql);
                $resultat['return'] = $resulset->fetchAll(PDO::FETCH _ ASSOC);
        }catch (PDOException $e) {
                                $resultat['exception']= $e->getMessage();
        }
}
else
{
        $resultat['exception'] = 'Il semble qu'un problème de communication
soit survenu';
}
//enfin on sérialise notre résultat
echo serialize( $resultat);
?>
```

**Listing 7.** *serveurweb.php*

```
<?php

//clé de verification
$cle = 'sdvbhjcblqdscvHVHçççé756éàé';
$recup_comptes = file_get_contents('http://www.monsite.com/test.
php?cle='.$cle);

// Gestion de la récupération du résultat
if ($recup_comptes== False) // erreure survenue lors du File_get_contents
{
         $recup_resultat['exception']= 'Le service demandé ne répond pas.
Merci de ré-essayer ulterieurement.';
}
else
{

         // le serveur a répondu on peut donc désérialiser sa réponse
         $recup_resultat = unserialize($content);
         if( false === $recup_resultat )// si la désérialisation s'est mal
passée
         {
        $recup_resultat['exception']= 'Un problème est survenu lors de la
dé-sérialisation des données.';
    }
         elseif( !is_array($recup_resultat) or empty($recup_
resultat['return']) )
         {

        //  si la dé-sérialisation n'a pas retourné un tableau, ou bien qu'il
n'y ai aucune réponse
        $recup_resultat['exception']= 'Le résultat reçu n\'est pas conforme
au résultat attendu.');
    }
}

// arrivée ici notre réponse est déjà „mise en forme", il nous reste plus
qu'a afficher la réponse en fonction du résultat.
if( $recup_resultat['exception'] != null)
{

    // On lève l'exception
         throw new Exception( $recup_resultat['exception'] );
}
else
{
    if( !is_array( $recup_resultat['return']) )
         {
        throw new Exception( 'Le résultat reçu n\'est pas un tableau de
données.' );
    }
         else
         {

        // Si le tableau est vide, il n'y a pas comptes
        if( empty($recup_resultat['return']))
                 {
                         echo '<p>Aucun compte n\'a été retourné.</p>';
                 }
        else {

            // Le résultat a passé tous les test et l'on peut l'afficher.
            echo '<ul>';
            foreach( $recup_resultat['return'] as $compte ) {
                                echo '<li> Compte n°'.$compte['Id_
compte']. ': '.htmlspecialchars($compte['Prenom_user']).'
'.htmlspecialchars($compte['Nom_user']).' Possède: '.$compte['credit_
compte']. '</li>';
            }
            echo '</ul>';
        }
    }
}

?>
```

action. Créez une page *index.php* sur votre serveur web qui possède le contenu du Listing 3 en modifiant l'adresse du script à exécuter. Le résultat devrait être le même que lors de l'exécution directe du script test.php du Listing 2. En cas de message d'erreur, pensez à vérifier l'option `allow_url_fopen` de `php.ini`.

### Arguments et sécurité

Notre application vient de réaliser ses premiers pas et ils se révèlent prometteurs. Cependant l'on aimerait améliorer 2 points : Pouvoir transmettre des paramètres à notre serveur objet pour *travailler* sur un compte précis. Par exemple celui possédant l'identifiant n°1. Il serait aussi intéressant d'améliorer la sécurité de l'application.

Nous allons donc définir une clé d'accès à notre serveur objet. Considérons notre clé comme : `$cle = 'sdvbhjcblqdscvHVHçççé756éàé';`

Cette clé pourrait contenir tout type de caractères. Si le serveur d'objet reçoit un paramètre appelé *cle* qui est égal à la valeur si dessus, il exécutera la fonction demandée, sinon il renverra un message d'erreur. Nous allons dans un premier temps contacter notre serveur d'objet à partir du serveur web en transmettant notre paramètres par la méthode `GET`.

Le Listing 4 reprend notre script *test.php* du serveur objet en implémentant la vérification de la clé. Le Listing 5 fait appel au Listing 4 en lui envoyant comme paramètre la clé. Ce Listing 5 est bien entendu sur notre serveur web. Pour des raisons de sécurité, il vous faudra vérifier que les register globals soient désactivées aussi bien sur *Web-srv* que sur *Obj-srv*, et même mettre la clé valide dans une constante dans le code du serveur objet. On pourrait pousser le vice jusqu'à la coder en *sha256* avec la fonction `hash()`.

## Différence de réponse

Nous venons de voir comment transmettre simplement nos arguments avec la méthode `GET`. Ce qui serait maintenant intéressant, c'est de pouvoir recevoir des réponses qu'on puisse traiter car dans l'état actuel des choses, le serveur web ne peut pas faire la différence entre une réponse valide, et un message d'erreur.

Nous allons utiliser un procédé bien connu : la sérialisation. Celle-ci permettra à notre serveur web de différencier les réponses et ainsi récupérer plusieurs valeurs en retour. La réponse de notre serveur objet sera un tableau contenant deux valeurs. La première contiendra un possible message d'erreur sinon elle sera *NULL*. La seconde valeur sera le retour attendu. Le Listing 6 traite ces nouveautés . Nous avons ici en plus une gestion des erreurs retournées par PDO. Le Listing 7 représente le code source qui peut être exécuté sur le serveur web faisant appel à notre Listing 6. Ce code désérialise les informations recueillies, vérifie l'existence d'une erreur et affiche un résultat en conséquence. En cas d'erreur,

un mail peut être envoyé à l'administrateur pour corriger ce problème.

Détaillons ce dernier code. Comme nous le faisons depuis le début, nous utilisons d'abord la fonction `file_get_content()` pour récupérer le résultat affiché d'une page web en envoyant en paramètre `GET` une clé de vérification permettant d'assurer que le script faisant la demande est autorisé à la faire. Le traitement de la réponse va se faire en 2 étapes : d'abord on vérifie que l'on ait bien reçu une réponse sinon cela signifie que le serveur objet n'a pas répondu. En cas de réponse l'on tente la dé-sérialisation. Si celle-ci se passe bien nous devrions avoir un tableau, donc la case *return* n'est pas vide. La seconde partie du traitement consiste à afficher un résultat. Celui-ci peut dépendre d'une exception ou alors de la valeur de return. Si la case exception du tableau n'est pas nulle, alors il y a une exception à lever. Sinon, on passe au traitement de la valeur return. Dans notre cas, la case return doit être un jeu de réponses, donc si elle ne contient pas un tableau, on lève une exception. Si le tableau est vide, cela signifie qu'il n'y a aucune réponse à notre requête. Il est alors possible d'afficher nos réponses via un foreach.

## Conclusion

Ce dernier code peut être un peu fastidieux à relire, mais il se révèle en fin de compte très simple. La mise en place d'une architecture de ce type peut sembler inapproprié à ses débuts mais elle révèle toute sa puissance lors qu'on doit ajouter de nouvelles applications qui utiliseront la même base de données. Plus besoin de gérer de connexion à la base de données à partir du serveur web, simplement appeler les bons fichiers sur votre serveur objet. Il serait cependant judicieux de faire 2 ou 3 choix : ne pas utiliser la sérialisation contrairement à l'article qui n'a pour but d'exemple car même si beaucoup de langages peuvent dé/sérialiser des données, ils ne le font pas de la même manière. Ainsi, une données sérialisée en Php ne pourra pas être dé-sérialisée par C++, Java ou C#. Afin de permettre une interopérabilité de votre serveur objet, il serait intéressant d'utiliser un langage de balisage tel que XML pour rendre sa réponse. Ainsi, votre serveur objet pourra être appellé via le site web (de la banque dans notre cas, pour les clients qui veulent gérer leur compte en ligne), ou alors via des applications bureau avec un programme écrit en C++, java ou autre sur les pc des banquier ou GAB (guichet automatique bancaire). Aujourd'hui, presque n'importe quel langage peut traiter un fichier xml et donc cela deviendrait un atout. Un deuxième choix serait l'utilisation de la méthode `POST` pour transmettre divers arguments au serveur objet . Une troisième amélioration se présente lors de la multiplication des applications existantes avec l'utilisation de SSL et une base de données contenant les clés d'identification. Notre exemple de banque se prête à la situation : un site web, une application pc. Plus besoin de faire le vérification de sécurité sur l'application, le serveur objet s'en chargera. Nous vous invitons à venir partager vos améliorations de ce *bout de code* sur le forum à *http://phpsolmag.org/fr/forum/category/518-scripts*, aussi bien d'un point de vue fonctionnel que sécurité.


**NICOLAS TURMEAU**

*Nicolas Turmeau est un étudiant à l'Institut d'Informatique Appliqué de Laval. Il utilise le langage PHP tous les jours depuis plusieurs années et participe à divers projets en ligne dont Numénor Online. Il aime partager ses connaissances et écrit par passion différents livres liés ou pas à l'informatique.*

*Contact : nturmeau@iia-laval.fr*

# Créer une application multi-Mobile avec HAWHAW

Les applications multi-Mobile offrent un contenu accessible par la plupart des plateformes mobiles (téléphone portable, PDA, etc) mais aussi par des navigateurs standards et les navigateurs vocaux. Ce contenu est formaté différemment en fonction des exigences spécifiques à chaque plate-forme.

**Cet article explique :**

- Comment utiliser le toolkit HAWHAW et PHP pour créer une application web multi-Mobile.

**Ce qu'il faut savoir :**

- Une connaissance des notions de base de PHP est la seule condition requise.

Niveau de difficulté


L'une des préoccupations majeures des développeurs des sites mobiles est de rendre le code compatibilité aux différents navigateurs. Préparer une version de code adapté à chaque navigateur est une tâche presque impossible vu le volume horaire que cela nécessiterait. Heureusement qu' il existe des outils permettant de simplifier la tâche aux développeurs. Dans cet article, nous nous intéresserons au toolkit nommé HAWHAW.

## Présentation de HAWHAW

HAWHAW est l'acronyme de *HTML and WML Hybrid Adapted Webserver*. Bien que cela semble drôle et étrange à la prononciation; ce puissant outil permet de mettre en place des portails mobiles universels et accessibles à partir de n'importe quelle plateforme.

## Caractéristiques de HAWHAW

HAWHAW supporte les standards suivants :

- XHTML 1.1,
- WML 1.X,
- WAP 2.0,
- iMode (cHTML),
- HDML,
- MML,
- VoiceXML.

Il est supporté par les navigateurs suivants (la liste n'est pas exhaustive) :

- navigateur des téléphones portables (WAP),
- navigateur de *Smartphone* (WAP 2.0),
- navigateur de PDA,
- *Opera Mini*,
- *AvantGo*,
- *iPhone*,
- *Android*,
- *Minimo*,
- *PIE*,
- *Blackberry*,
- *Nintendo DS*,
- *Internet Explorer, Firefox, Opera, Safari, Chrome, Lynx*.

HAWHAW est intégralement distribué en Open Source, il est très bien documenté et souvent mis à jour. Le seul hic est que la documentation est intégralement en anglais car la communauté francophone reste discrète et peu active.

## Structure de HAWHAW

La bibliothèque HAWHAW est constituée des éléments suivants :

- *hawhaw.inc* – la librairie des classes PHP,
- HAWHAW XML – langage de balisage,
- HAWXY – le proxy HAWHAW,
- HAWHAW.NET – la solution HAWHAW pour ASP.NET,
- HawTags – le *HAWHAW JSP Tag Library*.

La Figure 1 montre l'écosystème de cette boite à outils.

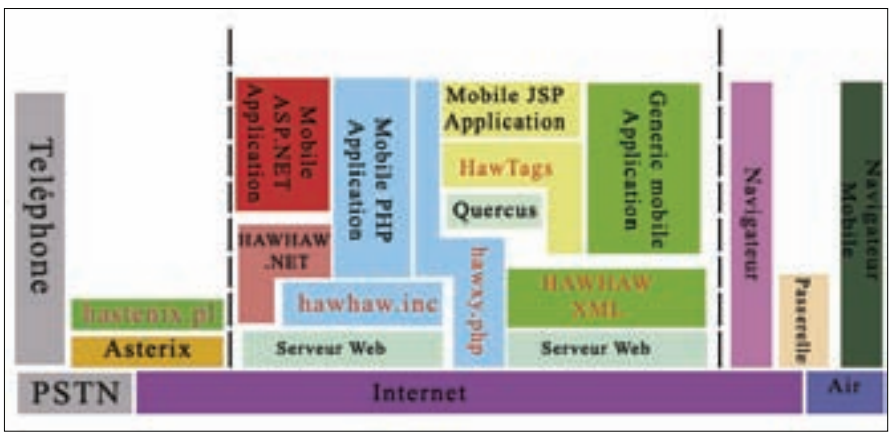



**Figure 1.** *Structure de HAWHAW*

## Environnement de travail

Votre système doit être à mesure d'exécuter du code PHP. Pour ma part, j'utilise XAMPP qui est un ensemble de logiciels offrant une bonne souplesse d'utilisation, réputé pour son installation simple et rapide.

Tous les exemples présentés ci-dessous ont été réalisés dans l'environnement suivant :

- *Windows XP,*
- *Eclipse PDT,*
- XAMPP,
- HAWHAW,
- *Firefox,*
- *Prophecy Voxeo.*

Nous verrons par la suite que HAWHAW n'exige pas d'installation ni de configuration particulière.

## Création d'une simple page

Téléchargez le fichier hawhaw.inc disponible à l'adresse suivante http://www.hawhaw.de/download/hawhaw.inc ; copiez ce fichier dans le dossier de votre application (serveur *web/mon application/hawhaw.inc*). Créez un fichier index.php dans le répertoire de votre application et ajoutez-y le code du Listing 1.

## Ajout des contrôles à la page

Nous pouvons visualiser notre page et constater qu'elle ne contient aucun contrôle. Nous allons modifier notre page index.php et y ajouter un formulaire et y ajouter un formulaire et des contrôles (Listing 2).

Comme vous pouvez le constater, grâce au fichier hawhaw.inc j'ai pu créer des objets de type `HAW_input`, `HAW_text`, `HAW_radio` et appelé des méthodes appropriées pour formater ces objets afin qu'ils soient accessibles par les différentes plate-formess. Chacun de ces champs ont une taille maximale définie. Il est plus élégant de valider la saisie de l'utilisateur quand le contrôle perd le focus, mais j'ai préféré mettre l'accent sur l'illustration des fonctionnalités qui sont offertes par ce toolkit plutôt que sur l'efficacité.

## Modification de l'apparence du simulateur

L'appel de la fonction `use_simulator()` permet d'obtenir un affichage identique à celui d'un appareil mobile dans un navigateur standard. Cependant, il est possible de changer l'apparence de ce simulateur en ajoutant une feuille de style comme paramètre à `use_simulator()` (Listing 3).

L'instruction `$page->use_simulator` (*http://www.testiphone.com/*) permet de simuler

**Listing 1.** *Création d'une simple page*

```
<?php

//inclusion de la classe 'hawhaw.inc' pour générer une sortie
require('hawhaw.inc');

//création d'une page
$page = new HAW_deck("Informations personnelles", HAW_ALIGN_CENTER);

//utilisation du simulateur par défaut
$page->use_simulator();

//création de la page
$page->create_page();
?>
```

**Listing 2.** *Formulaire de saisie de données*

```
<?php

//inclure la classe 'hawhaw.inc' pour générer une sortie
require('hawhaw.inc');
//création d'une page
$page = new HAW_deck("Informations personnelles", HAW_ALIGN_CENTER);
//utilisation du simulateur par défaut
$page->use_simulator();
//création d'un formulaire
$form = new HAW_form("index.php", HAW_METHOD_POST);

//création d'un bouton submit
$submit = new HAW_submit("Envoyer");
//création des champs de saisie
$input1 = new HAW_input("nom","","Nom : ");
//fixation de la taille du champs
$input1->set_size(8);
//fixation du nombre maximum de caractère
$input1->set_maxlength(8);
//ajout de 2 sauts de ligne après le champs
$input1->set_br(2);

$input2 = new HAW_input("prenom","","Prénom : ");
$input2->set_size(8);
$input2->set_maxlength(8);
$input2->set_br(2);
$text = new HAW_text(" TYPE DE CHAMBRE :");
$input3 = new HAW_radio("chambre");
$input3->add_button("Individuelle", "individuelle");
$input3->add_button("Double", "double");

$text1 = new HAW_text(" SERVICES & PRESTATIONS");
$cb1 = new HAW_checkbox("c1", "blanchisserie", "blanchisserie");
$cb2 = new HAW_checkbox("c2", "Jaccuzi", "jaccuzi");
$cb3 = new HAW_checkbox("c3", "banquet", "banquet");

//Ajout des champs et du bouton submit au formulaire
$form->add_input($input1);
$form->add_input($input2);
$form->add_text($text);
$form->add_radio($input3);
$form->add_text($text1);
$form->add_checkbox($cb1);
$form->add_checkbox($cb2);
$form->add_checkbox($cb3);
$form->add_submit($submit);
//Ajout du formulaire à la page
$page->add_form($form);
//création de la page
$page->create_page();
?>
```

**Listing 3.** *Modification de l'apparence du simulateur*

```
...

//création d'une page
$page = new HAW_deck("Informations personnelles", HAW_ALIGN_CENTER);

//utilisation du simulateur par défaut
$page->use_simulator("http://www.testiphone.com/");

...
```

**Listing 4.** *Affichage des résultats*

```
<?php
require('hawhaw.inc');
$page = new HAW_deck("Affichage des résultats", HAW_ALIGN_CENTER);
$page->use_simulator("http://www.testiphone.com/");
//initialisation des variables qui vont accueillir les valeurs en provenance
du formulaire
$nom = trim(addslashes($_REQUEST['nom']));
$prenom = trim(addslashes($_REQUEST['prenom']));
//contrôle des erreurs
if(!$nom || !$prenom)
{
$error_text1 = new HAW_text("Tous les champs doivent être renseignés.");
$error_text1->set_br(2);
$link = new HAW_link("Réessayez","index.php");
$page->add_text($error_text1);
$page->add_link($link);
$page->create_page();
 die;
}
if (strlen($nom)!=8) {
$error_text2 = new HAW_text("Le nom doit comporter au moins 8 caractères.");
$error_text2->set_br(2);
$link = new HAW_link("Réessayez!","index.php");
$page->add_text($error_text2);
$page->add_link($link);
$page->create_page();
die;
}
if (strlen($prenom)!=8) {
$error_text3 = new HAW_text("Le prénom doit comporter au moins 8
caractères.");
$error_text3->set_br(2);
$link = new HAW_link("Réessayez!","index.php");
$page->add_text($error_text3);
$page->add_link($link);
$page->create_page();
die;
}
// Affichage des résultats
$text1 = new HAW_text("Nom : $nom");
$text1->set_br(2);
$text2 = new HAW_text("Prénom : $prenom");
$text2->set_br(2);
$text4 = new HAW_text("Vous avez choisi une chambre " .$_REQUEST['chambre'])
;$text4->set_br(2);
$text3 = new HAW_text("Les services & les prestations choisies sont : " .$_
REQUEST['c1'] ." " .$_REQUEST['c2']." " .$_REQUEST['c3'] );
$text3->set_br(2);

$precedent = new HAW_link("Retour à la page précédente","index.php");
$page->add_text($text1);
$page->add_text($text2);
$page->add_text($text4);
$page->add_text($text3);
$page->add_link($précédent);
$page->create_page();
?>
```

**Listing 5.** *Confirmation réservation*

```
<noinput count="3">
<goto next="#transfer"/>
</noinput>

<form id="transfer">
<block>
<prompt>Veuillez patienter s'il vous plaît.  Nous vous transférons vers la
réception.</prompt>
</block>
<transfer name="tran1" bridge="true" connecttimeout="20s"
dest="tel:+12223334444">
 <filled>
  <if cond="tran1 == 'busy'">
  <prompt>La ligne est occupée. Veuillez Rappeler ultérieurement.</prompt>
  <exit/>
<elseif cond="tran1 == 'noanswer'"/>
<prompt>Aucun agent n'est disponible pour répondre à votre appel.  Veuillez
Rappeler ultérieurement.</prompt>
<exit/>
</if>
</filled>
</transfer>
</form>
```

**Listing 6.** *Capture du numéro du client*

```
<var name="ani" expr="session.
telephone.ani"/>

<catch event="connection.
disconnect.hangup">
  <var name="errType" expr="_
event"/>
   <submit next="error.php"
namelist="ani errType"/>
  <exit/>
</catch>
```

**Listing 7.** *Contenu du fichier error.php*

```
<?php

$ani = trim(addslashes($_
REQUEST['ani']));
$error = trim(addslashes($_
REQUEST['errType']));
$date = date('n-j-y,G:i:sa');

$message = "Une erreur de type:
$error,\n";
$message .= "est survenue,\n";
$message .= "Numero entrant:
$ani,\n";
$message .= "On $date.";

mail("name@example.com", "Erreur",
$message);

?>
```

l'affichage de notre page sur navigateur de *iPhone*.

## Affichage des résultats

Voyons maintenant comment récupérer les informations saisies dans le formulaire crée (Listing 4).

## Cas des navigateurs vocaux

Le cas des navigateurs vocaux est très particulier mais mérite d'être abordé. Je vais me baser sur *Voxeo Prophecy* pour élucider mes propos. La première étape est de télécharger *Prophecy* à l'adresse http://www.voxeo.com/prophecy/home.jsp et de l'installer bien entendu. Prophecy possède un serveur web intégré. Une fois installé, il faut lancer notre navigateur vocal en allant à *démarrer->Programmes->Voxeo->Prophecy System Tray*. Assurez-vous que tous les services Voxeo sont démarrés en faisant *Exécuter->services.msc*.

Dans ce volet, nous allons voir comment intégrer du *VoiceXML* à notre application. Nous allons créer notre application à la racine du serveur web intégré à Prophecy C:\Program *Files\Voxeo\webapps\www*. Un clic droit sur l'icone de *Prophecy* dans la barre des tâches puis cliquez sur *Prophecy Home* pour accéder à la page de configuration du navigateur (Figure 2).

Cliquez sur *Prophecy Commander*, le login et le mot de passe sont *admin* et *admin*.

**Listing 8.** *Modification du fichier hawhaw.inc*

```
...
//Propriété ajouté à la classe HAW_deck()
//pour définir la variable qui contiendra le chemin
//du document racine VoiceXML
var $application = "";
...
//La fonction "set_application()" permet d'associer le chemin du document racine //à la variable "application"
et de générer un fichier VoiceXML
//(par défaut hawhaw ne gère pas le VoiceXML).
function set_application($name)
{
if (!is_string($name))
die("argument invalide dans set_application()");
$this->application = $name;
}
...
//Modifications apportées à la sortie VoiceXML
//Ajout d'un nouvel attribut à la balise <vxml>
//lorsqu'une la valeur de la propriété attribut est définie

if ($this->application)
          $application = ' application="'.$this->application.'"';
         else
          $application = "";

                 if ($this->language)
          $language = sprintf(" xml:lang=\"%s\"", $this->language);
         else
          $language = "";

        printf("<vxml%s version=\"2.0\">\n", $application, $language);

        //Test de l'état du cache

        if ($this->disable_cache)
          echo "<meta http-equiv=\"Expires\" content=\"0\"/>\n";

             //Définition de la propriété voice

          while (list($key, $val) = each($this->voice_property))
          printf("<property name=\"%s\" value=\"%s\"/>\n", $val["name"],
             $val["value"]);

        echo "<form>\n";

        if (count($this->voice_navigator) > 0)
        {
             //Activation de la navigation des utilisateurs à travers
          //l'invite d'instructions

          echo "<var name=\"nav_counter\"/>\n";

          if (strlen($this->voice_navigator["repeat_dtmf"]) > 0)
            $dtmf = sprintf(" dtmf=\"%s\"",
            $this->voice_navigator["repeat_dtmf"]);
          else
            $dtmf = "";

          if ($this->voice_navigator["repeat_voice"])
            $voice_grammar = sprintf("<grammar>[%s]</grammar>",
            $this->voice_navigator["repeat_voice"]);
          else
            $voice_grammar = "";

          printf("<link%s event=\"repeat\">%s</link>\n", $dtmf, $voice_grammar);

          if (strlen($this->voice_navigator["forward_dtmf"]) > 0)
            $dtmf = sprintf(" dtmf=\"%s\"",
            $this->voice_navigator["forward_dtmf"]);
          else
            $dtmf = "";

          if ($this->voice_navigator["forward_voice"])
            $voice_grammar = sprintf("<grammar>[%s]</grammar>",
            $this->voice_navigator["forward_voice"]);
          else
            $voice_grammar = "";

          printf("<link%s event=\"forward\">%s</link>\n", $dtmf, $voice_grammar);
```

**Listing 8.** *Modification du fichier hawhaw.inc – suite*

```
        //Définition d'un lien et des capteurs du gestionnaire de liens

        echo "<catch event=\"repeat\"><goto expritem=\"'block' +
           nav _ counter\"/></catch>\n";
        echo "<catch event=\"forward\"><goto expritem=\"'block' +
           nav _ counter + 'end'\"/></catch>\n";
      }

      if ($this->voice _ text || $this->voice _ audio _ src)
      {
          //Création d'une sortie audio

        echo "<block><prompt>";

        HAW _ voice _ audio(HAW _ specchar($this->voice _ text, $this),
                      $this->voice _ audio _ src, HAW _ VOICE _ PAUSE, $this);

        echo "</prompt></block>\n";
      }
    }
  }
```

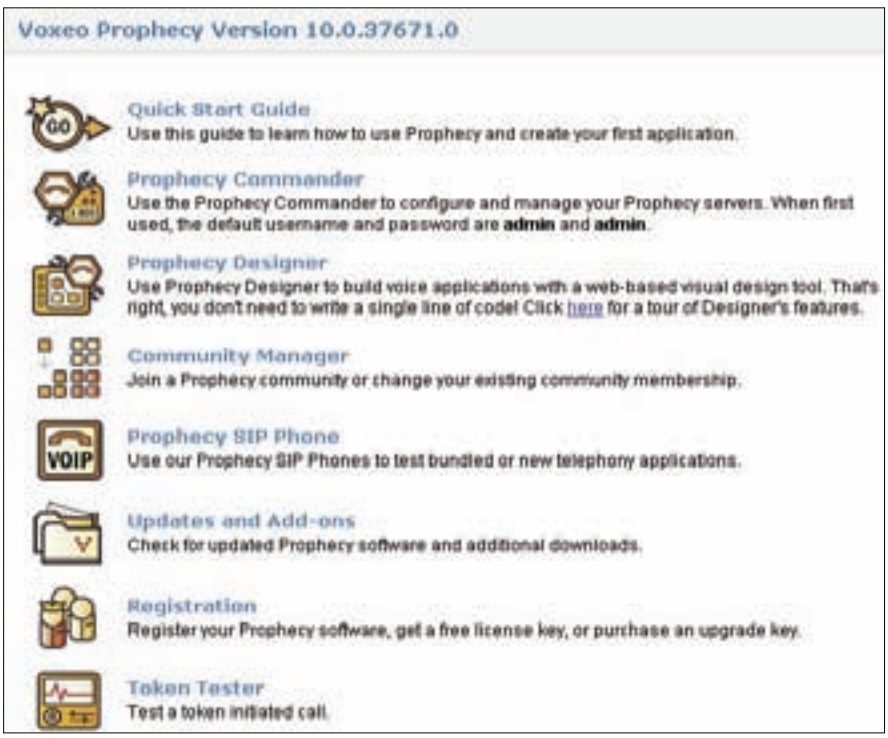



**Figure 2.** *Page d'administration de Phophecy*

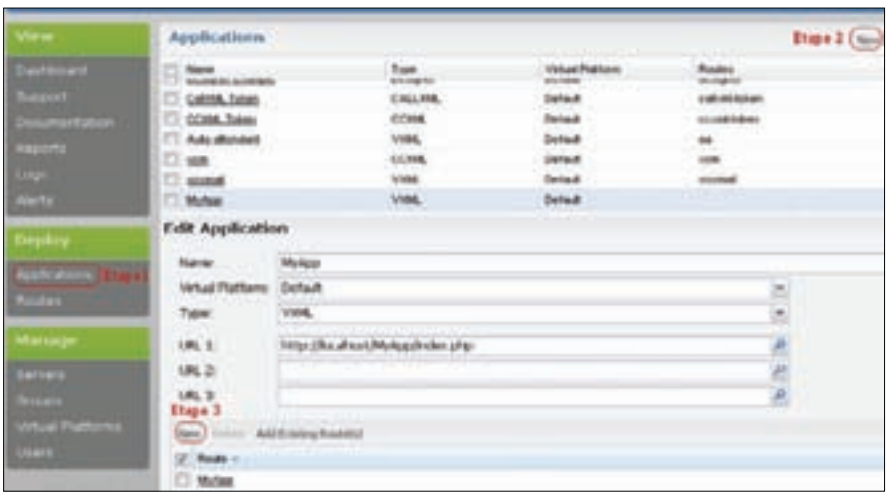



**Figure 3.** *Déploiement d'une application*

**Listing 9.** *Modification du formulaire de réservation*

```
require('hawhaw.inc');

$page = new HAW _
deck("Informations personnelles",
HAW _ ALIGN _ CENTER);
$page->use _ simulator();

//appel de la méthode set _
application intégré à la classe
HAW _ deck

$page->set _ application("appRoot.
vxml");
...

//activation de l'accessibilité
aux plate-forme de type VoiceXML

$input1->set _ voice _
text("Veuillez saisir votre
nom.");

//fixation du nombre maximum de
caractère

$input1->set _ voice _ type("digits?
length=8");
...
$input2->set _ voice _
text("Veuillez saisir votre
prénom.");

//fixation du nombre maximum de
caractère

$input2->set _ voice _
type("digits?length=8");
...
```

Il faut maintenant déployer notre application *Deploy -> Applications -> New* (Figure 3). Nous voulons transférer nos clients vers la réception de l'hôtel pour confirmer la réservation. Notre fichier *VoiceXML* contiendra les lignes de codes suivantes (Listing 5). Nous voulons maintenant capturer le numéro de téléphone du client (Listing 6). Ensuite nous apportons quelques modifications à notre fichier *hawhaw.inc* (Listing 8) afin de pouvoir intégrer du *VoiceXML*.

Le code du Listing 2 doit être modifier afin d'y intégrer du *VoiceXML* (Listing 9). La méthode `set_application()` a reçu en paramètre le fichier *appRoot.vxml*. Ainsi toutes les requêtes seront ré-orientées vers ce fichier . Voyons maintenant le contenu du dit fichier (Listing 10).

Pour voir le résultat, utilisez le *SIP Phone* intégré à *Prophecy* (*démarrer->Programmes->Voxeo->SIP Phone).*

## Gestion des sessions

La gestion des sessions par les appareils mobiles est un peu délicate. Les cookies ne sont parfois pas supportés par les plate-formes mobiles et les ID transitoires sont à considérer comme un problème de sécurité potentiel. C'est donc au programmeur de fixer les

**Listing 10.** *contenu du fichier approot.vxml*

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.0">

<!-- Capturer l'appel entrant -->
<var name="ani" expr="session.telephone.ani"/>

<!-- Activer l'aide grammaticale universelle -->
<property name="universals" value="help"/>

<!-- Gestionnaire d'événement de l'aide -->
<catch event="help">
        <prompt>Vous avez demandez de l'aide.</prompt>
        <reprompt/>
</catch>

<!-- Gestionnaire d'événement de  déconnexion -->
<catch event="connection.disconnect.hangup">
        <var name="errType" expr="_event"/>
        <submit next="error.php" namelist="ani errType"/>
        <exit/>
</catch>

<!-- Invite de commande vocal en cas d'erreur -->
<noinput count="1">
        <prompt>Désolé.  Les informations transmises sont erronées.</prompt>
        <reprompt/>
</noinput>

<noinput count="2">
        <prompt>Veuillez parler plus fort s'il vous plaît.</prompt>
        <reprompt/>
</noinput>

<noinput count="3">
<goto next="appRoot.vxml#transfer"/>
</noinput>

<nomatch count="1">
        <prompt>Veuillez répéter s'il vous plaît.</prompt>
        <reprompt/>
</nomatch>

<nomatch count="2">
        <prompt>Écoutez attentivement les instructions s'il vous plaît.</prompt>
        <reprompt/>
</nomatch>

<nomatch count="3">
<goto next="appRoot.vxml#transfer"/>
</nomatch>

<!-- Réception et transfert des appels entrants -->
<form id="main">
        <block>
                        <prompt>Bienvenu au service des renseignements.</prompt>
                        <goto next="index.php"/>
        </block>
</form>

<form id="transfer">
<block>
<prompt>Veuillez patienter.  Votre appel va etre transféré vers la réception.</prompt>
</block>

<transfer name="tran1" bridge="true" connecttimeout="20s" dest="tel:+12223334444">
        <filled>
    <if cond="tran1 == 'busy'">
      <prompt>La ligne est actuellement occupée. Veuillez rappeler ultérieurement</prompt>
      <exit/>
    <elseif cond="tran1 == 'noanswer'"/>
      <prompt>Aucun opérateur n'est actuellement disponible.  Veuillez rappeler ultérieurement.</prompt>
        <exit/>
                </if>
  </filled>
</transfer>

</form>

</vxml>
```

**Listing 11.** *Utilisation des sessions*

```
...
//création d'une page
$page = new HAW_deck("Informations personnelles", HAW_ALIGN_CENTER);
//activation des sessions
$page->enable_session();
...
//démarrage des sessions
start_session();
...
```

**Listing 12.** *Création d'un plugin*

```
function get_elementtype()
{
  return HAW_PLUGIN;
}

function create($deck)
{
  # code specifique au plugin ...
}
```

**Listing 13.** Intégration de Google AdSEnse

```
<?php
//intégration de la classe 'hawhaw.inc'
require("hawhaw.inc");
//intégration du plugin AdSense
require("HAW_plugin_AdSense4Mobile.php");

$page = new HAW_deck();
$contenu = new HAW_text("Placez le contenu mobile ici ...");
$page->add_text($contenu);

$adBlock = new HAW_plugin_AdSense4Mobile();
$page->add_plugin($adBlock);

$page->create_page();

?>
```

## Terminologie

- *WML* –Wireless Markup Language,
- *XHTML* – eXtensible HyperText Markup Language,
- *VoiceXML* – Voice eXtensible Markup Language ou langage de balisage extensible vocal,
- *WAP* – Wireless Application Protocol,
- *HDML* – Handed Device Markup Language,
- *URI* – Uniform Resource Identifier, soit littéralement identifiant uniforme de ressource.

indicateurs de contrôle du fichier *php.ini*. Pour faciliter la manipulation des sessions ; la classe `HAW_deck` offre une fonction appelée `enable_session()` (Listing 11).

La fonction `enable_session()` exécute quelques commandes importantes `ini_set(...)` pour modifier les paramètres des sessions durant l'exécution des scripts. Il est hautement recommandé d'utiliser au moins PHP 4.11 pour utiliser les sessions dans les applications basées sur HAWHAW.

## Création d'un plugin

HAWHAW fournit un mécanisme flexible qui permet d'ajouter des nouvelles fonctionnalités sans avoir à modifier la bibliothèque interne *hawhaw.inc*. Les plugins permettent d'étendre les fonctionnalités d'une application. Un *plugin* doit impérativement contenir une classe plugin dont le choix du nom est arbitraire. Cette classe contient deux fonctions obligatoires (Listing 12).

## Intégration d'un plugin

Pour illustrer mes propos, je vais prendre l'exemple de AdSense. AdSense est la régie publicitaire de Google utilisant les sites Web comme support pour ses annonces (source Wikipédia). HAWHAW dispose d'un plugin nommé AdSense4Mobile, téléchargeable à l'adresse suivante http://www.hawhaw.de/download/plugins/HAW_plugin_AdSense4Mobile.php.

Copiez et collez cet extrait de code dans un fichier et placez ce fichier dans le dossier de votre application. Créons notre page qui intégrera le *plugin* (Listing 13).

## Synthèse

HAWHAW permet de publier des pages WAP accessibles par des navigateurs standards et/ou vocaux. Il détermine automatiquement les spécificités des plate-formess clientes et crée un balisage approprié à celles-ci.

## Sur Internet

- *http://www.hawhaw.de/ref/php/html/index.html* – Documentation HAWHAW-PHP,
- *http://www.hawhaw.de/faq.htm* – FAQ/HowTo de HAWHAW,
- *http://www.w3.org/TR/voicexml20/* – Documentation VoiceXML.

**DONY CHIQUEL**

*Dony Chiquel est un ingénieur développeur spécialisé en technologies .NET et en conception objet avec UML. Il travaille depuis près de six ans dans le développement d'outils de gestion en tant que prestataire.*

# Rejoignez le Club .PRO

Pour plus de renseignement : *editor@phpsolmag.org*

## Stonfield Inworld

Stonfield Inworld propose aux entreprises des solutions globale d'intègration d'Internet et des Univers Virtuels dans leur stratégie de développement. Au-delà de ses services, la société consacre 30% de ses ressources à des travaux de R&D sur le e-Commerce et le e-Learning dans les Mondes Virtuels.

## COGNIX Systems

Conseil, conception et développement d'applications évoluées pour les systèmes d'informations Internet/intranet/extranet. Alliant les compétences d'une SSII et d'une Web Agency, Cognix Systems conçoit des applicatifs et portails web à l'ergonomie travaillée et des sites Internet à forte valeur ajoutée.
*http://www.cognix-systems.com*

## Anaska Formation

Anaska est le spécialiste des formations sur les technologies OpenSource. En partenariat avec MySQL AB, Mandriva, Zend et d'autres acteurs de la communauté, Anaska vous propose un catalogue de plus de 50 formations dédiés aux technologies du Libre.
*http://www.anaska.com*

## WEB82

Création et hébergements de sites web pour particuliers, associations, entreprises, e-commerce. Développement entierement aux normes W3C (www.w3.org) de sites web de qualité, au graphisme soigné et employant les dernieres technologies du web (PHP5, MySQL5, Ajax, XHTML, CSS2).
*http://www.web82.net*

## Core-Techs

Expert des solutions de gestion et de communication d'entreprise en Open Source, Core-Techs conçoit, integre, déploie et maintient des systemes de Gestion de Contenu Web, de Gestion Documentaire, de Gestion de la Relation Client (CRM), d'ecommerce et de travail ollaboratif.
*http://www.core-techs.fr*

## POP FACTORY

PoP Factory,SSII spécialisée Web. Développement de solutions applicatives spécifiques ; offre de solutions packagées : catalogue numérique, e-commerce, livre/magazine numérique, envoi SMS. Nous accompagnons nos clients tout au long de leur projet : audit, conseil, développement, suivi et gestion.
*http://www.popfactory.com / info@popfactory.fr*

## Blue Note Systems

Spécialistes en CRM Open Source, nous proposons une offre complète de prestations sur la solution SugarCRM. Notre valeur ajoutée réside dans une expertise réactive et une expérience des problématiques de la GRC. Nous vous aidons à tirer le meilleur parti de votre solution CRM.
*http://www.bluenote-systems.com*

## Intelligence Power

Conseil, Expertises, Formations et Projets E-business centrés au tour du cśur de métier : la Business Intelligence. Intelligence Power vous propose des solutions innovantes pour aligner la technologie sur la stratégie de votre entreprise.
*http://www.intelligencepower.com*

## Web Alliance

Vous souhaitez être en première page des moteurs de recherche ? Rejoignez-nous, 100% des clients Web Alliance sont en 1ère page de Google. Web Alliance, société de conseil spécialisée dans le référencement internet, vous propose son expertise (référencement, liens sponsorisés, web-marketing).
*www.web-alliance.fr*

Club .PRO

# Stratégies d'optimisation pour PHP 5

On entend tout et son contraire sur l'optimisation de scripts avec PHP : essayons de trier ce flot d'information et de ne conserver que les points pertinents.

**Cet article explique :**
- La place de l'optimisation dans la réalisation d'une application.
- En quoi de nombreuses recettes n'ont pas de sens.
- Quelques méthodes d'optimisation concrètes.

**Ce qu'il faut savoir :**
- Avoir une bonne vision des principaux mécanismes du langage.
- Quelques notions de génie logiciel et de conduite de projet.
- Éventuellement, avoir déjà programmé dans d'autres langages.

Niveau de difficulté


L'optimisation est un travail coûteux et important, pourtant cette tâche est souvent négligée ou incorrectement traitée. Dans le cadre de nombreux projets, l'optimisation est effectuée différemment selon le développeur, l'administrateur système ou l'intégrateur qui se concentrent chacun sur les points qu'ils jugent pertinents dans leur domaine de compétence. Pourtant un manque de communication entre eux rendra leurs tentatives généralement inefficaces. C'est une perte de temps et de moyens généralement causée par une mauvaise identification des zones critiques et qui doivent effectivement être optimisées, due à l'absence d'une stratégie d'optimisation mise en commun et étudiée en amont du développement.

### Optimiser le code

Quand on pense à l'optimisation, on pense avant tout au code. Dans le cas d'une application développée en PHP ou dans un autre langage, il est important de connaître les mécanismes mis en œuvre pour maitriser ceux sur lesquels se concentrer.

Avant tout, rappelons que PHP est un langage interprété, ce qui signifie qu'avant chaque exécution, le serveur doit lire et transformer le code en une série d'instructions que la machine peut comprendre, c'est le contraire de la compilation, où le code sera traduit une seule fois et transformé en un résultat binaire prêt à être exécuté. L'interprétation est plus gourmande, car elle nécessite un travail en amont, et souvent, la lecture de plusieurs fichiers avant l'exécution.

PHP est pourtant très rapide : le moteur interne Zend est optimisé pour que l'interprétation soit la plus efficace possible, et en contrepartie, apporte de nombreux mécanismes d'optimisation internes transparents pour le développeur. Les données sont souvent manipulées *juste à temps*, c'est-à-dire que PHP limitera tant que possible la lecture ou l'écriture de ressources afin que ces opérations soient effectuées seulement si nécessaires et quand elles sont nécessaires.

Contrairement aux cas des langages compilés, à l'échelle d'une série d'instructions, la meilleure optimisation n'est généralement pas celle directement effectuée par le développeur mais celle gérée en interne lors de l'interprétation.

L'optimisation du code passe donc moins par une réduction à minima des instructions que par une organisation plus fine de leurs effets. Concrètement, on veut dire par là que certaines opérations a priori très couteuses exécutées une fois seront peut-être plus intéressantes que certaines très rapides mais qui devront être répétées régulièrement. Les algorithmes étudiés devront donc être optimisés selon deux axes : leur coût spatial correspondant à l'usage qui sera fait de la mémoire, et leur coût temporel, c'est-à-dire le temps dont cet algorithme monopolisera le processeur pour être intégralement traité. Bien souvent, optimiser un axe pénalise l'autre : en évitant de stocker une donnée, il faut la recalculer... et vice-versa.

La meilleure des stratégies dépendra clairement de l'algorithme. La plupart du temps, il est plus avantageux de stocker un calcul dans la mémoire vive ou dans un fichier si elle est particulièrement volumineuse et réutilisée lors d'exécutions distinctes. À contrario, il est inutile de conserver une donnée atomique rapidement périmée, comme le nombre d'éléments d'une structure de données évoluant au cours de l'exécution.

La plupart du temps, la grosse majorité des opérations réalisées sont rapides, seules certaines seront particulièrement couteuses en temps ou en mémoire. Il est donc plus judicieux de passer du temps sur celles-ci, en omettant parfois d'en retravailler d'autres. Il existe des outils qui permettent de détecter concrètement ces opérations couteuses lors de l'exécution, nous présenterons plus tard l'un de ces outils. Par ailleurs, on peut déjà envisager que les appels à la base de données ou les manipulations de fichiers seront particulièrement couteux. N'oubliez pas que plus le volume de données manipulées est important, plus leur traitement nécessitera de ressources.

Enfin, la maîtrise de la qualité du code et de la validité des données sur l'ensemble de l'application permet également d'éviter de devoir effectuer des tests récurrents parfois inutiles. Concrètement, on trouve souvent au début de fonctions ou méthodes une série de tests sur la validité des données transmises en paramètres : ces opérations dont le temps d'exécution est négligeable permettent d'éviter des effets indésirables lors de l'exécution et de faire remonter des

erreurs nuisibles à la stabilité du programme. Malheureusement, en faisant la somme des ressources prises par ces tests, on réalise que le temps gaspillé devient très important.

On peut alors mettre en place une stratégie de programmation par contrat : plutôt que de réaliser ces tests dans la méthode, on informe les utilisateurs de la méthode des pré-requis qui valident les données, généralement par les commentaires. C'est alors au développeur utilisateur de la fonction (ou méthode) de savoir si la donnée est valide et de mettre le test en place s'il ne peut pas en avoir la certitude. On évite alors souvent des tests sur des données générées par l'application, donc nécessairement valides. Bien entendu, les contrats ne s'appliquent pas avec l'utilisateur final : toute donnée externe devant être vérifiée et nettoyée pour garantir la sécurité de l'application !

Par ailleurs, la mise en place de contrats ne s'applique pas à l'intérieur d'une classe pour les attributs protégés et privés : la classe doit être stable car l'utilisateur ne sera pas en mesure de vérifier des données auxquelles il n'a pas accès. On peut en conclure que le remplacement des tests préliminaires par des contrats ne s'appliquent qu'aux paramètres d'une fonction ou méthode.

Le Listing 1 montre la mise en place d'un contrat sur un exemple simple : une fonction divisant deux nombres. Pour garantir la stabilité du programme, le diviseur ne doit pas être nul. Dans le premier cas (`diviseAvecVerif()`), nous effectuons le test avant de réaliser l'opération pour pouvoir gérer l'erreur. La seconde fonction `diviseAvecContrat()` est plus rapide puisqu'elle n'effectue pas de comparaison avant le calcul. Lors de l'exécution de notre programme, deux cas se présentent. Dans le premier, la valeur du diviseur est connue et non nulle, le test est dont inutile. Dans le second, la valeur est générée aléatoirement, une valeur nulle est possible, le test est requis. Dans cet exemple, l'utilisation de la fonction `disiveAvecVerif()` aurait été plus coûteuse sur de nombreuses itérations.

**Listing 1.** *Mise en place d'un contrat sur une fonction*

```
/**
 * Divise $divise par $diviseur. Si $diviseur est nul, une exception
 * est levée.
 * @param float $divise
 * @param float $diviseur
 * @return float
 */
function diviseAvecVerif($divise, $diviseur)
{
  if($diviseur == 0.0)
    throw new LogicExcpetion('Division par zéro impossible !');
  return $divise/$diviseur;
}
/**
 * Divise $divise par $diviseur.
 * Contrat : $diviseur ne doit pas être nul
 * @param float $divise
 * @param float $diviseur
 * @return float
 */
function diviseAvecContrat($divise, $diviseur)
{
  return $divise/$diviseur;
}

// Utilisation de diviseAvecContrat
$valeur_connue = 10.0;$valeur_inconnue = rand(0, 10000)/100;
echo $valeur_inconnue.'/'.$valeur_connue.' = '.diviseAvecContrat($valeur_
inconnue, $valeur_connue);
if($valeur_inconnue == 0.0)
  echo 'Erreur : le diviseur est nul';
else
  echo $valeur_inconnue.'/'.$valeur_inconnue.' =
'.diviseAvecContrat($valeur_inconnue, $valeur_inconnue);
```

## Gérer l'environnement d'exécution

Au delà du code, il y a l'environnement dans lequel le code sera exécuté : le serveur. Son optimisation passe généralement par un choix judicieux des paquets installés et des démons actifs, utilisant également de la mémoire. Sans entrer dans les détails, on peut déjà penser aux extensions d'Apache ou de PHP qui ne seront pas utilisées, n'installer que les systèmes de gestion de bases de données nécessaires et par exemple, ne pas installer de serveur FTP si un démon SSH/SFTP est déjà en place. Tous ces choix dépendront du rôle du serveur et des besoins spécifiques aux applications qu'il hébergera, mais certaines recettes sont valables dans la plupart des cas.

Pour la plupart des niveaux applicatifs (serveur HTTP, interpréteur PHP, bases de données), il existe des outils qui permettent d'améliorer leurs performances : mise en cache des résultats de requêtes HTTP (grâce à un proxy par exemple), stockage du résultat d'interprétation par PHP ou du résultat d'une requête.

Au niveau de PHP, la manipulation des *opcodes*, le code interprété et prêt à être exécuté, est très courante dans un environnement de production. Ces *opcodes* sont souvent optimisés et mis en cache, pour éviter de réinterpréter le code PHP original avant chaque exécution. Vous trouverez une liste bien fournie d'outils permettant d'améliorer les performances de PHP sur le *Wikipédia* anglophone : *http://en.wikipedia.org/wiki/List_of_PHP_accelerators*, dont voici les principaux :

- *Zend Optimizer* : développé par l'entreprise des deux principaux développeurs de PHP, cet outil est considéré comme l'optimiseur le plus efficace du marché. Il est généralement inclu dans d'autres produits de l'entreprise et son support est garanti. En contrepartie, vous devrez débourser près de 500 $ pour obtenir une licence.
- *Xcache* : c'est un projet open-source originalement conçu pour être utilisé avec le serveur HTTP *Lighttp*, il reste néanmoins utilisable avec Apache et offre de bonnes performances. Plus récent que les autres, il a été conçu pour garantir sa compatibilité avec les nouvelles versions de PHP dès qu'elles sont disponibles.
- *MMCache for PHP* : réputé pour être l'une des meilleures alternatives open-source au *Zend Optimizer*, mais le projet risque cependant d'être moins maintenu depuis que le principal développeur travaille chez Zend !
- *eAccelerator* : dérivé de MMCache toujours maintenu.
- *APC* : *Alternative PHP Cache* est un optimiseur disponible sous la forme d'une extension PECL, qui offre par ailleurs des fonctions exploitables directement par le développeur pour choisir de mettre en cache certaines données lors de l'exécution.

Pensez à faire des tests en faisant varier différents paramètres, vous réaliserez souvent que les principaux ralentissements ne sont pas issus que de votre code : un environnement dont la configuration est maîtrisée est la base de la stabilité et de la rapidité du système et donc par extension, de votre application.

## Tâches planifiées et désynchronisation

De nombreuses tâches couteuses nécessitant le balayage d'un fichier, un parcours de la base de données ou des calculs intenses ne doivent pas nécessairement être exécutés en temps réel. Pour éviter de ralentir le système lors des pics d'utilisation, vous pouvez souvent reporter ces activités à des périodes où le serveur est moins sollicité. Un script PHP peut être exécuté en ligne de commande (CLI), vous pouvez donc différer certaines tâches en les planifiant avec *Cron*.

Vous pouvez par exemple lancer des tâches d'optimisation de la base de données, l'indexation de données ou la mise en cache de calculs répétitifs pendant la nuit ou les week-ends.

### Optimisations côté client

À l'échelle de l'affichage d'une page, la répartition des traitements entre le client et le serveur peut être gérée de plusieurs manières : certains calculs non critiques peuvent être réalisés sur le poste client avec Javascript pour alléger le travail du serveur.

Le résultat de l'exécution d'un script PHP est souvent une page web affichée dans un navigateur. Garantir un affichage rapide et fluide est important pour le confort de l'utilisateur. Les stratégies à adopter dans ce cas n'ont généralement pas grand chose à voir avec celles retenues côté serveur où on se préoccupe de l'efficacité de nombreuses exécutions simultanées. Ici, on cherchera donc la moindre ligne de code ou donnée qui ralentira l'ensemble.

Les extensions pour Mozilla Firefox *Yahoo! Slow* et *Google Page Speed*, à utiliser avec *Firebug* peuvent vous aider a détecter les points pouvant être améliorés grâce au calcul d'un indice de performance ou la réalisation d'un test de vitesse. Bien souvent, les tests que vous réaliserez avec ces extensions mettront en évidence que les principaux ralentissements ne sont ni issus du travail de PHP, ni de celui du navigateur (les plus récents étant très optimisés) mais des transferts sur le réseau et du temps de traitement de la requête HTTP.

Pour améliorer l'expérience de l'utilisateur, essayez de réduire au minimum le flot des données échangées sur le réseau et de limiter les échanges au nécessaire. Privilégiez le mécanisme des sessions à celui des cookies, réduisez le nombre de fichiers CSS et/ou javascript inclus, et optimisez le poids de toutes les données qui seront transmises au navigateur. Vous trouverez très facilement sur internet des outils de compression des documents HTML ou XML, des feuilles de styles CSS ou des fichiers javascripts, en java ou en PHP (qui peuvent donc être exécutées sur le serveur, sous la forme d'une tâche planifiée par exemple).

Dans le cadre d'une application interactive avec Ajax, favorisez tant que possible les calculs sur le poste du client plutôt qu'une requête, qui sera souvent plus longue. Quand l'interactivité n'est pas en jeu, il est souvent judicieux de grouper les requêtes, pour mettre à jour certaines données dans la page par exemple (flux RSS, actualités en temps réel, ...).

## Définir une stratégie d'optimisation

Une stratégie d'optimisation s'établit à l'échelle d'un projet. Elle sert à spécifier clairement les objectifs de performance et à partager entre les différents acteurs du projet une liste de choix techniques et de bonnes pratiques qu'ils devront respecter.

### Processus d'optimisation et cycle de développement

Le travail d'optimisation s'effectue à chaque étape du cycle de développement. Lors de l'étude des spécifications, il faudra déterminer des objectifs de performance pour les sections principales du projet. On réfléchit généralement à plusieurs axes parfois contradictoires : la rapidité d'exécution, les performances matérielles requises et les performances à forte charge. On utilise souvent le terme anglais *scalability* pour désigner ce dernier point. Il est particulièrement critique et souvent négligé du développeur qui s'attache principalement à la rapidité d'exécution : on cherche ici à s'assurer qu'une application fonctionnera toujours correctement avec de nombreux utilisateurs et un volume de données important. À chaque fois que la performance sera en question, il faudra pouvoir chiffrer le nombre d'utilisateurs et le volume des données manipulées, en se plaçant dans le cas moyen et le pire cas possible.

Pour choisir les technologies qui seront employées dans l'application, on réalise plusieurs prototypes qu'il faut tester en réalisant des *benchmarks* (*banc d'essais* en anglais) dans un environnement qui simule les cas définis dans la spécification. Ces tests permettent naturellement de départager les technologies qui sont à votre disposition, mais aussi d'identifier les manipulations qui seront lentes dans l'application.

Lors du développement, il faudra reproduire ces tests régulièrement pour pouvoir modifier certains éléments de la conception si les résultats obtenus ne respectent pas les spécifications. Pendant ces tests, vous pourrez détecter les goulots d'étranglement, qui sont des moments d'exécution particulièrement lents ou gourmands en ressources. Dans la plupart des cas, les principales (voire les seules) lenteurs de votre application seront causées par ces fameux goulots d'étranglement.

D'une manière générale, cette stratégie doit permettre d'offrir une optimisation pertinente et à un coût raisonnable. C'est en effet un poste de dépense important et difficile à chiffrer dans un projet, et c'est d'autant plus vrai que les retours sont difficiles à évaluer. C'est pourquoi il faut garder en tête que la meilleure optimisation n'est pas toujours maximale : passer du temps à concevoir des algorithmes fins et à retravailler les sections identifiées plus tôt permettent d'adopter une démarche efficiente, en garantissant une bonne stabilité globale de l'application et à un coût raisonnable. Soyez cependant vigilant aux fonctions plus petites, qui n'apparaissent généralement pas comme lentes pendant les tests, mais qui risquent d'être souvent sollicitées : la somme des ressources employées et du temps d'exécution de ces *petites fonctions* peut rapidement devenir plus conséquent que prévu !

Dans la stratégie que vous mettrez au point, n'oubliez pas de tenir compte de la configuration de l'environnement. C'est peut-être évident à priori, mais dans certains projets, une bonne gestion des serveurs qui supporteront l'application peut également faire économiser beaucoup de temps et d'argent. Pensez donc à envisager la mise en place des tests et l'évaluation des performances dans dans un environnement de test et un environnement fortement optimisé pour la production. Dans ce dernier, on supprimera les outils permettant de tracer et débugger le code qui sont particulièrement couteux au profit d'un système de cache et d'outils stables. Les systèmes de logs sont également très coûteux en ressources, notamment à cause de l'écriture sur le disque et de la redondance de l'action, limitez-les donc au strict nécessaire dans l'environnement de production.

Au passage, n'oubliez pas qu'à chaque version de PHP les développeurs nous offrent non seulement des nouvelles fonctionnalités, mais aussi de nouvelles optimisations qui améliorent très nettement les performances du langage. D'après le test de Pascal Martin (*http://blog.pascal-martin.fr/post/bench-php-5.2-vs-php-5.3-cpu*) sans système d'optimisation, PHP 5.3 est, près de 25% plus performant que PHP 5.2.9 !

### Premature optimization is the root of all evil

Cette célèbre citation de Donald Knuth s'applique, selon lui, à 97% des cas : l'optimisation prématurée coûte cher et donne souvent de très mauvais résultats. Dans le sens que je donne à cette phrase, il faut comprendre qu'avant d'optimiser, notre fameuse stratégie doit être clairement établie.

Nous allons essayer de faire le tri entre les bonnes pratiques à suivre et les moins bonnes dont on entend souvent parler, qui poussent souvent à trancher prématurément dans le code, et qui risquent de coûter cher plus tard !

Premièrement, nous allons briser quelques mythes répandus sur internet, qui s'attachent particulièrement à travailler certains détails du code pour épargner au processeur quelques instructions. Ces pratiques doivent plutôt être du fait du développeur, à condition que cela ne nuise pas à la qualité et la lisibilité du code.

Pour prendre un exemple concret, j'ai trouvé sur de nombreux sites et blogs des recettes de cuisines comme celles-ci :

- préférer `echo` plutôt que `print`,
- bannir `include_once`, `require_once`, et l'auto-chargement de classes,
- éviter les exceptions qui sont très lentes par rapport aux fonctions de gestion d'erreurs comme `trigger_error()`,

- ou même préférer systématiquement, dans vos classes, les méthodes statiques qui sont quatre fois plus rapides que les méthodes *normales*.

Ce sont de très mauvaises idées ! Ces optimisations prématurées et irréfléchies peuvent avoir des conséquences désastreuses sur le développement d'un produit volumineux.

N'oubliez pas que pour que le projet puisse être mené à terme, il est nécessaire de privilégier la qualité du code à des optimisations *sauvages*. Le code doit toujours rester clair, lisible et *auto-documenté* : on doit pouvoir comprendre ce qu'effectue le code en le lisant, grâce à des noms de fonctions explicites et un découpage qui élimine la présence de code ne correspondant pas directement au rôle d'un fonction dans son corps.

Le confort de développement ne doit pas être abandonné : il existe de nombreuses pratiques qui n'apportent qu'un gain de performance minime, surtout lorsqu'un système de cache comme APC est installé sur le serveur de production. La suppression des commentaires n'apporte pas de gain sensible, l'utilisation des méthodes magiques de PHP (`__get()`, `__set()`, `__call()` ou `__autoload()` par exemple) apportent un réel gain de lisibilité et peuvent être utilisées suffisamment intelligemment pour ne pas devenir des monstres gourmands. Vous pourrez trouver un exemple d'utilisation optimisé du mécanisme d'autochargement de classes dans cet article : *http://www.martiusweb.net/post/Chargement-automatique-de-classes-avanc%C3%A9-avec-PHP-5*.

Enfin, il faut toujours être très exigent au sujet de la souplesse de l'architecture de l'application : des classes que l'on peut manipuler facilement et réutiliser seront beaucoup plus économiques à moyen ou long terme que des optimisations hasardeuses.

Le découpage du code ne doit pas non plus être négligé : quelques inclusions sont parfois plus élégantes qu'une centaine de lignes interprétées mais jamais exécutées.

### La règle des 80-20

Cette règle bien connue signifie que 80% du travail est généralement effectué en 20% du temps. Elle s'applique particulièrement bien dans notre cas. L'essentiel des optimisations que vous pourrez apporter à une application seront basées sur des schémas connus et répétitifs, qui ne nécessitent pas une analyse profonde de l'application.

Par la suite, toutes les améliorations apportées risquent d'être couteuses pour le projet. Le retour sur investissement de ces tâches n'est pas toujours positif, et il est important d'évaluer l'impact de ces optimisations en terme de gain de performance : assurez vous que ces optimisations complexes seront effectivement bénéfiques et permettront de réaliser des économies supérieures aux dépenses engendrées par leur mise en place.

### Sur Internet

- *http://www.journaldunet.com/developpeur/tutoriel/php/040330-php-nexen-optimiser1.shtml* – Un article du journal du net faisant le point sur les principaux points techniques à aborder lors de la mise en place d'une stratégie d'optimisation avec PHP,
- *http://phplens.com/lens/php-book/optimizing-debugging-php.php* – Méthodologie d'optimisation de PHP (en anglais) : cet article synthétise la philosophie développée dans cet article, en donnant des exemples techniques concrets.

## Dans la pratique

Après ces longues lignes théoriques, nous allons voir des outils et méthodes qui permettront concrètement d'analyser votre application et de l'optimiser.

### Outils d'analyse du code

Nous allons nous intéresser à l'outil *Xdebug*, conçu pour être mis en place dans un environnement de test, il propose de nombreuses fonctionnalités intéressantes :

- une amélioration de l'affichage des fonctions `print_r()`, `var_dump()` et des erreurs en affichant la pile d'appels (l'ensemble des fonctions appelées de la racine du script à celle ayant déclenchée l'erreur, avec la ligne, le fichier, le nom de la fonction, les paramètres et l'utilisation de la mémoire),
- l'utilisation du protocole de débogage *DBGp* qui permet d'utiliser les points d'arrêts et l'exploration de l'exécution avec un IDE comme *Eclipse,*
- l'analyse du code et création d'un fichier de type *cachegrind*, permettant d'afficher visuellement l'exécution du code en fonction du temps avec un outil comme *KCacheGrind* ou *WinCacheGrind*,
- et enfin, le calcul de la couverture du code par l'exécution, c'est-à-dire, le nombre de lignes exécutées par rapport au nombre de lignes interprétées.

Je fais l'impasse sur les deux premiers points qui se concentrent plus sur le débogage que l'optimisation. Mais si vous ne connaissez pas les fonctionnalités de débogage interactif avec *DBGp,* n'hésitez pas à les tester : elles sont facilement mises en place (vous trouverez de nombreux articles à ce sujet) et sont tellement pratiques qu'on ne peut rapidement plus s'en passer !

L'installation de *Xdebug* peut s'effectuer simplement grâce à l'outil pecl de PHP, il faut néanmoins avoir un accès direct au serveur (il est donc impossible de l'utiliser avec un hébergeur mutualisé) : `pecl install xdebug`. La documentation de l'outil stipule de refuser la modification du fichier de configuration `php.ini` par pecl mais d'ajouter manuellement la directive `zend_extension="/usr/local/php/modules/xdebug.so"` (en modifiant éventuellement le chemin vers l'extension en fonction de votre système). Pour installer *Xdebug* sous windows ou par compilation manuelle, vous pouvez consulter la documentation de l'outil à cette adresse : *http://xdebug.org/docs/install*, par ailleurs, certaines distributions GNU/Linux proposent un paquet pour *Xdebug* dans leurs dépôts.

La configuration de *Xdebug* s'effectue également dans le fichier `php.ini`, pour bénéficier des outils de profilage de code, il faut ajouter les lignes du Listing 2. La directive `xdebug.profiler_enable=1` permet d'activer le profilage de code, les options suivantes sont optionnelles : `xdebug.profiler_output_dir` correspond au répertoire dans lequel les fichiers de *cachegrind* seront stockés et `xdebug.trace_output_name` le nom des fichiers générés ; ici `%t` sera remplacé par le timestamp unix et `%R` par l'URI de la requête. N'oubliez pas de relancer *Apache* après avoir configuré *Xdebug.*

Lorsqu'un script sera exécuté par le serveur, un fichier cachegrind sera généré, il pourra être lu avec un logiciel comme *WinCacheGrind* sous Windows, *MacCacheGrind* sous *MacOS* ou *KCacheGrind* sous Linux. La figure n°1 correspond à une capture d'écran de *KCacheGrind* sur un fichier généré par une exécution de *phpMyAdmin*.

L'outil vous permet de repérer facilement les zones les plus lentes grâce à un affichage sous la forme d'un diagramme. Vous pouvez naviguer dans les différentes zones du diagramme pour vous concentrer sur certaines parties de l'exécution. Dans l'exemple présenté (qui n'est pas nécessairement très révélateur), l'opération la plus couteuse est le lancement du module de sessions de PHP. Cela signifie qu'il peut être

**Listing 2.** *Configuration de Xdebug*

```
xdebug.profiler_enable=1
xdebug.profiler_output_dir=/tmp
xdebug.trace_output_name=cachegrind.out.%t.%R
```

**Listing 3.** *Lecture d'un fichier en une seule fois*

```
<?php
header('content-type: text/plain');

$filesize = filesize('imgtest.jpg');
$start = explode(' ', microtime());
for($it = 0; $it < 1; ++$it)
{
        $f[$it] = fopen('imgtest.jpg', 'rb');
        echo 'step : '.$it."\n";
        $img[$it] = fread($f[$it], $filesize);
        fclose($f[$it]);
        echo ' > '.memory_get_usage()."\n";
}

$end = explode(' ', microtime());
echo '>>> RESULT : '.(($end[0] + $end[1])-($start[0] + $start[1])).."\n";
```

**Listing 4.** *Lecture du fichier par tranches*

```
<?php
header('content-type: text/plain');
define('NB_ITERATIONS', 10);
$start = explode(' ', microtime());
for($i = 0; $i < NB_ITERATIONS; ++$i)
{
        echo 'step : '.$i."\n";
        $f[$i] = fopen('imgtest.jpg', 'rb');
        while(!feof($f[$i]))
        {
                $line[$i] = fread($f[$i], 1024);
        }
        echo ' > '.memory_get_usage()."\n";
        fclose($f[$i]);
}

$end = explode(' ', microtime());
echo '>>> RESULT : '.(($end[0] + $end[1])-($start[0] + $start[1])).."\n";
```

**Listing 5.** *Mise en cache du résultat d'une exécution d'un scripts*

```
$cached_file = md5($_SERVER['QUERY_STRING']).'.tmp';
if(is_readable('cache/'.$cached_file))
{
  // Le fichier existe dans le cache, on l'affiche sans
  // relancer l'exécution complète du script.
  $f = fopen('cache/'.$cached_file, 'r');
  while(!feof($f))
  {
    echo fread($f, 1024);
  }
  fclose($f);
}
else
{
  ob_start();
  // code PHP générant la page inclu
  include('affichage_page.php');
  $result = ob_get_flush();

  // enregistrement du résultat de l'exécution dans
  // le fichier de cache
  file_put_contents('cache/'.$cached_file, $result);

  // affichage du résultat de la requête
  echo $result;
}
```

**Listing 6.** *Utilisation de memcache*

```
<?php
$mcache = new Memcache;
$mcache->connect('localhost', 11211);
$complex = $mcache->get('complexComputedValue', MEMCACHE_COMPRESSED);
if(!$complex)
{
  $complex = computeComplex();
  // enregistrement de la valeur
  $mcache->add('complexComputedValue', $complex, MEMCACHE_COMPRESSED, 180);
}
$mcache->close();
?>
```

intéressant de trouver un moyen de gérer les sessions qui sera plus rapide, en les stockant dans la RAM plutôt que sur le disque par exemple.

N'oubliez pas que le résultat affiché ne correspond qu'à une seule exécution: une opération parfois un peu longue mais économe en mémoire est parfois plus intéressante quand elle est exécutée de nombreuses fois. *KCacheGrind* permet par ailleurs de cumuler l'analyse de plusieurs fichiers *CacheGrind*, vous pourrez ainsi tester le résultat sur plusieurs exécutions.

L'autre fonctionnalité intéressante apportée par *Xdebug* est l'analyse de la couverture du code. Vous pouvez encadrer le code à analyser avec les fonctions `xdebug_start_code_coverage();` et `xdebug_get_code_coverage();` qui retourneront un tableau contenant la liste des lignes effectivement exécutées. Ce tableau ne sera cependant pas très révélateur sans l'utilisation d'outils comme *PHPUnit* et *Phing*. Pour en savoir plus à ce sujet, je vous invite à consulter l'article de Stefan Priebsch sur *http://devzone.zend.com/article/2955-Creating-Code-Coverage-Statistics-with-xdebug*.

## Manipulation de fichiers

La manipulation de fichiers est particulièrement longue et couteuse : elle nécessite d'une part du temps pour lire le contenu sur le disque dur et une certaine quantité de mémoire vive pour analyser le fichier.

Nous pouvons adopter deux approches pour manipuler un fichier : le lire dans sa globalité ou morceau par morceau. La première approche est généralement plus rapide car un seul accès disque est effectué, mais la mémoire vive utilisée sera égale à la taille du fichier.

J'ai réalisé quelques tests sur une image de 8,2Mo : cette situation ne reflète pas nécessairement la réalité mais illustrera, je l'espère, la différence entre les deux approches. Le code effectuant la lecture du fichier en une étape est visible au Listing 3, le Listing 4 présente le code d'une lecture par tranches d'un ko (1024 octets). Afin de simuler plusieurs visites simultanées, chacune des lectures a été répétée plusieurs fois, et chaque variable a été indicée en fonction de l'itération pour ne pas vider la mémoire automatiquement : chacune des requêtes utilise une certaine quantité de mémoire.

Sur un serveur récent et rapide, pour 10 itérations, les résultats sont les suivants :

- la lecture complète a pris 0.02 secondes, l'utilisation de la mémoire vive a été multipliée à chaque itération, allant de 8,2 Mo à 81 Mo,
- la lecture par tranche a pris 0.08 seconde, la mémoire vive occupée a varié de 11 Ko à 123 Ko.

La lecture par tranches est légèrement plus lente, mais ne consomme presque pas de mémoire.

Sur 1000 itérations, le second script a pris 8 secondes, le premier n'a pas pu se terminer à cause d'un manque de mémoire vive : le serveur sera saturé et l'accès au site sera particulièrement difficile.

D'ailleurs, un autre test a montré que pour le second script, choisir des tranches de 10ko a permis d'obtenir un temps d'exécution de 0.025 seconde, et par tranche de 100ko 0.022 seconde : à partir d'une certaine taille, l'amélioration est donc négligeable.

### Persistance des données et des ressources

D'une requête à l'autre, l'utilisateur peut avoir besoin plusieurs fois d'une même donnée, et à chaque exécution, les objets nécessaires au traitement de la requête doivent être reconstruits. Afin d'éviter d'exécuter à nouveau certaines requêtes sur la base de données, on peut chercher à sauvegarder les données entre deux exécutions.

On peut alors penser à utiliser un système de fichiers temporaires ou utiliser le mécanisme de sessions de PHP. Les données pouvant être conservées doivent pouvoir être sérialisées, c'est-à-dire copiées sous forme binaire ou d'une chaine de caractères. Autrement dit, les liens entre les objets, les ressources ouvertes (fichiers, curseur vers la base de données, etc) ne peuvent être conservées.

Il est cependant possible de rendre certaines ressources persistantes, comme la connexion à la base de données lorsque celle-ci est fortement sollicitée. Ce sera en fonction de l'outil et de vos besoins que vous choisirez la persistance ou la recréation du lien entre PHP et la ressource.

### Mise en cache

Je pense que la notion de mise en cache n'a plus vraiment besoin d'être présentée ! Cette technique est mise en place dans tous les niveaux de l'informatique, du processeur au navigateur internet. Le principe est assez simple : quand l'obtention d'une donnée par calcul, transfert (sur le réseau par exemple) est plus longue que sa lecture dans la mémoire vive ou sur le disque dur, et si cette donnée risque d'être utilisée plusieurs fois avant d'être modifiée, alors on en garde une copie.

La mise en cache dans la mémoire vive concerne généralement le résultat de l'exécution d'un programme ou d'un algorithme, sur le disque dur, on enregistrera des données plus volumineuses sous la forme d'un ou plusieurs fichiers. On peut mettre en cache le résultat d'une requête à la base de données ou le résultat de l'exécution d'un script PHP complet (notamment quand la page ou le fichier n'est mis à jour que rarement). Cette dernière pratique peut être réalisée facilement grâce aux fonctions de PHP `ob_start()` et `ob_get_flush()` qui permettent de temporiser l'envoi de la réponse HTTP et de récupérer son contenu dans le code.

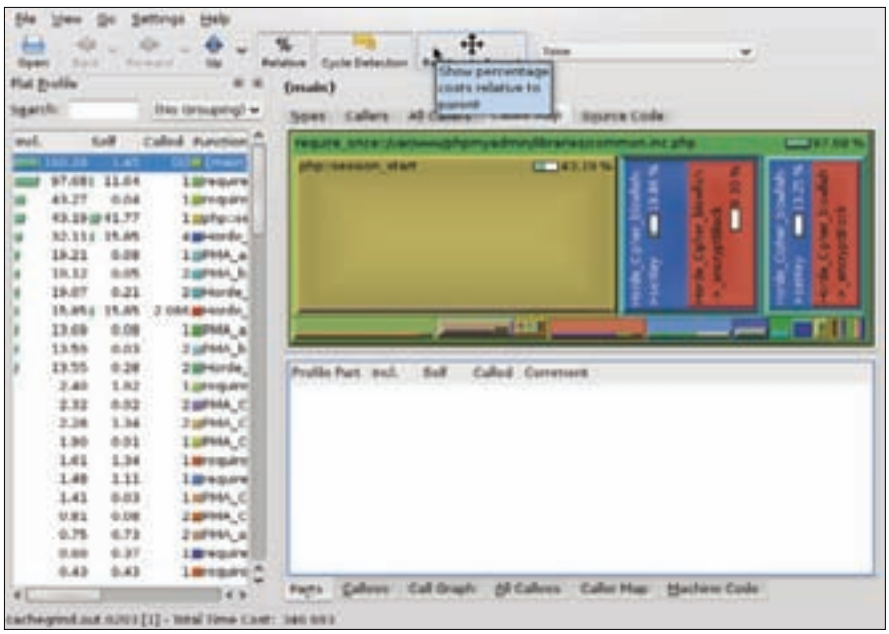


**Figure 1.** *Capture d'écran de KCacheGrind*

Pour mettre notre page en cache, nous allons copier son résultat dans un fichier dont le nom est généré à partir de la requête du visiteur. Pour les exécutions suivantes, nous vérifierons que ce fichier existe avant d'exécuter à nouveau le code. Lorsque la page subira une modification (dans une console d'administration par exemple), ce fichier temporaire sera effacé et recréé au prochain affichage de la page. Vous pouvez voir le code de cette stratégie de mise en cache dans le Listing 5.

Nous allons maintenant nous concentrer sur l'installation et l'utilisation d'outils qui permettent de mettre en cache le résultat de certaines opérations effectuées par PHP. Pour cela, nous utiliserons l'outil *memcache*, un démon manipulable dans plusieurs langages, dont PHP qui fournit une interface facile à utiliser. L'installation de *memcache* est indépendante de PHP, sous linux, vous pourrez probablement utiliser les dépôts de votre distribution. L'installation de son interface d'accès pour PHP passe à nouveau par pecl, avec la commande `pecl install memcache`.

Le module *memcache* est un moyen puissant de gérer les données manipulées d'une instance d'un script à un autre. Il rend l'architecture client-serveur du démon *memcache* aussi transparente que l'utilisation d'un serveur de bases de données. Le Listing 6 est un exemple de mise en cache d'une donnée calculée et sa récupération. Il est possible de transmettre au serveur n'importe quel type de donnée pouvant être sérialisée. La méthode `add()` de la classe admet ces paramètres :

- la clé de la donnée (une chaine de caractère),
- sa valeur,
- les drapeaux : ici *MEMCACHE_COMPRESSED* signifie que la donnée est compressée avec *zlib* (l'extension doit être installée), sinon sa valeur est 0,
- le dernier paramètre correspond à la durée de validité de la donnée, en secondes.

La méthode `get()` permet de récupérer une donnée mise en cache, elle retournera `false` en cas d'erreur ou si elle n'a pas été trouvée.

Vous pouvez choisir d'utiliser une connexion persistante avec la méthode `pconnect()` à la place de `connect()`, la méthode `close()` devient alors inutile. Vous pouvez également consulter le manuel de référence à l'adresse *http://www.php.net/manual/fr/book.memcache.php* pour en savoir plus sur le module *memcache* de PHP.

## Conclusion

Nous n'avons pas cherché à savoir quelle fonction de `print()` ou `echo()` est la plus rapide : cela n'a pas beaucoup d'importance si l'application est bien conçue et que l'optimisation est traitée intelligemment dans le cycle de développement d'une application. N'oubliez pas de faire des tests entre les différentes solutions que vous proposez pour résoudre un problème : l'optimisation est un processus long et compliqué, qui vous demandera de faire des compromis entre l'économie de temps et de mémoire, mais qui ne doit jamais être réalisée au détriment de la qualité du code et de la conception.

**MARTIN RICHARD**

*L'auteur est étudiant en informatique à l'INSA de Lyon et collabore à de nombreux projets d'applications pour le web ou de communication. Ses travaux et activités sont résumés sur son site internet www.martiusweb.net où il publie articles et cours sur ses découvertes.*

# Tester son code avec PHPUnit

La tâche de test est particulièrement répétitive. Vous devez tester si telle ou telle méthode fonctionne correctement. Et cela avec toutes vos méthodes. En tant que bon développeur, vous devez savoir que toute tâche répétitive peut être automatisée. C'est ce que nous allons faire avec nos tests.

**Cet article explique :**
- Pourquoi tester de manière automatisée une application.
- Comment tester de manière automatisée une application.

**Ce qu'il faut savoir :**
- Les bases du développement PHP.
- Installer PEAR.

Niveau de difficulté


Tant que votre application web reste d'une taille faible, il est assez simple de tester celle-ci manuellement. Mais dès que vous commencez à intégrer de multiples modules, tout changement peut devenir une horreur d'un point de vue de vos tests. Nous allons donc voir ensemble dans cet article pourquoi et comment tester automatiquement votre application web en PHP.

### Pourquoi et comment tester son code ?

Supposons une classe `bowling` qui va compter vos points. Tant que vous ne renversez aucune quille, vous ne gagnez aucun point. Notre test va donc initialiser la classe de gestion du *bowling*, effectuer une vingtaine de lancées en ne renversant aucune quille. Et vérifier que le score total est bien égal à 0. Pour l'instant, nous ne testons que cela et notre classe `bowling` est particulièrement simple.

Mais quand, par la suite, vous continuerez d'implémenter celle-ci, vous serez bien content de voir en une simple exécution de vos tests que le calcul de votre score fonctionne correctement.

**Listing 1.** *La classe bowling*

```
<?php
class Bowling {
    public function hit($pins) {

    }

    public function score() {
        return 0;
    }
}
?>
```

## Installer PHPUnit

PHPUnit est disponible sous la forme d'un package PEAR. Vous pouvez donc l'installer de manière particulièrement aisée en lançant une console et en tapant :

```
pear channel-discover pear.phpunit.de.
```

Afin d'ajouter le canal du framework à PEAR. Puis pour installer l'application en elle même :

```
pear install phpunit/PHPUnit .
```

### Écrire un test avec PHPUnit

Il existe, pour les applications développées en PHP, de nombreux frameworks de test. Dans cet article, nous verrons l'utilisation de *PHPUnit (phpunit.de)*. Restons sur notre exemple de classe simpliste de *bowling* (Listing 1).

Ecrivons maintenant le test dont nous avons parlé plus haut et qui ira vérifier que en lançant 20 fois la boule sans toucher aucune quille, nous aurons bien un score égal à 0 (Listing 2).

Nous exécuterons ce code en mode console afin de visualiser nos tests au fur et à mesure de leur exécution. Ouvrez donc votre console (commandes MS-DOS sous *Windows* ou *shell* sous *Linux/Mac*) et entrez la commande suivante : `phpunit MonTest` ou *MonTest* est le nom de votre document de tests.

**Listing 2.** *Les tests de la classe bowling*

```
<?php
# On inclut PHPUnit
require_once 'PHPUnit/Framework.php';
# Et notre classe de gestion du bowling.
require_once 'bowling.php';

class BowlingTest extends PHPUnit_Framework_TestCase {
    public function testScoreEqualsZeroIfNoHit() {
        $bowling = Bowling.new();

        # On lance 20 fois la boule
        for($i=0;$i<=20;$i++) {
            $bowling->hit(0);
        }

        # Et on vérifie que le score est bien égal à 0.
        $this->assertEquals(0, $bowling->score());
  }
}
?>
```

**Listing 3.** *La classe bowling avec calcul de la moyenne*

```
<?php

class Bowling {
    private $hits
    public function hit($pins) {
        $this->hits[] = $pins;
    }

    public function average() {
        return count($this->hits) / $this->score();
    }
}

?>
```

**Listing 4.** *Le test du calcul de la moyenne*

```
<?php

# On inclut PHPUnit
require_once 'PHPUnit/Framework.php';

# Et notre classe de gestion du bowling.
require_once 'bowling.php';

class BowlingTest extends PHPUnit_Framework_TestCase {
    public function testAverageScore() {
        $bowling = Bowling.new();

        # On lance 20 fois la boule
        for($i=0;$i<=20;$i++) {
            $bowling->hit(5);
        }

        # Et on vérifie que la moyenne est bien égale à 5
        $this->assertEquals(5, $bowling->average());
  }
}

?>
```

**Listing 5.** *Correction de la classe bowling avec gestion du score égal à 0*

```
<?php

class Bowling {
    private $hits
    public function hit($pins) {
        $this->hits[] = $pins;
    }

    public function average() {
        if ($this->score() == 0) return 0;
        return count($this->hits) / $this->score();
    }
}

?>
```

**Listing 6.** *Le test de calcul de la moyenne avec un score égal à 0*

```
<?php

# On inclut PHPUnit
require_once 'PHPUnit/Framework.php';

# Et notre classe de gestion du bowling.
require_once 'bowling.php';

class BowlingTest extends PHPUnit_Framework_TestCase {
    public function testAverageWithNoScore() {
        $bowling = Bowling.new();

        # Nous vérifions que le score est bien égal à zéro
        $this->assertEquals(0, $bowling->score);
        # Et que récupérer le score ne soulève pas d'exception
        $this->assertEquals(0, $bowling->average());
  }
}

?>
```

L'extension .php sera ajoutée automatiquement. Plutôt aisé non ?

Ici, notre application est particulièrement simple et de toute façon comme nous n'incrémentons pas le score lorsqu'il y a une boule de lancée, notre score n'augmentera jamais. Cependant, si vous commencez à calculer correctement votre score et faites des tests, par exemple en vérifiant que celui-ci est correct si vous ne faites que des strikes ; ou encore si vous n'en faites aucun et que vous ne renversez qu'une seule boule à chaque fois, vous devez vous assurer que votre calcul est correct.

Et vous n'aurez plus à revérifier toutes les exceptions possibles à chaque fois que vous améliorerez votre code.

Supposons par exemple que vous désiriez vérifier une méthode vous permettant d'obtenir la moyenne des points rapportés pour chaque *hit*. Tout d'abord, codons cette méthode de moyenne (Listing 3). Et testons ce code (Listing 4).

Notre code va ici inévitablement échouer puisque nous tentons de faire une division par zéro qui est mathématiquement impossible. Pour que notre test soit correctement exécuté, nous allons éviter les divisions par zéro. Mais nous allons également créer un test qui vérifiera que récupérer la moyenne des scores si le total de ceux-ci est égal à zéro ne soulève pas d'exception. Notre classe `bowling` modifiée est le Listing 5. Quant à nos tests il s'agit du Listing 6.

Nos tests passent correctement et nous avons évité un possible oubli futur du à une division par zéro le jour où nous mettrons notre classe à jour.

## Conclusion

Vous constaterez rapidement que écrire vos tests prends au moins autant de temps qu'écrire le code en lui même. Cependant, bien que faisant perdre du temps à chaque écriture ou modification d'une méthode, vos tests vous feront gagner énormément de temps lorsque votre application aura pris de l'envergure et que tester celle-ci manuellement entièrement deviendra particulièrement ardu.

Tester son application de manière automatique est vous fera donc gagner du temps sur le long terme. Mais vous permettra également d'augmenter la qualité de votre application et de pouvoir enfin dire *oui, je n'ai aucun bug* sans douter de vous.

**DAMIEN MATHIEU**

*Damien MATHIEU est développeur web salarié et la société O2Sources et auto entrepreneur spécialisé dans le développement d'applications Ruby on Rails.*

# Réaliser un injecteur de dépendances, en utilisant de bonnes pratiques logicielles

Un injecteur de dépendances ? Peut être cela ne vous parle-t-il pas vraiment. Tant mieux, l'objectif de cet article est d'éclaircir ces termes, et de présenter une implémentation que j'ai eu l'occasion de mettre en place, en m'appuyant sur de bonnes pratiques logicielles.

**Cet article explique :**

- Ce qu'est l'injection de dépendances, et l'inversion de contrôle.
- Quelques bonnes pratiques de développement logiciel.

**Ce qu'il faut savoir :**

- Utiliser les concepts orienté objet de PHP 5.

Niveau de difficulté


Dans cet article nous parlerons de l'injecteur de dépendances de Spiral, un framework maison réalisé avec quelques amis, et dont l'objectif principal est de découvrir les rouages des frameworks, ainsi que de nous initier aux bonnes pratiques logicielles. Alors que nous travaillions sur ce projet, notre principal but était de réellement comprendre, dans le détail, comment un injecteur de dépendances pouvait fonctionner. *Ré-inventer la roue*, pour mieux comprendre comment une roue fonctionne, en quelque sorte. Aussi, l'objectif de cet article n'est pas de fournir une documentation sur l'utilisation du composant, mais bien d'expliquer *comment* il à été réalisé.

L'injecteur de dépendances est disponible dans une version intégrée à Spiral ou dans une version *standalone*. Vous pouvez trouver le code sur le dépôt mercurial associé au projet. À l'heure ou j'écris ces lignes, l'injecteur de dépendances de spiral n'est pas encore terminé, mais est dans un état avancé, et devrait être disponible en fevrier 2010. L'ensemble des exemples de ce document sont en PHP, mais les concepts discutés ici peuvent être (et sont) implémentés dans d'autres langages.

## Comment gérons-nous nos objets ?

Avant toute chose, il est indispensable de bien comprendre ce qu'est l'inversion de contrôle. Lorsque nous réalisons des logiciels en utilisant le *paradigme* orienté objet, nous travaillons avec des classes, et la majeure partie du temps, nous faisons interagir ces classes entre elles. En pratique, certaines classes sont dépendantes d'autres classes.

Pour mettre un exemple derrière ces concepts, tout au long de ce document, nous allons nous mettre dans la peau d'*Alice*, une jeune fille qui adore manger des glaces, spécialement celles à la fraise ! Certains disent même qu'Alice est dépendante de la glace à la fraise, comme le souligne le Listing 1.

Il est clair, au regard de cette implémentation, qu'à chaque fois qu'Alice mange une glace, il s'agit d'une glace à la fraise. Génial, mais un jour, la mère d'Alice souhaite lui faire découvrir d'autres parfums...

En réalité, avec cette implémentation, il est impossible de changer la glace qu'Alice va manger.

**Listing 1.** *Une classe avec des dépendances.*

```
class Alice {

    public function mangerGlace(){
        $glace = new GlaceALaFraise();
        $glace->manger();
    }
}
```

## L'inversion de Contrôle (IoC)

Il apparait nécessaire de supprimer les dépendances entre nos deux classes, pour permettre à *Alice* de gouter de nouveaux parfums. Comment ? C'est assez simple, regardez donc le code :

```
class Alice {

    public function mangerGlace(Glace
$glace){
        $glace->manger();
    }
}
```

Quand Alice mange une glace (via la méthode `mangerGlace`), nous devons lui passer la glace, ce n'est plus elle qui choisit, *nous* le faisons à sa place. Ce principe est connu comme étant le principe d'Hollywood : *Ne nous appelez pas, nous vous appellerons*. En d'autres termes, n'utilisez pas l'opérateur `new` dans vos classes, mais préférez passer (ou qu'on vous passe) les objets par référence.

Alice peut faire d'autres choses avec sa glace, la laisser tomber par terre par exemple, grâce à la méthode `lacherGlace()`. Nous pouvons alors choisir de passer la glace à cette méthode également, ou choisir de la donner directement à Alice, la laissant s'occuper du reste, et évitant de lui passer une glace pour chaque action qui en nécessite une.

Le Listing 2 présente la classe `Alice`, faisant usage de l'inversion de contrôle. Il est bien plus facile maintenant de choisir la glace à donner

a *Alice*, et ainsi de contrôler les dépendances d'*Alice* vis à vis de la glace. Ici, il subsiste des dépendances dans le code. Il s'agit de dépendances vis à vis de contrats (interfaces) et non d'implémentations données (classes), puisque j'ai choisi d'utiliser le paradigme de programmation par contrats.Et c'est tout pour le principe d'inversion de contrôle ! Il s'agit *simplement* du fait d'inverser le flux de contrôle de vos application, en déléguant à un plus haut niveau la création des objets.

### Injection de dépendances

Maintenant que le concept d'inversion de contrôle est clair, expliquons ce qu'est l'injection de dépendances. Les deux concepts sont assez proches, et souvent utilisés de pair, mais il est important de bien saisir la frontière entre les deux.

Dans la méthode `mangerGlace`, nous considérons que la glace en question est déjà donnée à Alice. C'est un comportement vraiment utile : nous n'avons plus à nous occuper de la manière dont la glace est arrivée là, nous l'avons déjà (dans une propriété privée par exemple). Dans la section précédente, Alice était *dépendante* de sa glace. En inversant le flux de contrôle, le comportement d'Alice vis à vis des glaces est plus facilement contrôlable, et testable (utiliser des *mocks*, ou *bouchons de tests* est aussi facile que de régler une propriété, nous parlerons de tests plus tard).

Notre travail (celui de la mère d'Alice), est de créer les objets et de les passer à Alice. Les *injecter* est le bon mot. En utilisant des mutateurs, ou en utilisant le constructeur, injectant les objets nécessaires. Allons-y :

```
$alice = new Alice();
$glaceAuPaté = new GlaceAuPaté();
$alice->setGlace($glaceAuPaté);
```

L'inversion de contrôle est donc le fait d'exposer des méthodes publiques (ou des constructeurs) pour régler certaines propriétés, et l'injection de dépendances est le fait de, justement, injecter ces dépendances, utiliser ces méthodes et constructeurs.

### Un conteneur ?

L'exemple utilisé jusqu'ici est volontairement simple, et il à été choisi afin d'expliquer les concepts le plus clairement possible : *nous avons uniquement deux classes, et une dépendance*. En pratique, vous serez surement d'accord pour dire qu'un projet est rarement aussi simple. Aussi, dans les projets importants, la gestion du cycle de vie des objets et de l'injection de leurs propriété peut rapidement devenir un vrai casse tête. L'idéal est alors d'automatiser le processus de création, d'injection et de gestion de ces cycles de vie. Le conteneur fait exactement ça.

Pourquoi *conteneur* ? Parce que la création automatique et l'injection est effectuée grâce à un objet, qui se charge de contenir toutes les informations sur les dépendances. Une fois les objets créés, le conteneur garde une référence vers ces derniers au cas ou nous en aurions encore besoin (voir la définition de portée d'un service - les *scopes* - plus loin). Il s'agit d'une sorte de registre des objets, qui, afin d'alléger la tâche, ne les crée que lorsqu'il en à besoin, en se basant sur une définition des dépendances de l'ensemble des objets.

Le conteneur va donc se charger d'injecter les objets pour nous, en quelque sorte, il fait le travail de la Mère d'*Alice* à sa place. Nous souhaitons donc que lorsque nous appellerons *Alice*, via le conteneur, elle nous soit retournée avec une glace prête à être mangée !

```
$alice = $container-
>getService('Alice');
$alice->mangerGlace();
```

Ici, le conteneur à injecté la bonne glace à *Alice* (peu importe laquelle d'ailleurs, nous souhaitons juste avoir une glace). Si la glace elle même avait été dépendante d'autres objets (disons, des noix de coco par exemple), c'est le rôle du conteneur que de résoudre l'ensemble des dépendances, dans le bon ordre, simplifiant au maximum la tâche de gestion des dépendances entre les objets et les classes, laissant la tâche simple pour le développeur.

**Listing 2.** *Une classe qui utilise l'inversion de contrôle.*

```
class Alice {
    protected $_glace = null;

    public function setGlace(Glace $glace){
        $this->_glace = $glace;
    }

    public function mangerGlace(){
        $this->_glace->manger();
    }

    public function lacherGlace(){
        $this->_glace->lâcher();
    }
}
```

## Concepts logiciels

Maintenant que les concepts d'inversion de contrôle et d'injection de dépendances sont clairs (enfin, j'espère), nous pouvons commencer à parler de *comment* nous avons réalisé cette bibliothèque. Les concepts discutés ici sont des concepts assez simples, dont le principal objectif est de fournir une structure solide aux composants. Chaque composant a ainsi un rôle et un emplacement précis au sein de notre architecture.

### Le Schéma

Dans le schéma, et dans l'injecteur de dépendances en général, un service est un objet qui est géré par le conteneur. Le schéma représente les liens entre les différents services. Il décrit les dépendances de nos objets. Si vous connaissez le patron de conception de *fabrique abstraite*, vous pouvez vous représenter le schéma comme une configuration alors que le conteneur serait la fabrique elle même (ou quelque chose d'approchant).

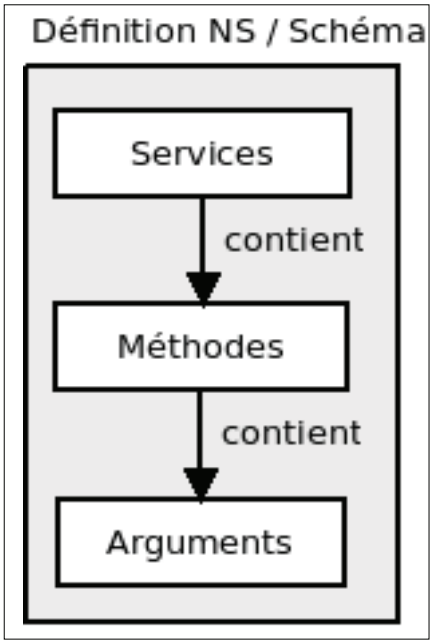



**Figure 1.** *Le schéma représente les liaisons de nos services*

Le schéma contient toutes les informations sur les méthodes qui doivent êtres appelées pour injecter les objets, le type des arguments qui doivent être passés, et tout autre type d'information potentiellement utile au moment de l'injection. Pour en revenir à notre exemple, le schéma contiendrait des informations sur le type de glace qui doit être passée à Alice (une glace à la fraise bien sur), et sur la manière de donner cette glace à Alice (via la méthode `setGlace()`). Jusqu'à maintenant, nous avons parlé de dépendances simples, mais le schéma peut aussi gérer d'autres types de services, méthodes et arguments. Tout est décrit dans les sections suivantes : `Services`, `Methods` et `Attibutes`.

**Tableau 1.** *Liste des types de services*

| Type | Description |
|---|---|
| Défaut | Un service *simple*, composé de méthodes, et qui peut être construit comment un simple objet |
| Alias | Un alias vers un autre service. Seul le nom est différent. Ce type de service permet de gérer facilement les dépendances dans le temps. «Pour le moment, il s'agit d'un alias, mais peut être qu'un jour nous aurons besoin d'un autre type de service» |
| Héritage de services | Plutôt que de se répéter maintes et maintes fois lors de la description de services qui se ressemblent, il est possible d'utiliser l'héritage. Cela ressemble grandement à l'héritage de classes : les méthodes que vous redéfinissez ou ajoutez dans les services enfants écraseront ceux des parents. |

### *Services*

Un service représente un objet. Dans notre exemple, la *Glace* et *Alice* sont des services. Un service se compose de :

- un nom,
- un ensemble de méthodes,
- une manière de se construire,
- une portée (scope).

La portée d'un service définit comment la durée de vie des services doit être gérée par le conteneur : Est-ce que le service doit rester dans le conteneur pendant toute la durée du script (singleton), ou doit il être systématiquement supprimé après avoir été construit (prototype) ? L'instance de l'objet courant peut être la même pour l'ensemble des services si la portée du service est définie comme étant un *singleton*, ou être à chaque fois différente si la portée est définie comme *prototype*.

D'autres types de portées peuvent êtres imaginées comme une portée de «session», qui retournerait la même instance durant une session unique, ou une sorte de portée «immortelle», qui retournerait toujours le même objet, en faisant persister cet objet à travers différentes sessions. L'injecteur de dépendances est fourni avec les types de service décrits dans le Tableau 1.

### *Méthodes*

Chaque service contient des méthodes. Une méthode permet d'injecter certains paramètres dans nos services, ou de définir certaines ressources qui doivent être appelées au moment de la construction des services. Dans le cas d'Alice, `setGlace()` est une méthode. Une méthode est composée d':

- un nom,
- optionellement, un nom de classe,
- une liste d'arguments,
- une information disant si la méthode est statique ou non.

Vous pouvez trouver les différents types de méthodes actuellement implémentées dans le Tableau 2.

### *Arguments*

Les méthodes contiennent donc des arguments, et il existe plusieurs types d'arguments également. Les arguments sont le bout de la chaine services / méthodes / arguments. L'ensemble des arguments sont décrits dans le Tableau 3.

## Stratégies de construction

Maintenant que nous avons un schéma qui représente les relations entre nos services, nous allons nous occuper de la construction de ces services. Nous avons choisi de séparer complètement les logiques de construction et de définition, pour permettre de favoriser un maximum d'usages possibles pour l'un et l'autre des composants.

Chaque type, dans le schéma, peut être lié à un type de stratégie pour se construire. Il y à donc plusieurs stratégies de construction pour les services, les méthodes et les arguments. L'intérêt d'utiliser des stratégies de construction est de permettre à chacun de nos types, dans le schéma, de se construire *eux mêmes*, en utilisant leur méthode `build()`, qui va elle déléguer la tache de construction aux stratégies. En interne, il est possible d'utiliser des stratégies de construction différentes, et d'en changer à tout moment. Ce comportement suit, en fait, le patron de conception stratégie.

**Tableau 2.** *Liste des types de méthodes*

| Type | Description |
|---|---|
| Défaut | Une simple méthode, avec des arguments. Peut être une méthode statique |
| Attributs | Utilisé pour régler directement les propriétés en utilisant les attributs publics de l'objet (`$service->attribut = $valeur`). Ce type de méthode peut contenir uniquement un argument. Il peut paraître étrange de gérer les attributs comme des méthodes. En réalité, il est important de comprendre la différence entre une méthode et un argument. Alors qu'un argument représente une valeur, une méthode représente une manière d'utiliser ces arguments. Dès lors, il parait plus logique de gérer les attributs comme des méthodes que comme des arguments. |
| Rappels (Callback) | Avant ou après la création de vos services, il est possible d'appeler des méthodes spécifiques, appelées méthodes de rappel. |

**Tableau 3.** *Liste des types d'arguments*

| Type | Description |
|---|---|
| Défaut | Types PHP natifs (int, string, float etc) |
| Conteneur | Il est possible d'injecter directement le conteneur. Ce type d'argument n'est utilisé que par les services qui nécessitent d'utiliser le conteneur. Ils sont appelées services «ContainerAware». |
| Service courant | Il est possible d'injecter le service en cours, et de l'utiliser comme argument. En pratique, ceci est uniquement utile pour les méthodes de rappel (callback) |
| Argument vide | Il s'agit d'un type d'argument qui n'a pas de valeur. L'argument «conteneur» et «service courant» étendent ce type. Attention, l'argument vide est différent de null. |
| Référence à un service | C'est un des types d'argument le plus utilisé, il représente un autre service. |
| Argument résolu grâce aux services | Parfois, il est utile d'utiliser un service pour récupérer un argument, je pense à la configuration entres autres. Ce type d'argument utilise donc une méthode spécifique d'un autre service pour être résolu. |

**Listing 3.** *Exemple d'utilisation des monteurs*

```
// utilisation du monteur XML pour construire le schema
$builder = new XmlBuilder();
$schema = $builder->build('schema.xml');
// et une fois le schéma construit, passons le au conteneur !
$container = new DefaultContainer($schema);
```

### Builders / Monteurs

Puisque nous parlons de patrons de conception (*design patterns*), parlons du motif *Monteur*. Vous serez sans doute d'accord avec moi pour dire qu'écrire un schéma entièrement à la main, en utilisant les classes dont nous avons parlé un peu plus haut peut s'avérer rapidement assez pénible. En tout cas, pour l'avoir expérimenté lors de l'écriture des tests, je peux dire qu'il ne s'agit pas d'un gain de temps, loin de là.

Une solution pratique consiste à utiliser le motif *Monteur*. L'idée est d'écrire le schéma sous une forme sympathique et facile à écrire pour nous, développeurs, et d'utiliser une classe intermédiaire pour transformer notre représentation du schéma dans la représentation compréhensible par notre composant. Cette classe intermédiaire *monte* donc notre schéma, en déchiffrant une autre structure.

Le premier type de *monteur* qui me vient à l'esprit (le plus pratique, en fait), est le *monteur* XML. Il est capable de lire un schéma, décrit au format XML, et de construire le schéma en utilisant les objets de notre bibliothèque. L'écriture du schéma XML à plusieurs avantages : il est facile à écrire, permet d'utiliser des outils extérieurs pour l'éditer facilement, et bénéficie, grâce a `XML Schema`, d'une auto-complétion et d'une vérification à la volée, lors de l'écriture.

Le Listing 3 montre un exemple d'utilisation du monteur XML. Le fichier XML n'est volontairement pas présent ici, pour des raisons de place. Il est possible de trouver un exemple d'utilisation en ligne, sur le dépôt de spiral. Les injecteurs de dépendances `Google Juice` et `Spring` permettent l'utilisation des annotations directement dans le code, pour définir les règles d'injection (le schéma pour nous).

Bien qu'il ne s'agisse pas d'un comportement recommandé (les annotations ne sont exploitables que par un type d'injecteur, même si `une spécification est actuellement en cours`), il est possible d'utiliser la réflexion sur un projet, et de la combiner a l'utilisation d'annotations pour déduire facilement la structure de notre schéma, pour le remplir ensuite à notre guise. Ce composant est également un *monteur*. Les monteurs suivants sont fournis de base :

- Le monteur XML.
- Le monteur PHP, qui utilise une interface fluide, pour permettre des configurations de ce type : `$monteur->addService()->withMethod()`.
- Le monteur Réflexion (utilise la réflexion sur nos classes pour construire un schéma).

### *Dumpers*

Un *dumper* est un objet qui copie des données d'un type de format vers un autre. Effectivement, il peut s'avérer utile d'avoir une manière simple de se représenter un schéma déjà défini.

Les dumpers permettent par exemple de représenter un schéma sous une forme graphique, ou bien sous une forme plus compréhensible pour nous, avec un simple texte par exemple. Il est donc vraiment facile de montrer les dépendances de vos projets, en utilisant simplement le dumper Dot (qui est le format utilisé par `graphviz`) par exemple. Voici la liste des dumpers :

- Le dumper texte.
- Le dumper Dot (graphviz).
- Le dumper XML.
- Le dumper PHP.

Ces composants laissent entrevoir des pistes intéressantes : il est ainsi possible d'écrire ses classes, puis de générer un schéma partiel grâce au monteur réflexion, de le *dumper* en XML, de le compléter à la main (avec de l'auto-complétion), et de le monter à nouveau, grâce au monteur XML.

### *Implémentation*

Voici quelques règles que nous avons suivi lors du développement en lui même.

### Espaces de noms / PHP 5.3

Alors que nous nous penchions sur ce projet, PHP 5.3 n'était pas encore sorti, mais puisque cette version apportait des fonctionnalités vraiment intéressantes (late static binding, espaces de noms et closures), nous avons choisi d'utiliser alors la version en cours de développement de PHP 5.3.

Maintenant, PHP 5.3 est disponible en version stable, et permet de faire fonctionner notre projet. Notre bibliothèque se sépare selon les espaces de noms suivants :

- L'espace de nom `Construction`, qui contient toutes les classes liées au concept de construction (les stratégies de construction).

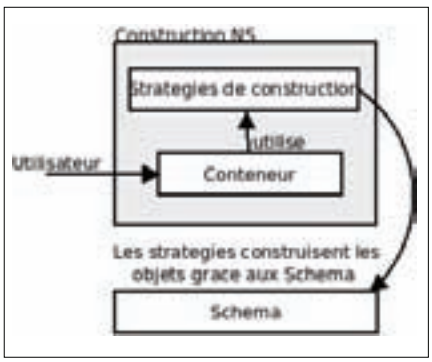



**Figure 2.** *Les stratégies de construction*

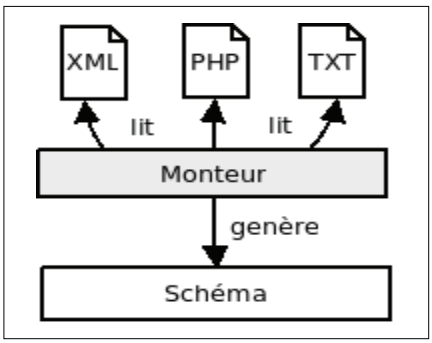



**Figure 3.** *Les monteurs*

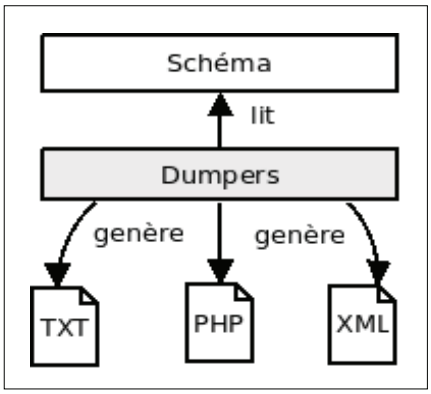



**Figure 3.** *Les dumpers*

- L'espace de nom `Definition` , qui contient le schéma.
- L'espace de nom `Transformation` qui contient les *Dumpers* et les *Monteurs*.

### Développement piloté par les tests (TDD)

Ce projet fut également l'occasion d'écrire nos premiers tests, pour finir par utiliser une approche pilotée par les tests. Le développement piloté par les tests préconise de réaliser ses tests avant d'écrire ses classes. Au début, ça chatouille un peu, mais on comprend rapidement l'intérêt de cette méthodologie, qui est une vraie bonne pratique.

Écrire ses tests avant d'avoir codé la classe nous oblige à la fois à privilégier une utilisation logique de nos composants, et à fixer les interfaces. Le code produit est réellement comme on souhaite l'utiliser, et non pas comme il est plus facile de l'implémenter. Écrire

**Listing 4.** *Les dépendances sont décrites dans un fichier «schema.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<services [...]>
    <service id="alice" class="Alice">
        <method name="setGlace">
            <argument ref="glace" />
        </method>
    </service>
    <service id="glace" class="GlaceAuPate">
        <constructor>
            <argument type="string" value="gelatine" />
        </constructor>
    </service></services>
```

**Listing 5.** *Les dépendances sont décrites dans un fichier XML*

```
interface Glace{
    function manger();
    function lacher();
}
class GlaceAuPate implements Glace{
    protected $_additif;
    public function __construct($additif){
        $this->_additif = $additif;
    }
    public function manger(){
        echo 'Et une glace à la '.$this->_additif.', une !';
    }
    public function lacher(){
        echo 'ooops!';
    }
}
// on récupère le schema grace au monteur XML
$dumper = new XmlDumper();
$schema = $dumper->dump('schema.xml');
$container = new DefaultContainer($schema);
/* puis on peut utiliser directement le conteneur, qui nous retourne des
service déjà configurés, et prêts à être utilisés ! */
$alice = $container->alice;
// ou $container->getService('alice');
/* à ce moment, l'injecteur de dépendances, à, tout seul, appelé la
méthode « setGlace » de l'objet $alice */
$alice->mangerGlace();
// retourne « Et une glace à la gélatine, une ! »
```

des tests, c'est aussi penser à l'ensemble des scénarios d'utilisation de ces classes, même les plus farfelus. Cela nous oblige à réfléchir à tous ces cas d'utilisation, et ça fait le plus grand bien !

Pour revenir aux tests, ils permettent de tester que notre application se comporte bien comme elle le devrait, mais cela permet aussi de détecter rapidement des régressions que de nouvelles fonctionnalités peuvent apporter. Rapidement, on écrit des tests pour tout : bugs, idées, etc. Ça favorise vraiment le développement d'une application. Un peu plus haut, je parlais de Mock objets (ou objets bouchon, en français). Je vous laisse consulter l'`article wikipédia sur les mocks` pour vous faire une idée plus précise, mais il s'agit, rapidement, d'objets qui permettent de simuler le comportement d'autres objets, ces derniers pouvant communiquer avec la suite de tests.

### Interfaces

Dans l'ensemble de nos classes, nous essayons d'utiliser des interfaces plutôt que des implémentations particulières. Pourquoi ? Parce que travailler avec des interfaces nous permet de changer à tout moment d'implémentation !

Dans le cas d'Alice, elle n'est pas dépendante d'un type particulier de glace (celle à la fraise), mais simplement aux glaces, à l'*interface* `Glace`, pour être exact. Chacune des interfaces ci dessous représente un comportement décrit plus haut :

- Schema.
- Service.
- Method.
- Argument.
- Container.
- Dumper.
- Builder.

### L'écriture des classes

Pour écrire nos classes, et parce que nous souhaitons fournir un système facilement extensible, nous fournissons quasi systématiquement une interface, et une classe abstraite, pour que chaque concept puisse être étendu facilement. D'ailleurs, l'écriture des classes en elle même est assez simple, une fois que tous les concepts ont étés décrit et sont clairs.

Vous pouvez regarder le code sur `le dépôt mercurial de spiral.` Je ne vois pas grand chose à ajouter à propos de l'implémentation, si ce n'est, peut être, qu'il est indispensable de commenter votre code : cela permet aux potentiels futur contributeurs de s'y retrouver facilement, et de comprendre le détail des opérations !

## Reprenons notre exemple

Et *Alice* dans l'histoire ? Le listing 4 fournit un fichier XML de description du schema de dépendances d'*Alice*, et le listing 5 décrit comment utiliser notre injecteur dans ce cas précis. Il faut bien sur garder à l'esprit que l'utilisation d'un injecteur de dépendances est surtout utile sur des projets d'envergure. Pour *Alice*, il semble bien plus simple d'injecter les dépendances *à la main*.

## Conclusion

J'espère que cet article vous aura intéressé, en tout cas j'ai pris beaucoup de plaisir à l'écrire, et vous aurez au moins appris comment nous avons choisi d'implémenter un injecteur de dépendances en utilisant quelques bonnes pratiques logicielles ! Si vous êtes intéressés pour discuter à ce propos, ou à propos d'autres concepts logiciels, vous pouvez me contacter, je serais ravi d'échanger avec vous.

### Sur Internet

- *http://hg.spiral-project.org/* – Le dépôt de spiral,
- *http://www.spiral-project.org* – Le site de spiral,
- *http://fr.wikipedia.org/wiki/Programmation_par_contrat* – La programmation par contrats.
- *http://fr.wikipedia.org/wiki/Strat%C3%A9gie_(patron_de_conception)* – Le patron de conception stratégie.

**ALEXIS MÉTAIREAU**

*L'auteur est actuellement en 4ème année au sein de l'ESI SUPINFO. Il travaille en alternance chez Makina Corpus, au sein du pôle django/python. Parallèlement, il publie de temps à autre des réflexions sur l'architecture logicielle, sur son blog personnel : http://www.notmyidea.org. Vous pouvez le contacter en utilisant l'adresse alexis@spiral-project.org.*


KERNIX
Web Application Provider
DÉVELOPPEMENT
D'APPLICATIONS WEB
SUR MESURE
Nous sommmes convaincus que la réussite de nos clients et de leurs projets web repose sur des solutions spécifiquement adaptées à leurs besoins. KerniX vous propose donc un service sur mesure : de la conception à l'hébergement.
E-COMMERCE PARIS 2009
Retrouvez-nous les 29, 30 septembre & 1er octobre 2009 sur le stand n°147 au SALON E-COMMERCE 2009
En savoir plus sur www.kernix.com

# Une belle boutique avec Joomla et VirtueMart

Vos produits sont les plus beaux et les moins chers ? Vous n'avez malgré tout que 10 secondes pour séduire votre futur client, et qu'il vous accorde sa confiance. Pour que votre boutique rencontre le succès qu'elle mérite, il est important de présenter vos produits avec un graphisme soigné et adapté à vos produits.

**Cet article explique :**

- Après une brève présentation de Joomla et VirtueMart, l'article vous explique comment personnaliser le graphisme de votre boutique en ligne en modifiant les thèmes.

**Ce qu'il faut savoir :**

- Un serveur où tester le site avec Joomla et VirtueMart, via des systèmes comme LAMP.
- Quelques connaissances de programmation en XHTML, ou CSS sont requises.
- Quelques rudiments de programmation en PHP.

Niveau de difficulté


Il existe plusieurs solutions Open Source de boutique en ligne. VirtueMart est une solution bien connue d'un large public. Ce logiciel pour fonctionner s'appuie sur un autre logiciel, le très populaire Joomla. Joomla est un CMS, acronyme de Content Managment System, qui se traduit par Système (ou logiciel) de gestion (ou administration) de contenu. Une des forces de Joomla est de pouvoir y ajouter des extensions qui vous permettent de multiplier les fonctionnalités de votre site. Plus de 3600 extensions sont proposées sur le site de *http://joomla.org*. Et ce nombre augmente de jour en jour. VirtueMart est l'extension qui vous permet de transformer votre site en boutique en ligne (*http://www.virtuemart.net*). Récemment plébiscitée comme la meilleure extension de Joomla, elle offre toutes les fonctionnalités classiques d'un site de e-commerce: organisation du catalogue, gestion du panier, gestion des clients, des commandes, et aussi de nombreux plugins qui en font un outil complet.

## Les différentes version de Joomla et VirtueMart

Si vous vous n'êtes pas familier avec Joomla, nous vous conseillons de tester une version prémodée de Joomla et VirtueMart, qui vous permet en quelques clics d'avoir une boutique en ligne installée. Le site *Joomla.fr* (onglet *Extensions*) vous propose une version prémodée complètement francisée, avec plusieurs autres extensions préinstallées. Cette version est disponible à l'adresse *http://www.joomlafrance.org/telecharger/fileinfo/Joomla_e-commerce_edition.html*. Le site de *VirtueMart* propose aussi une version prémodée, nommée *VirtueMart_1.1.2_eCommerce_Bundle_Joomla_1.5.9*. Elle contient Joomla, VirtueMart, des données d'exemples, un exemple au graphisme spécifique pour la boutique. Elle est en anglais, mais vous trouverez les fichiers pour la franciser sur le site de *http://www.Joomla.fr*. L'avantage de cette version est qu'elle propose un thème spécifique, sujet que nous vous proposons d'aborder dans cet article.

### Terminologie

- XHTML est un langage qui permet d'afficher des pages Web.
- CSS, acronyme de *Cascading Style Sheets* (en français feuilles de style en cascade) est un langage qui décrit la présentation des documents XHTML.
- PHP est un langage de programmation dynamique.

## Installer Joomla et VirtueMart

Pour tester votre boutique, vous devez disposer d'un serveur de type AMP (Apache, MySQL, PHP). Il en existe plusieurs: XAMP pour Windows, ou MAMP pour Mac. Décompressez l'archive téléchargée et copiez-le dans votre répertoire web appelé *htdocs*. Dans votre navigateur, tapez l'URL suivante: *http://localhost/mabelleboutique/*. L'installation de ce pack de e-commerce se fait en plusieurs étapes assez intuitives. Les 3 premières sont sans difficulté. L'étape 4 vous demande de fournir des informations concernant la base de données: *Nom du serveur*, *Nom d'utilisateur*, *Mot de passe*, *Nom de la base de données*. Ces paramètres dépendent de votre environnement. Poursuivez l'installation jusqu'à la fin, et comme demandé à la dernière étape, supprimez le dossier *Installation*. Pour accéder à l'administration de votre site, ajoutez */administrator* à la suite du nom de domaine pour obtenir la page d'authentification. (exemple: *http://localhost/mabelleboutique/administrator*). Entrez l'identifiant et le mot de passe fourni pendant l'installation.

## Template Joomla et thèmes VirtueMart

Les templates gèrent le graphisme de Joomla, les thèmes gèrent le graphisme de VirtueMart.

### Les templates de Joomla

Joomla sépare le fond de la forme. Le fond c'est votre texte. La forme c'est les couleurs, l'aspect graphique de la page, la disposition

des textes sur une/deux ou trois colonnes. La partie graphique de Joomla se gère par les templates. Il existe de nombreux templates Open Source disponible sur Internet. Ils s'installent facilement à partir de l'administration de Joomla en allant dans *Extensions* > *Installer/Desinstaller.* Le but de cet article n'est pas de vous expliquer comment personnaliser un template Joomla, mais il est cependant important de comprendre la différence entre le template Joomla et les thèmes VirtueMart.

Le template Joomla gère tout l'aspect graphique. La zone grisée de la Figure 1 est géré par le template Joomla. VirtueMart est un composant extérieur à Joomla. Son contenu s'affiche dans la zone blanche de la figure ... Le thème de VirtueMart est propre à l'affichage des données de VirtueMart, c'est-à-dire affichage d'un ensemble de produits, ou d'une fiche produit.

### Les thèmes VirtueMart

Un thème va définir l'apparence de votre boutique. Toutes les installations de VirtueMart viennent avec un thème nommé *default*. La version prémodée anglaise propose son propre thème intégré avec le template nommé *ja_larix*. Vous trouverez des thèmes Open Source et commerciaux dans la rubrique *Themes* du site *http://extensions.virtuemart.net.*

### Sur Internet

- *http://www.joomla.org/* – Site officiel de Joomla,
- *http://www.virtuemart.net* – Site officiel de VirtueMart,
- *http://www.joomla.fr* – Site de la communauté française de Joomla et de toutes les extensions Joomla,
- *http://www.afuj.fr* – Site de l'Association Française des Utilisateurs de Joomla.

## Créer son thème VirtueMart

Les thèmes vous permettent de personnaliser la présentation de vos produits, et de la commande de votre boutique en ligne.

### Création ou modification d'un nouveau thème

Dans VirtueMart, les thèmes sont placés dans le répertoire *components/com_virtuemart/themes/*. Chaque thème a son propre sous-répertoire. Pour installer un nouveau thème, transférez le dossier dans le répertoire *components/com_virtuemart/themes/*. Puis dans l'administration de VirtueMart, allez dans *Configuration générale* > *Configuration* , sélectionnez l'onglet *Site*, puis choisissez dans la liste déroulante *Choisir le thème de sa boutique* celui que vous désirez pour votre site. Si vous souhaitez modifier un thème existant, copiez le thème nommé *default*. Et donnez lui un nom approprié. Gardez le thème d'origine *default*, qui est utilisé par VirtueMart dans plusieurs cas. Il pourra en outre vous servir de référence si vous avez fait une erreur. L'arborescence du répertoire de votre thème doit être identique à celle du thème par défaut. Les fichiers que vous aurez dans un premier temps envie de modifier seront probablement les fichiers de mise en page des produits d'une catégorie, et de présentation d'une fiche produit.

### Connaître les fichiers appelés

Pour savoir quel fichier est appelé, dans administration de VirtueMart, allez dans *Configuration générale* > *Configuration,* Onglet *Général* et en bas de la page dans la zone *Paramètres principaux*, cochez la case *Debuggage ?*. Enregistrez votre modification. Allez sur votre boutique, des icônes I s'affichent, en les survolant les noms des fichiers appelés sont affichés.

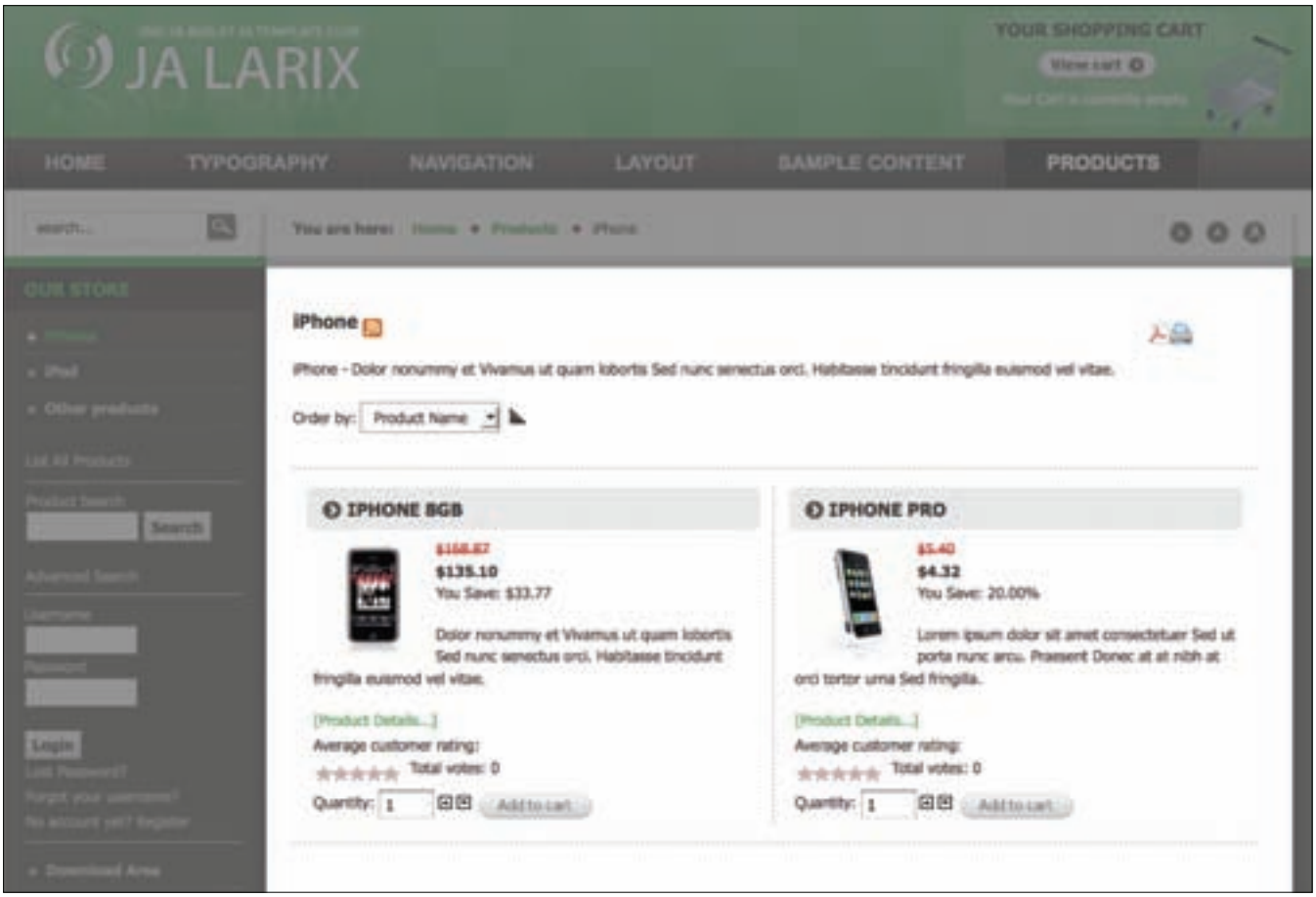



**Figure 1.** *Zones gérées par les templates et les thèmes*

### Présentation d'une fiche produit

Les fichiers concernant la mise en page des fiches produit sont dans le répertoire *product_details*. Vous pouvez créer plusieurs présentations de fiches produits, mais elles sont les mêmes pour tous les produits d'une catégorie. Les fichiers sont écrits en XHTML et CSS. Le code PHP est utilisé pour afficher les variables de façons dynamiques, il s'insère entre les balises <?*php et* ?>. La liste des variables disponibles pour votre fiche produit est présentée dans le Tableau 1.

### Présentation des produits d'une catégorie

Les fichiers pour la mise en page des produits d'une catégorie se trouvent dans le répertoire *browse*. Vous pouvez avoir des mises en pages différentes pour chaque catégorie. Elle est précisée dans l'administration de VirtueMart, lors de la création de la catégorie, ainsi que le nombre de produits à afficher par ligne.

Notez que dans le thème *default*, le modèle *browse_1* est le plus adapté pour la présentation d'un produit par ligne, *browse_2* pour 2 produits par ligne, et ainsi de suite. VirtueMart propose aussi dans la liste un modèle *managed*. Il ne correspond à aucun fichier. Dans ce cas VirtueMart va automatiquement choisir le fichier *browse* correspondant au nombre de produits par ligne.

### Votre panier

Le répertoire *basket* (panier) contient les fichiers utilisés pour la mise en page de votre panier. Il y a 4 fichiers, deux d'entre eux avec l'appellation B2B et les 2 autres avec appellation B2C. Dans le jargon du ecommerce, B2B et B2C correspondent aux acronymes anglais

**Tableau 1.** *Liste des variables disponibles pour la fiche produit*

| *Nom de la variable* | *Rôle* |
|---|---|
| $product_name | Nom du produit |
| $product_sku | Référence du produit |
| $product_s_desc | Résumé de la description du produit |
| $product_description | Description complète du produit |
| $product_weight_uom | Unité de mesure du poids |
| $product_length | Longueur du produit |
| $product_height | Hauteur du produit |
| $product_width | Largeur du produit |
| $product_lwh_uom | Unité de mesure du produit |
| $product_in_stock | Nombre de produits en stock |
| $product_available_date | Timestamp de la date de disponibilité |
| product_special | Si la case produit spécial a été cochée |
| $product_sales | Nombre de produits vendus |
| $product_unit | Unité du produit |
| $product_packaging | Emballage |
| $product_price_lbl | Libellé *Prix* |
| $product_price | Le prix du produit formaté |
| $product_packaging | Information sur l'emballage du produit |
| $file_list | La liste des fichiers supplémentaires de ce produit |
| $product_availability | Informations formatées sur la disponibilité des produits provenant du fichier */default/templates/common/availability.tpl.php /*; Inclut *le nombre de produits en stock* et *délai de livraison*; |
| $addtocart | Code du bouton Ajouter au panier |
| $product_type | Valeur des types de produits |
| $product_reviews | Liste des commentaires |
| $product_reviewform | Formulaire pour poster un commentaire |
| $product_image | Vignette de l'image, avec une URL vers l'image complète |
| $product_full_image | Chemin relatif vers l'image du produit |
| $product_thumb_image | Chemin relatif vers la vignette du produit |
| $buttons_header | Les boutons PDF, Email et Print |
| $navigation_pathway | Le fil d'ariane vers le produit (exemple: Catégorie / Sous-catégorie / le produit) |
| $more_images | Lien vers les Images du produit lorsque celui-ci en a plusieurs |
| $manufacturer_link | Lien vers la page d'information du fabricant |
| $vendor_link | Lien vers la page d'information du vendeur |
| $ask_seller | Un lien vers la page Poser une question sur ce produit |
| $related_products | Liste des produits associés |
| $navigation_childlist | Liste des sous-catégories de la catégorie actuelle |
| $images | Liste de toutes les images associées au produit |
| $files | La liste d'objets de tous les fichiers supplémentaires du produit |

**Tableau 2.** *Liste des variables définis pour la présentation des produits d'une catégorie*

| *Nom de la variable* | |
|---|---|
| $product_name | Nom du produit |
| $product_sku | Référence du produit |
| $product_s_desc | Résumé de la description du produit |
| $product_weight_uom | Unité de mesure du poids |
| $product_length | Longueur du produit |
| $product_height | Hauteur du produit |
| $product_width | Largeur du produit |
| $product_lwh_uom | Unité de mesure du produit |
| $product_in_stock | Nombre de produits en stocks |
| $product_flypage | URL vers la fiche produit |
| $product_available_date | Date de disponibilité formatée du produit |
| $product_availability | Nombre de produits en stock et date de disponibilité |
| $product_price | Le Prix du produit formaté |
| $product_availability | Informations formatées sur la disponibilité des produits provenant du fichier */default/templates/common/availability.tpl.php /*; Inclut *le nombre de produits en stock* et *délai de livraison*; |
| $form_addtocart | Code du bouton Ajouter au panier |
| $product_rating | Evaluation du produit |
| $product_details | La description complète du produit |
| $product_full_image | Chemin relatif vers l'image du produit |
| $product_thumb_image | Chemin relatif vers la vignette du produit |
| $images | Liste de toutes les images associées au produit |
| $files | La liste de tous les fichiers supplémentaires du produit |
| $navigation_childlist | Liste des sous-catégories de la catégorie actuelle |
| $navigation_pathway | Le fil d'ariane vers le produit (exemple: Categorie / Sous-catégorie / le produit) |
| $buttons_header | Les boutons PDF, Email et Print Buttons |
| $browsepage_header | Description associée à la catégorie |
| $parameter_form | Formulaire de recherche |
| $orderby_form | Formulaire de tri et navigation haut de page |
| $browsepage_footer | Navigation en pied de page |

Business to Business et Business to Customer, et indiquent le type de vente: entre deux commerçants (B2B) ou entre un commerçant et un consommateur (B2C). Dans VirtueMart, cette distinction se fait au niveau des taxes. Si le prix inclut les taxes, les fichiers B2C sont appelés, dans le cas contraire ce sont les fichiers B2B. Les fichiers ayant appellation *ro_* sont les fichiers appelés juste avant la validation de la commande. Ils ne contiennent pas les cases permettant de modifier la quantité / ajouter / supprimer un produit.

### La commande

Les fichiers qui affichent les différentes étapes de la commande sont dans le répertoire *checkout*. Il concerne les pages : demande de l'adresse de livraison, choix du transport, choix de la méthode de paiement.

**Tableau 3.** *Liste des fichiers du répertoire checkout*

| *Noms du fichier* | *Rôle* |
|---|---|
| checkout_bar.tpl.php | Affiche la barre de progression de la commande. |
| login_registration.tpl.php | Page de connexion et inscription |
| customer_info.tpl.php | Affiche les informations du clients |
| list_shipto_addresses.tpl.php | Liste les adresses d'expédition |
| list_shipping_methods.tpl.php | Liste les méthodes d'expédition |
| get_shipping_address.tpl.php | Affiche pour obtenir l'adresse d'expédition |
| get_shipping_method.tpl.php | Permet de sélectionner la méthode d'expédition |
| list_payment_methods.tpl.php | Liste les des méthodes de paiement |
| get_payment_method.tpl.php | Permet de sélectionner la méthode de paiement |
| get_final_confirmation.tpl.php | Affiche la confirmation finale |

### Les emails

Les fichiers mise en page des emails de confirmation de commande, et de questions sur un produit sont dans le répertoire *order_emails*.

## Conclusion

Dans cet article, nous vous avons fait découvrir les thèmes, et la possibilité de personnaliser le graphisme pour assurer le succès de votre boutique e-commerce. Nous espérons que cette première approche vous permettra de réaliser ce que votre imagination à créé.


## VALÉRIE ISAKSEN

*Valérie Isaksen est auteur du livre Joomla et VirtueMart, Réussir sa boutique en ligne paru aux Éditions Eyrolles. Elle est développeur free lance et intervient dans des missions de développement de sites internet et de ecommerce.*

# Les méthodes agiles

Les méthodes agiles proposent une alternative séduisante aux classiques méthodes de gestion de projet : basées sur une approche itérative et incrémentale, elles permettent une meilleure satisfaction du client.

**Cet article explique :**

- Scrum.
- Extreme Programming.
- Kanban.
- Lean.

**Ce qu'il faut savoir :**

- Notions de gestion de projet informatique.

Niveau de difficulté


Les méthodes de développement et de gestion de projet utilisées couramment dans l'informatique s'inspirent de l'ingénierie traditionnelle ; la plupart exploitent le modèle en cascade, soit un processus séquentiel dans lequel tout retour en arrière est impossible. Le projet se découpe en phases successives, et débute par le recueil des besoins du client, puis viennent la conception, l'implémentation, la recette puis la phase de maintenance. Ce modèle est évidemment nécessaire dans l'industrie ou la construction, domaines dans lesquels le moindre changement au cours du projet entraîne des coûts supplémentaires très importants, quand il n'est pas totalement impossible. Mais son adoption dans le domaine logiciel n'est pas sans poser nombre de problèmes.

De fait, un nombre très important de projets logiciels échouent purement et simplement. La dernière étude du *Standish Group* en la matière montre que 24% des projets de développement sont annulés en cours de route, et que 44% sont soit en retard, soit explosent le budget prévu, ou encore ne remplissent que partiellement leurs objectifs. Nous travaillons donc dans un domaine où seulement un tiers des projets connaissent le succès !

Bien sûr, il existe des équipes qui s'en sortent très bien en utilisant ce modèle en cascade ; prenons par exemple l'équipe de *Lockheed Martin* responsable du développement du logiciel animant les ordinateurs de la navette : les méthodes qu'ils utilisent sont extrêmement conventionnelles, un développeur ne peut changer la moindre ligne de code sans des spécifications décrivant le moindre détail du changement à opérer (les spécifications complètes du logiciel font d'ailleurs plus de 40000 pages aujourd'hui). Et pourtant, le logiciel qu'ils développent est proche de la perfection : en moyenne un seul *bug* par nouvelle version sur un total de 420000 lignes de code, et cela tout en respectant à la fois le budget et les délais. C'est en fait le contexte qui autorise (et impose) ce type de méthode : criticalité forte, besoins stables, grande équipe et une culture de l'ordre très prononcée chez le client.

Mais ce type de contexte est très rare dans notre domaine, où la plupart des projets sont exécutés par de petites équipes, dans un environnement chaotique et pour un client dont les besoins changent sans arrêt. Outre le fait que nos clients ont le plus souvent une conception assez vague du logiciel qu'ils souhaitent obtenir, cette conception va nécessairement évoluer au fil du temps, que ce soit à la suite de changements organisationnels ou concurrentiels, ou bien après avoir vu une première version du logiciel et s'être rendu compte qu'elle ne répondait pas à leurs attentes réelles. Les méthodes agiles sont nées de ce besoin d'être plus réactif aux demandes du client, afin de le satisfaire lui plutôt que les termes du contrat nous liant à lui. Dans ce but, elles prônent 4 valeurs essentielles :

- l'équipe : dans les méthodes agiles, la communication entre les personnes de l'équipe est bien plus importante que les procédures et outils mis en place ;
- l'application : la priorité numéro un est d'avoir un logiciel qui fonctionne ; tout le reste, et notamment la documentation technique, ne constituent pas des buts du projet. La documentation nécessite une charge de travail très importante, et très souvent, elle n'est pas lue au final et/ou pas mise à jour. Il est préférable de passer du temps à améliorer et commenter le code pour le rendre plus facilement compréhensible, et de s'assurer que l'ensemble de l'équipe dispose des compétences nécessaires à la maintenance et à l'évolution du code ;
- la collaboration : le client doit s'impliquer dans le projet, fournir un feedback permanent sur le logiciel, et répondre aux questions des développeurs sans attente ;
- l'acceptation du changement : en contrepartie de l'implication forte du client, l'équipe s'engage à permettre l'évolution des demandes du client à l'aide d'une planification et d'une structure du logiciel flexibles. Pour cela, les méthodes agiles prônent un développement itératif et incrémental, soit une succession de petits cycles de développement complets.

Cette notion de méthode agile et ces 4 valeurs qui la constitue ont été formalisées en 2001 par le manifeste agile (voir encadré) signé par 17 figures du développement logiciel, notamment *Ward Cunningham* (inventeur

du *Wiki), Kent Beck* (créateur de *Junit* et père de l'*Extreme Programming*), *Jeff Sutherland* (cofondateur de *Scrum*), *Alistair Cockburn*, *Martin Fowler, Dave Thomas*, etc... Les méthodes agiles existaient toutefois depuis la seconde moitié des années 90, les premières apparues étant notamment *DSDM, ASD* et *FDD*. Nous allons examiner en détail les 2 plus connues en France, *Scrum* et *Extreme Programming* (*XP*).

## Scrum

*Scrum* est une méthode de développement centrée sur l'organisation du projet, et non sur les aspects techniques. Elle est née des travaux de *Ken Schwaber* et *Jeff Sutherland* entre 1995 et 1996. *Scrum* est le mot anglais pour «mêlée» en rugby : être une équipe soudée est en effet fondamental en rugby pour faire avancer le ballon pendant une mêlée. Il y a 3 rôles principaux dans cette méthode :

- Le directeur de produit (*Product Owner*) représente le(s) client(s) et les utilisateurs. C'est lui qui choisit dans quel ordre les fonctionnalités doivent être développées en leur donnant un degré de priorité. Il est donc responsable de l'orientation globale du projet ;
- Le Scrum Master n'est pas un chef de projet au sens classique du terme : son rôle est de s'assurer que l'équipe travaille dans de bonnes conditions, et donc de résoudre tous les problèmes non techniques (administratifs, matériels, etc...).
- L'équipe n'a donc pas de véritable chef : elle s'auto-gère. Cela a pour effet de responsabiliser fortement les développeurs et de fait aboutit à un meilleur niveau de qualité.

### Product Backlog

Le recueil des besoins des utilisateurs débouche sur une liste de fonctionnalités à réaliser, appelée *Product Backlog*. Le plus souvent, les fonctionnalités y sont décrites sous la forme d'histoires orientées utilisateur (*User Stories*), comme dans *XP*. Le *Product Backlog* peut être maintenu dans un simple fichier de tableur, l'important est que les champs suivants soient présents :

- l'importance pour le client : définie par le directeur de produit, il s'agit d'une valeur arbitraire, qui sert simplement à définir l'ordre dans lequel les fonctionnalités seront implémentées. On peut par exemple s'accorder sur une valeur comprise entre 1 et 100 ;
- l'estimation initiale du travail nécessaire pour implémenter la fonctionnalité : l'idée ici n'est pas d'estimer le nombre de jours nécessaires pour chaque item, mais d'obtenir des estimations relatives entre les items. Il faut donc utiliser des points relatifs, sans unité, par exemple les premières valeurs de la suite de Fibonnaci (1, 2, 3, 5, 8, 13) ; un item noté 3 demandera donc 3 fois plus de travail qu'un item noté 1, sans que cela indique le nombre de jours réellement nécessaires.

A ces champs de base peuvent s'en ajouter d'autres, comme un ID, un champ pour les commentaires, etc...

### Sprint Planning

Scrum est bien sûr une méthode itérative : le développement s'effectue en une série de cycles de durée fixe (1 à 4 semaines en général) appelés Sprints. Au début de chaque sprint, une réunion de planification (Sprint planning) est organisée pendant laquelle vont être choisis les items du backlog à implémenter pendant le sprint. Cette sélection d'items (ou Sprint backlog) ne se fait bien sûr pas au hasard : normalement les items de plus haute importance sont choisis, mais le directeur de produit peut durant la réunion changer l'ordre des items si par exemple les priorités du client ont changé. D'autre part, la somme des estimations des items choisis doit rester inférieure à la vélocité de l'équipe : cette vélocité est tout simplement la somme des points des items réellement implémentés lors du sprint précédent. Naturellement, lors du premier sprint, il est nécessaire de définir une vélocité théorique de l'équipe ; cette vélocité sera révisée par la suite.

Cette réunion est importante à plus d'un titre : non seulement le *Sprint backlog* représente un engagement fort pour l'équipe, mais surtout cette réunion est une occasion de discuter des items et de poser aux divers intervenants les questions qui permettront d'affiner les estimations et de lever les doutes fonctionnels. En fonction du degré de complexité des différents items, cette réunion sert aussi à découper les items en tâches courtes (quelques heures, jamais plus de 2 jours).

Pendant le déroulement du sprint, chaque développeur s'affecte des tâches du *backlog* de sprint et les réalise. Il n'y a pas d'affectation des tâches par le *Scrum Master* : chaque équipier, une fois qu'il a terminé une tâche, s'affecte la tâche de plus haute importance suivante. Afin que tout le monde ait une vue globale du sprint, on utilise le plus souvent un tableau sur lequel les tâches sont accrochées, avec 3 colonnes : à faire, en cours et terminé.

### Daily Scrum

Tous les jours, de préférence en début de journée, l'équipe se réunit pendant une dizaine de minutes pour la mêlée quotidienne (*daily scrum*). Chaque membre de l'équipe doit répondre à tour de rôle aux 3 questions suivantes :

- Qu'est-ce que j'ai fait hier ?
- Qu'est-ce que je prévoie aujourd'hui ?
- Quelles difficultés ai-je rencontré ?

L'objectif de cette réunion est de s'assurer de la bonne synchronisation de l'équipe et de lever les problèmes éventuels afin de pouvoir les solutionner en tant qu'équipe. Elle ne doit pas dériver en discussion technique : celle-ci peut néanmoins se dérouler après, entre les développeurs directement concernés.

### Le manifeste agile

- Notre première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles.
- Le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.
- Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.
- Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet.
- Bâtissez le projet autour de personnes motivées. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.
- La méthode la plus efficace de transmettre l'information est une conversation en face à face.
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.
- Les processus agiles promeuvent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.
- Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité.
- La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle.
- Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent.
- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

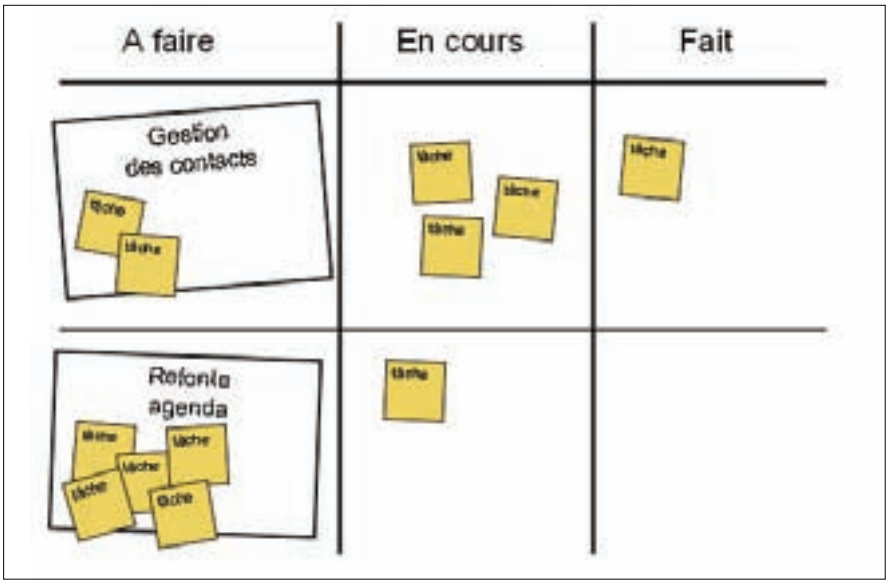



**Figure 1.** *Exemple de tableau des tâches*

Pierre : «*Hier j'ai terminé le formulaire de mise à jour des contacts. Je voudrais maintenant m'occuper de l'indexation Solr des contacts, mais je ne sais pas trop comment m'y prendre.*»
Paul : «*Cela tombe bien, j'ai fini de débugger l'agenda. Je peux te montrer aujourd'hui comment marche Solr.*»

### Graphiques d'avancement (burndown charts)

*Scrum* utilise un type particulier de graphiques pour visualiser l'avancement du travail et faire des projections sur le futur. Le graphique d'avancement appliqué au sprint représente le reste à faire au fil des jours, et selon la forme de la courbe, permet de dire rapidement si toutes les fonctionnalités choisies pourront être implémentées ou non durant le sprint. Si ce n'est pas le cas, il faut en parler avec le client pour voir si cela modifie à ses yeux l'ordre d'implémentation des fonctionnalités, ou si certaines ne peuvent pas être simplifiées. Si l'on constate des augmentations du reste à faire, il est important de comprendre pourquoi et de trouver un moyen d'y remédier : cela correspond en général à une ré-estimation à la hausse des items, qui peut par exemple être due à une mauvaise communication avec le client.

### Revue de sprint et rétrospective

À la fin de chaque sprint, l'équipe se réunit pour assister à une démonstration des fonctionnalités implémentées, que le directeur de produit doit valider. Il s'agit à la fois d'un moment de célébration et d'un moment de réflexion sur le travail accompli pendant le sprint, en le replaçant dans le contexte général du projet.

La rétrospective est, quant à elle, une réunion de fin de projet pendant laquelle les membres de l'équipe réfléchissent sur ce qui a bien et moins bien marché pendant le projet, et comment s'améliorer.

### Petite mise en garde

De nombreuses équipes adoptent *Scrum* parce qu'elle relativement facile à mettre en place et pas trop contraignante. Or, il est courant qu'au bout d'un moment le code se mette à ressembler à un plat de spaghetti et que la productivité de l'équipe en soit sérieusement affectée. C'est le syndrome de la dette technique, qui, comme toute dette, doit forcément être payée un jour, mais dont le paiement devrait être échelonné tout au long du projet. Le problème est que *Scrum* est centré sur les techniques de gestion de projet, et n'inclue pas de pratiques relatives au développement. Si vous souhaitez vous mettre à *Scrum*, pensez à introduire également des bonnes pratiques techniques, en vous inspirant par exemple de celles de *XP*.

## eXtreme Programming (XP)

Comme *Scrum*, *XP* est une méthode itérative : au contraire des méthodes en cascade, les équipes *XP* travaillent en parallèle sur les activités de conception, développement et test en des cycles très courts d'une semaine appelés itérations. Les membres de l'équipe travaillent sur des *stories* : de petites fonctionnalités, voire des bouts de fonctionnalités, décrites dans un langage simple et fonctionnel, et qui apportent une réelle valeur ajoutée au client une fois implémentées. Chaque semaine, l'équipe s'engage à implémenter un petit nombre de *stories*. Pendant la semaine, ils travaillent sur toutes les phases de développement pour chaque *story*, et à la fin de la semaine, ils déploient leur application, soit dans un environnement de test interne, soit directement chez le client. Cette approche permet tout d'abord d'obtenir des retours réguliers de la part du client, et ainsi de corriger le tir plus facilement si l'on se rend compte que l'application ne remplit pas parfaitement sa mission. Mais surtout, les équipes *XP* produisent chaque semaine une application potentiellement utilisable par le client : il n'y a donc pas à attendre des mois pour commencer à récolter les fruits de son investissement financier.

### L'équipe XP

Avec *XP*, l'*open space* est la règle : tout le monde s'assoit ensemble afin d'améliorer la communication entre les membres de l'équipe. Tout ceux qui ont travaillé en *open space* savent que cela peut vite devenir un endroit très bruyant et donc peu propice à un travail concentré : il est donc très important d'aménager cet espace, par exemple en prévoyant des boxes ou des salles dédiées aux appels téléphoniques. Comme avec *Scrum*, l'équipe s'auto-organise : il n'y a pas d'affectation prédéfinie des tâches, chacun planifie son propre travail.

Afin d'éviter le long travail de spécification des méthodes classiques, *XP* nécessite d'avoir des représentants du client sur site. Certains de ces représentants doivent être des experts métier : ce sont eux que les développeurs iront questionner pour définir les spécifications détaillées de chaque *story*. Ils doivent également mettre au point avec l'aide des testeurs des scénarios de tests fonctionnels. D'autres seront responsables de la définition du planning de livraison : ils identifient les *stories* à implémenter prioritairement, et font en sorte de les regrouper en ensembles cohérents afin de pouvoir livrer le plus souvent possible une application apportant de la valeur ajoutée. Dans bien des cas, les clients ne souhaitent pas déléguer plusieurs membres de leur personnel pour travailler à plein temps avec les développeurs. Dans ce cas, il est parfois nécessaire de faire l'inverse, et d'envoyer l'équipe travailler chez le client.

Comme dans *Scrum*, l'équipe comprend un directeur de produit dont le rôle est relativement similaire : il est le défenseur et le promoteur de la vision globale du produit, et à ce titre est le principal lien entre les investisseurs et l'équipe. Il guide le travail des représentants du client en définissant les priorités à un niveau plus élevé que ces derniers.

Les développeurs doivent tous mettre les mains dans le cambouis : bien sûr il est préférable d'avoir un ou plusieurs développeurs seniors, un développeur très qualifié en bases de données, ou encore une personne s'y

connaissant en architecture système, mais ils ne doivent pas se contenter de travailler dans leur domaine d'expertise. Chacun doit avoir une vision globale du code, et pouvoir intervenir sur celui-ci. Il est par contre préférable de disposer de testeurs se consacrant uniquement à l'écriture de tests fonctionnels automatisés et à l'exploration de l'application. Attention, être testeur nécessite de l'expérience et un savoir-faire qui s'apprend : il existe des techniques de test exploratoire qui doivent leur être familières. Enfin, un des développeurs doit être suffisamment expérimenté pour pouvoir épauler les autres concernant les pratiques techniques d'*XP*.

L'équipe doit également avoir un chef de projet, dont le rôle s'apparente à celui du *Scrum Master* : il s'occupe de tous les problèmes non techniques que peut rencontrer l'équipe. Nous allons maintenant passer en revue les différentes pratiques qui composent *XP*. Traditionnellement, elles sont au nombre de 13. J'en ai néanmoins ajouté d'autres, employées avec succès par certaines équipes et qui me semblent illustrer parfaitement le concept d'agilité.

### Itérations

Comme *Scrum*, *XP* est une méthode itérative, cependant le concept d'itération est ici poussé à l'extrême. En effet, une itération ne dure ici qu'une semaine : chaque semaine, l'équipe s'arrête de travailler, examine ce qu'elle a accompli et le montre au client, bref, compare la réalité au planning. Cette pratique permet de également de limiter l'un des risques majeurs de tout projet de développement : les 20% restants d'un projet finissent par prendre autant de temps que les premiers 80%. En effet, chaque retard de quelques heures dans l'accomplissement d'une tâche peut paraître négligeable en soi, mais la somme de tous les retards accumulés à la fin d'un projet est dévastateur. Le fait de travailler en itérations vous évite ce genre de mauvaises surprises. À la fin de la semaine, le travail cesse quelque soit le nombre de fonctionnalités implémentées. Cette limite dans le temps ne prévient pas les problèmes, mais les fait apparaître immédiatement, ce qui vous donne une chance de les résoudre.

Au début de chaque itération est organisée une réunion entre le directeur de produit et les développeurs pour planifier l'itération. En fonction de la vélocité de l'équipe lors de la précédente itération, un certain nombre de *stories* (voir plus bas) sont sélectionnées en fonction de leur priorité pour le client. Ceci fait, les programmeurs vont travailler sur les *stories* pour les découper en tâches techniques élémentaires du type :

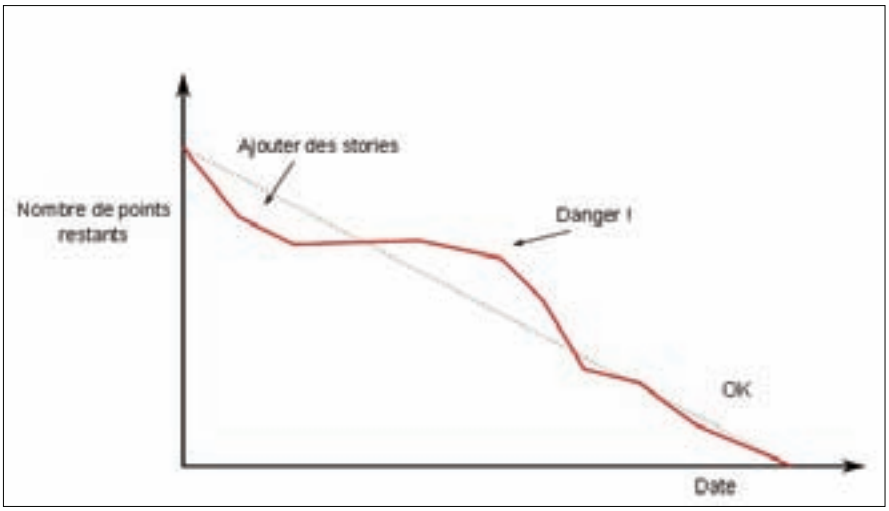



**Figure 2.** *Exemple de graphique d'avancement*

- Implémenter la logique métier,
- Créer la table associée,
- Créer une nouvelle classe de formulaire,
- Concevoir le gabarit HTML,
- etc...

Toutes ces tâches, dont la durée ne doit pas dépasser quelques heures, sont écrites sur des *post-its*, des fiches cartonnées ou tout autre support de dimension appropriée. Les développeurs peuvent bien sûr poser des questions aux représentants du client présents pour affiner les besoins de ce dernier. Puis commence le travail d'estimation des tâches, le but étant de continuer à discuter jusqu'à aboutir à un consensus. Une fois terminé, on vérifie que la somme des estimations ne dépasse pas ce qu'il est possible de faire pendant une itération, et si c'est le cas, on retire une *story*. Outre le fait que cette activité recouvre une bonne part de conception, elle permet aussi de donner confiance à l'équipe qu'elle pourra livrer ce qui a été promis à la fin de l'itération.

Il est important de toujours prévoir à la fin d'une itération une période de relâche. Celle-ci pourra servir en cas de problème à réussir quand même à livrer ce qui était prévu, et si tout se passe bien, elle pourra être mise à profit par les développeurs pour réduire la dette technique ou faire des recherches personnelles.

### User stories

Le but des *stories* est de représenter de façon simple les fonctionnalités à implémenter afin de servir de support aux futures discussions autour de leur planification. Elles ne sont donc ni des spécifications, ni des cas d'utilisation. Il s'agit juste de descriptions de quelques lignes formulées de telle façon que la valeur ajoutée pour le client est explicite, et que leur critère de complétion est clair. Le *Behavior-Driven Development* (BDD) propose pour cela un formalisme intéressant. Les *stories* doivent être écrites de la façon suivante :

En tant que `<rôle>` je veux pouvoir `<action>` afin que `<résultat>`.

Par exemple :

*En tant qu'administrateur, je veux pouvoir ajouter de nouveaux utilisateurs afin qu'ils puissent se connecter à l'application.*

Il faut donc absolument éviter d'utiliser un langage technique, de fournir des détails d'implémentation ou de décrire des résultats flous (du type *améliorer les performances*). Le support le plus efficace pour les *stories* est la fiche cartonnée, les *post-its* ayant tendance à s'envoler et à se perdre, mais c'est à vous de voir ce qui vous convient le mieux. L'avantage d'utiliser un support physique est qu'ainsi vous pouvez les placer sur un tableau blanc, ou sur une table de réunion pour la planification, ou la prendre avec vous pour en parler avec le client. Lors de la réunion de planification (voir plus bas), l'estimation retenue par les programmeurs sera notée directement sur la fiche.

Les *stories* ne remplacent pas les spécifications : vous aurez toujours besoin de détails pour leur implémentation, mais avec *XP*, la présence de représentants du client sur site vous permet d'obtenir ces détails directement de la bouche des intéressés.

### Le jeu du planning

Planifier un projet de développement est excessivement difficile. Le chef de projet qui s'efforce de s'acquitter de cette tâche ne dispose jamais d'informations suffisantes pour estimer correctement le travail nécessaire, et la plupart des plannings ne sont du coup jamais tenus. Pour pallier à ça, XP propose de regrouper clients et programmeurs pour travailler sur la planification. En effet, les clients savent ce qui servira le mieux leurs intérêts, quelles fonctionnalités ont le plus

de valeur ajoutée à leurs yeux. Quant aux programmeurs, ils sont les mieux placés pour estimer les coûts nécessaires à l'implémentation d'une fonctionnalité. Le fonctionnement est le suivant : quelqu'un choisit une *story* qui n'a pas encore été intégrée dans le planning ; les programmeurs estiment alors le temps nécessaire à son implémentation. Pour rendre cela plus ludique, ils utilisent souvent un jeu de cartes spécial (d'où le nom de *planning poker*) contenant les cartes suivantes : 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100. Chacun à leur tour, les développeurs vont montrer une carte correspondant à leur estimation. Les unités utilisées varient selon les équipes : cela peut être des jours/homme, des jours idéaux, ou des points, soit une unité arbitraire (cf *Scrum*). Bien souvent, les développeurs n'auront pas la même estimation, et c'est en fait le but du jeu : obtenir un consensus par la discussion. Les développeurs ayant donné les estimations les plus hautes et les plus faibles doivent justifier leur point de vue, ce qui entraîne une discussion. Les clients peuvent aussi être amenés à donner des éclaircissement sur la fonctionnalité souhaitée. À la fin d'un temps limite, un nouveau tour de cartes est effectué, et le processus se répète jusqu'à ce qu'on arrive à un consensus. En utilisant cette méthode, on lève beaucoup d'incertitudes et de risques et on aboutit nécessairement à des estimations beaucoup plus proches de la réalité. Enfin, une fois l'estimation faite, les clients place la *story* dans le planning en fonction de sa priorité à leurs yeux.

### Espace de travail informatif

L'ensemble des membres de l'équipe doit pouvoir être au courant en permanence de l'avancement du projet. Pour cela, l'information doit être disponible facilement, et l'espace de travail doit donner aux membres de l'équipe des moyens de communiquer : des tableaux blancs doivent donc être disposés tout autour de l'équipe. Certains seront utilisés pour dessiner des graphiques montrant l'avancement du travail, comme par exemple le nombre de *stories* restant à implémenter pendant l'itération, ou les prochaines date importantes, d'autres seront réquisitionnés par les membres de l'équipe lorsqu'ils souhaitent débattre d'un problème à plusieurs.

### Programmation en binôme

Beaucoup de dirigeants considèrent le travail en binôme comme un gâchis de ressource, pourtant les équipes qui le pratiquent ont bien souvent une productivité supérieure, et surtout produisent un code de meilleure qualité. La raison en est que pendant que l'un code, l'autre peut penser librement : celui qui tient le clavier (le pilote) se concentre sur l'écriture d'un code rigoureux et propre, pendant que l'autre (le navigateur) peut réfléchir aux tâches à suivre et à la façon dont le code en cours d'écriture va s'intégrer avec le reste. De plus, vous serez nettement moins sensible aux interruptions : vous constaterez que vos collègues sont naturellement moins enclins à interrompre 2 personnes qui travaillent ensemble qu'une seule, et surtout, le navigateur peut prendre en charge l'interruption tandis que le pilote continue à travailler et ne perd pas sa concentration. Travailler en binôme nécessite également d'échanger fréquemment les rôles : cela permet au pilote de se reposer, et ainsi de rester productif.

Il n'est pas forcément utile de travailler ainsi tout le temps : certaines tâches simples peuvent tout à fait s'effectuer seul, mais la majorité du code devrait néanmoins être écrite en binôme. Les binômes ne doivent en aucun cas être assignés par quelqu'un : c'est à chaque développeur quand il commence une tâche de demander à un autre de l'assister. Il est cependant important de pousser les gens à changer souvent de binôme si cela ne se fait pas naturellement. Cela améliorera la cohésion de l'équipe et permettra à chacun de travailler sur l'ensemble du code.

Il n'est pas toujours facile de tenir le rôle du navigateur au début : on a souvent tendance à vouloir s'emparer du clavier pour aller plus vite, ou à signaler la moindre esperluette oubliée. Il est pourtant indispensable de laisser le temps au pilote de se corriger tout seul, et de consacrer son cerveau à un travail de niveau supérieur : quels autres tests unitaires pourrait-on écrire ? N'y a t-il pas du *refactoring* à faire ? Cette portion de code ne pourrait-elle pas être simplifiée ? Notez toutes vos suggestions sur un *post-it*, et attendez une pause du pilote pour en parler avec lui.

D'un point de vue pratique, travailler en binôme nécessite un bon matériel : les bureaux doivent être assez larges pour accommoder 2 personnes de front, de même que l'écran doit être suffisamment grand pour que les 2 y voient sans problèmes. Utiliser 2 moniteurs (sur lesquels l'affichage est dupliqué par exemple) est aussi une bonne solution.

### Rythme soutenable

C'est un fait, un programmeur fatigué et démotivé travaille moins bien. Il fait des erreurs et prend des raccourcis. Même s'il passe beaucoup d'heures au travail, il sera tenté de passer le temps à surfer sur le web ou à vérifier ses emails toutes les 5 minutes. Parce que c'est un professionnel, il s'efforcera malgré tout d'écrire un minimum de code de bonne qualité, mais il sera très loin de la productivité à laquelle il pourrait prétendre. Pour éviter cela, le plus simple est de ne pas faire d'heures supplémentaires. Rentrer à la maison à l'heure, passer du temps avec sa famille et ses amis, bien dormir, et dégager son esprit des problèmes du travail permet de recharger ses batteries et souvent de résoudre les problèmes inconsciemment, simplement en prenant du recul. Après une journée de travail normal, on sera tout simplement satisfait du travail accompli, et on conservera ainsi sa motivation intacte.

Bien évidemment, seul un bon management permet cela : donner des objectifs clairs aux développeurs les gardent motivés ; les protéger des pressions politiques et leur éviter les réunions inutiles permet de les laisser se concentrer sur leur travail. Suggérer des pauses régulières et faire en sorte que les heures supplémentaires soient une exception et pas la règle évite à la fatigue de s'installer.

### Analyse de cause racine

Parce qu'il est plus judicieux de traiter les causes d'un problème que d'en traiter ses symptômes, les équipes XP pratiquent ce type d'analyse pour corriger les défauts de leurs méthodes de travail. Tout le monde peut faire des erreurs, et blâmer les gens pour leurs erreurs ne fait que les encourager à les cacher ou à faire porter le chapeau à quelqu'un d'autre. Face à une erreur ou à un problème, toute l'équipe doit donc travailler ensemble à en trouver la cause afin d'empêcher qu'elle se reproduise. La technique la plus simple, et bien souvent la plus efficace, consiste à poser 5 fois la question «Pourquoi ?». Voici un exemple :

Problème : Il nous arrive de livrer des versions de notre application avec des bugs énormes, empêchant quasiment toute utilisation.

Pourquoi ? Parce que la batterie de tests fonctionnels n'est pas exécutée systématiquement.

On pourrait s'arrêter là, et se dire que l'on a trouvé le problème. Pourtant les développeurs savent qu'ils devraient exécuter systématiquement les tests...

Pourquoi n'exécute t-on pas toujours les tests ? Parce qu'ils prennent 8 heures à exécuter. Pourquoi prennent-ils aussi longtemps à exécuter ? Parce qu'entre chaque test, beaucoup de temps est perdu à mettre la base de données dans un état connu. Pourquoi mettre la base de données dans un état connu prend-il autant de temps ? Parce que

cela est fait en utilisant l'UI de l'application plutôt qu'en accédant directement à la base de données. Pourquoi ne le fait-on pas en accédant directement à la base de données ? Parce que l'outil qui nous permettrait de le faire ne correspond pas totalement à nos attentes.

Nous y voilà ! En posant 5 fois la question *pourquoi*, nous sommes arrivés au coeur du problème, et à sa solution : il nous faut passer un peu de temps à améliorer cet outil afin d'en gagner énormément à l'avenir, et surtout d'éviter que notre application soit livrée sans que les tests aient été exécutés.

Il y a cependant des travers dans lesquels il faut éviter de tomber quand on emploie ce genre de technique. Par exemple, le 5ème pourquoi pourrait pointer vers une personne en particulier qui serait la cause du problème. Si c'est le cas, posez-vous la question : pourquoi cette personne a t-elle fait cela ? Continuez à creuser jusqu'à ce que vous compreniez comment éviter que le problème se reproduise. Parfois vous vous retrouverez en face de plus de causes que vus ne pouvez traiter simultanément. Essayez d'identifier la ou les principales, et attaquez vous-y une à la fois. Enfin, il y aura des cas où résoudre le problème risque de nécessiter beaucoup de temps ou d'entraver votre progression. Demandez vous alors si le problème est suffisamment récurrent pour mériter d'être résolu.

### Rétrospectives

Tout *process* est améliorable, et les équipes XP s'efforcent d'améliorer continuellement leurs pratiques. A cette fin, des *rétrospectives* doivent être organisées à la fin de chaque itération, donc toutes les semaines. Cette réunion, d'une heure maximum, est destinée à chercher ensemble des moyens de s'améliorer. S'il est mal conduite, elle peut facilement tourner à la séance d'auto-flagellation, il convient donc pour tous les participants de l'aborder avec l'esprit ouvert et de laisser de côté leur cynisme ou leurs rancoeurs éventuelles. Si vous sentez que ce n'est pas le cas, attendez la semaine suivante.

En premier lieu, le facilitateur de la réunion doit demander aux membres de l'équipe de penser aux évènements survenus pendant l'itération et de noter des idées sur un tableau blanc divisé en colonnes du type : «Agréable, Frustrant, Mystérieux, Plus, Moins». Selon le nombre d'idées écrites, il pourra être nécessaire de les regrouper en catégories. Ceci fait, demandez aux gens de voter pour l'idée ou la catégorie qui leur semble la plus importante. Il est alors temps de réfléchir à la façon d'améliorer la catégorie choisie. Cela peut être le moment de faire une analyse de cause racine. Une fois que plusieurs idées d'amélioration ont été données, demandez à l'équipe laquelle leur semble la plus judicieuse. Si aucun consensus ne se dégage, votez. Comme il n'y a rien de pire qu'une décision qui n'est pas suivie d'effets, assurez-vous qu'une personne prenne la responsabilité de veiller à ce que la décision soit appliquée.

### Utiliser un même langage

Dans une équipe XP, les développeurs travaillent main dans la main avec les experts métier : pour qu'ils se comprennent et communiquent efficacement, ils doivent absolument utiliser le même langage, celui du métier, pas celui de la technique. Cette approche doit aussi se refléter dans le code écrit : les noms de classes, de méthodes et de variables doivent utiliser des termes métiers.

### Conventions de codage

Tous les développeurs intervenant sur tout le code, il est indispensable d'établir et de respecter des conventions de codage. Ces conventions doivent aller bien au-delà des vieux débats tabulations contre espaces ou retour à la ligne avant une accolade, les développeurs doivent s'accorder sur des points importants comme la gestion des erreurs, la structuration du code, les outils standards, la façon d'utiliser le *logging*, etc... Cela doit être un des premiers points sur lesquels l'équipe travaille. Bien sûr, il sera difficile d'obtenir l'agrément de tous sur certains points, mais ce n'est pas grave, laissez ces points de côté pour l'instant. Si le sujet est d'importance, cela finira fatalement par se voir dans le code, et à ressurgir au cours d'une rétrospective. L'avantage est que vous aurez à ce moment là une expérience supplémentaire qui poussera l'équipe vers la bonne décision.

### Démos d'itération

Une équipe XP produisant une application fonctionnelle toutes les semaines, il est intéressant d'organiser à la fin de l'itération une démonstration pour le client. Cela permet de constater les progrès à la fois pour le client et pour l'équipe, et c'est excellent pour le moral de cette dernière. Cela permet également d'avoir des retours de la part de quelqu'un d'autre que les représentants du client sur site. A la fin de la démonstration, il est d'usage de demander au client s'il est satisfait et si l'équipe doit continuer. Si le client répond négativement à la première question, c'est qu'il y a un problème quelque part. Parlez lui après la démonstration pour essayer de comprendre ce qu'il se passe. Normalement, vous n'êtes pas censés entendre un «non» à la seconde question, si c'est le cas, c'est que vous êtes cuits.

### Vraiment terminé

Bien souvent, les développeurs déclarent avoir terminé une fonctionnalité lorsqu'ils ont simplement fini de coder. Mais la fonctionnalité n'est pas encore montrable au client : elle n'est ni testée, ni intégrée. Avec XP, une fonctionnalité est terminée lorsqu'elle est prête à être mise en production et qu'il n'y aura plus à revenir dessus. Les fonctionnalités à moitié terminées représentent des coûts cachés dans le projet. Quand c'est le moment de livrer, il reste une somme de travail imprévisible à effectuer qui ruine tous les efforts de planification entrepris. Pour éviter ce problème, assurez-vous que toutes les fonctionnalités planifiées sont vraiment terminées à la fin de chaque itération. Chaque fonctionnalité doit être :

- codée,
- testée,
- refactorisée,
- intégrée dans l'application,
- installable (cela comprend les éventuelles mises à jour du schéma de base de données),
- débuggée,
- revue et acceptée par le client.

Cette pratique est étroitement liée avec d'autres que nous allons voir par la suite, en particulier le développement dirigé par les tests et l'intégration continue.

### Intégration continue

L'objectif de l'intégration continue est de s'assurer que l'application est prête à être livrée en permanence. La plupart des équipes de développement ont besoin d'un délai entre le moment où ils ont terminé de coder et la livraison finale, et ce délai est parfois très long : il faut parfaire l'intégration des différents bouts de code, créer un paquet pour l'installation, générer la documentation, tester manuellement, débugger, tester à nouveau, etc... La pratique de l'intégration continue permet d'éviter tout cela, et de conserver un produit livrable chaque semaine. Cette pratique nécessite bien sûr l'utilisation d'un système de contrôle de version (*CVS, Subversion, Git, Mercurial*, ... à vous de choisir !) et la mise en place d'un serveur d'intégration continue (les plus connus sont *CruiseControl* et *Hudson*). La démarche est la suivante : lorsqu'un développeur commence à travailler sur une fonctionnalité, il commence par récupérer une copie de travail du code existant

depuis le dépôt. Il implémente la fonctionnalité, sans oublier de coder les tests associés, vérifie que les tests passent sur sa machine locale, et soumet (commit) le code et les tests, en résolvant les conflits éventuels. Selon la façon dont le serveur d'intégration continue est configuré, à chaque commit, ou toutes les heures, ou tous les jours, il va exécuter l'ensemble des tests automatisés afin de vérifier qu'il n'y a pas de régression, et, selon les cas, construire un paquet déployable de l'application ou la déployer sur un environnement de pré-production (si bien sûr tous les tests ont été exécutés avec succès). L'équipe doit faire en sorte de ne jamais *casser le build*, il est donc virtuellement interdit de soumettre du code non finalisé faisant échouer certains tests. Étant donné qu'il peut être douloureux pour les développeurs de résoudre les conflits lorsque l'on n'a pas fait de commit depuis plusieurs jours, cela rend nécessaire de décomposer les fonctionnalités en petites tâches ne nécessitant pas plus de quelques heures de travail, ceci afin que les développeurs puissent soumettre leur code plusieurs fois par jour. Enfin, il est important de s'assurer que la phase de *build* ne soit jamais trop longue, car cela pourrait avoir des effets très négatifs sur la productivité. Si cela devient le cas, il importe de prendre du temps pour comprendre d'où vient la lenteur et de résoudre le problème.

### Appropriation collective du code

Le fait de concentrer la connaissance du code dans la tête de quelques-uns fait peser un risque énorme sur un projet. Demandez vous ce qui arriverait si votre développeur principal, le seul à connaître la façon dont fonctionne l'algorithme majeur de votre application, venait à quitter la société. Les équipes XP font donc porter la responsabilité de maintenir le code à tous les développeurs. Tout le monde est responsable de sa qualité, et personne ne peut s'attribuer une portion de l'application : tous peuvent faire des changements où que ce soit. Dans la pratique, si un développeur trouve un problème quelque part dans le code, que ce soit un mauvais choix de nommage de variable, un manque de tests unitaires, ou un algorithme mal conçu, il est de son devoir de régler le problème, sans se demander qui l'a engendré. Cela demande bien sûr aux programmeurs de mettre de côté leur ego : ils doivent trouver de la fierté dans le code que leur équipe a écrit, pas uniquement dans le leur ! Là encore, cette pratique est indissociable d'autres : la programmation en binôme vous permet de découvrir les portions de code que vous ne connaissez pas encore, simplement en vous mettant avec quelqu'un qui a déjà travaillé dessus. Les tests unitaires agissent quant à eux à la fois comme une documentation du code et une sécurité. Si vous avez modifié un bout de code que vous ne comprenez pas bien, l'exécution des tests vous révèlera immédiatement la moindre erreur. Il est bien sûr difficile pour un développeur de laisser n'importe qui modifier un code qu'il a écrit et dont il est fier, mais l'avantage est que ce développeur n'est pas obligé d'assumer seul la responsabilité de sa maintenance ! De plus, cela permet à tout le monde d'accroître ses compétences et de travailler sur autre chose que sa spécialité. Si vous employez des développeurs juniors et que vous n'avez pas trop confiance en leurs capacités, assurez-vous qu'ils travaillent en binôme avec des personnes expérimentées, et gardez un oeil sur leur travail. Vous constaterez qu'ils progresseront nettement plus vite qu'avec une méthodologie classique de conduite de projet.

### Pas de documentation écrite

Attention, cette pratique ou plutôt absence de pratique peut prêter à confusion ! Je connais beaucoup d'équipes de développement qui, en lisant ce titre, pourraient se dire : *Dans notre équipe on ne documente jamais, cela veut-il dire que nous sommes agiles ?* Certainement pas. Le but de toute documentation, technique ou non, est la communication. Les différentes pratiques d'XP font que certains types de documentation ne sont pas nécessaires. Le fait que des représentants du client sont sur site, que tout le monde travaille dans un même espace et utilise un langage métier compréhensible par tous, fait que le travail d'écriture de spécifications détaillées est inutile. En utilisant le développement dirigé par les tests, en s'efforçant d'écrire du code clair et simple, et en s'appropriant collectivement le code, les développeurs n'ont plus besoin de documentation technique. Bien sûr, certaines choses doivent être documentées : la vision globale du projet, des décisions de conception importantes ou des détails fonctionnels difficiles à mémoriser. Les *stories* doivent être stockées quelque part, et les divers schémas ou dessins d'interface faits au tableau blanc doivent être pris en photo avant d'être effacés. Certains types de documentation peuvent faire partie des livrables demandés par le client, comme un manuel utilisateur ou une documentation de référence de l'API. Si c'est le cas, écrivez des *stories* pour cela. Et assurez-vous que le client ne vous le demande pas par principe, mais que ces documentations seront réellement exploitées. Les développeurs n'aiment déjà pas ce genre de travail, mais s'ils savent en plus qu'il est inutile, ils risquent de traîner les pieds et d'y passer beaucoup plus de temps que nécessaire.

### Développement dirigé par les tests

Le TDD mérite à lui seul un ou plusieurs articles, je me contenterais donc ici d'une description générale. Le TDD n'est pas une technique de test, mais une technique de conception : il s'agit d'un cycle rapide de test, écriture de code, et remaniement de celui-ci. Lorsqu'un développeur doit implémenter une fonctionnalité, comme par exemple écrire une nouvelle classe métier, il commence par réfléchir à la façon dont son code doit se comporter. Il écrit alors un test vérifiant ce comportement. A ce stade, bien sûr, le test échoue. Le développeur doit alors s'efforcer d'écrire le minimum de code nécessaire à ce que le test passe. Il écrit alors un nouveau test, et ainsi de suite. Au bout d'un moment, le besoin se fera sentir de refactoriser le code pour l'améliorer. En relançant les tests écrits, le développeur peut s'assurer que son remaniement n'a pas modifié le comportement du code. Le TDD cumule ainsi plusieurs avantages. En premier lieu, il incite les développeurs à rester simple et à n'implémenter que ce qui est strictement nécessaire. Le code écrit à l'aide de cette technique contient toujours beaucoup moins de *bugs*, et de plus, il est en général mieux structuré car le fait de devoir le tester vous oblige à respecter les principes de l'orienté objet. Il documente également l'utilisation du code. Enfin, la batterie de tests écrits pendant le développement d'une fonctionnalité complète vous servira de tests de non-régression lorsque vous devrez apporter des modifications à cette fonctionnalité.

### Tests de recette

Le principal travail des représentants du client sur site, outre de répondre aux questions des développeurs, est d'écrire les différents scénarios d'utilisation d'une fonctionnalité, qui seront transcrits par les développeurs sous la forme de tests fonctionnels automatisés. Comme pour les *stories*, le BDD propose un formalisme intéressant pour l'écriture de ces tests, en voici un exemple (simpliste, je vous l'accorde) :

Étant donné qu 'il n'existe pas d'utilisateur *johndoe.*

Quand je crée un utilisateur *johndoe* avec un mot de passe *test.*

Alors l'utilisateur *johndoe* peut se connecter avec le mot de passe *test.*

Notez que le *étant donné que* n'est qu'une traduction approximative du mot-clé anglais *given*. Cet exemple est plutôt un test

d'intégration : l'UI étant concernée, il s'automatiserait avec *Selenium* notamment. Mais les tests fonctionnels peuvent être plus bas niveau et tester la logique métier de l'application. Ces tests doivent alors comprendre des tableaux de données d'entrée et de données attendues en sortie. Si le sujet vous intéresse, je vous recommande de lire la documentation de *Fitnesse* (*http://fitnesse.org*), un framework Java dédié à ce type de tests.

### Refactorisation

Au fur et à mesure que l'on avance dans un projet de développement, le code se complexifie et, hélas, s'enlaidit bien souvent. La tentation du copier/coller est parfois trop grande, le corps des méthodes s'allongent, les cas particuliers amènent leur lot de `if` et autres `switch`. La refactorisation (c'est un anglicisme, le terme original étant *refactoring*) est une pratique visant à remanier le code pour en améliorer sa structure et sa lisibilité, et en faciliter ainsi sa maintenance, sans en affecter le comportement. Elle est donc quasiment indissociable du développement dirigé par les tests, puisque relancer la panoplie de tests permet justement de s'assurer que le comportement n'a pas été modifié. La refactorisation se pratique en permanence : certains symptômes doivent vous alerter et vous inciter à refactoriser. Le code dupliqué est l'exemple le plus simple : remplacez les portions de code concernées par un appel à une méthode unique. De même, d'une méthode trop longue peuvent être extraites plusieurs méthodes simples qui, par un nommage explicite, amélioreront grandement la lisibilité du code. L'important est de procéder par petites étapes, en s'assurant à chacune d'entre elles que le comportement du code n'est pas modifié. Certains outils facilitent grandement cette pratique : un IDE comme Eclipse permet d'automatiser certaines refactorisations simples comme l'extraction d'une méthode ; d'autre part, utiliser un système de contrôle de version décentralisé comme *Git* ou *Mercurial* vous permet de sauvegarder régulièrement vos modifications en local avant de pousser l'ensemble sur le serveur central dès que la refactorisation est terminée.

### Conception simple

Comme le disait Antoine de Saint-Exupéry, *il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher*. Les développeurs agiles cherchent à écrire le code le plus simple possible, c'est même l'un des principes de base du développement dirigé par les tests. Il n'y a en effet rien de pire que d'essayer d'anticiper le changement en introduisant des fonctionnalités qui pourraient éventuellement être utiles plus tard, ou en cherchant à tout prix à permettre l'extensibilité du système. La complexité introduite en faisant cela finit en effet toujours par se payer, que ce soit en difficultés de maintenance ou en problèmes de performances. En restant simple, on ne s'interdit pas les changements futurs ; bien au contraire, ils en seront d'autant facilités. On pourrait croire que cette idée va à l'encontre de l'utilisation des motifs de conception, mais ce serait une erreur : les motifs de conception sont utiles, mais ils ne doivent être utilisés que lorsque le besoin de flexibilité est là, pas en prévision d'un besoin qui n'arrivera peut-être pas. Simple ne veut pas dire simpliste : n'écrivez pas pour autant du code procédural à la va-vite ! Une conception simple produit du code clair et élégant, et certains principes de base peuvent vous y aider :

- You Aren't Gonna Need It (YAGNI) : pas de spéculation ; lorsque vous êtes sur le point d'ajouter un bout de code qui ne répond pas à un besoin immédiat, c'est simple : ne le faites pas !
- Don't Repeat Yourself (DRY) : évitez toute duplication, de code comme de concept.
- Le code ne devrait pas avoir besoin de commentaires : il s'agit bien sûr d'un idéal, et aux yeux de certains, spécialement des développeurs fainéants... Pourtant, un code simple et expressif est facile à comprendre sans l'aide de commentaires, à part peut-être pour certains algorithmes très complexes. Faites relire votre code par vos collègues : s'ils ont des difficultés à le comprendre, c'est qu'il a besoin d'être simplifié.
- Attention aux APIs : dès lors que vous publiez une API (un service web par exemple) ou même une simple interface, vous réduisez vos chances de pouvoir effectuer des changements sans causer des problèmes à ceux qui l'utilisent. Efforcez vous donc de minimiser le nombre d'APIs publiques, et s'il est vraiment nécessaire pour votre produit d'en disposer, ne publiez une API que lorsque le besoin est là et que vos clients la demandent.

### Conception incrémentale

Avec *XP*, l'objectif premier des développeurs est de produire de la valeur ajoutée pour le client, et ce dès la première semaine du projet. Vous n'avez donc pas 6 mois pour écrire un framework ou réfléchir à une architecture, vous devez immédiatement implémenter les *stories* demandées par le client. C'est là que la conception incrémentale rentre en jeu : vous commencez par écrire le code plus simple possible et vous l'étendez au fur et à mesure des besoins, en évaluant en permanence la robustesse de votre code et en le refactorisant si nécessaire. Soyez le plus spécifique possible lorsque vous introduisez un nouveau bout de code, n'essayez pas de généraliser : les abstractions viendront plus tard, lors de la refactorisation si elle s'avère nécessaire. On retrouve bien sûr ici les principes de conception simple et de développement dirigé par les tests. Cela peut sembler contre-nature de développer ainsi, sans conception préalable et sans un énorme schéma UML pour vous guider, et pourtant cela fonctionne mieux. La raison en est simple : vous ne pouvez pas anticiper tous les problèmes et cas particuliers lors d'une phase de conception initiale. Vous produirez certes une architecture qui vous semblera séduisante, mais il y aura fatalement un moment où vous regretterez certains choix faits au départ.

### Pas d'optimisation prématurée

L'optimisation des performances prend du temps, et engendre souvent un code plus complexe et/ou plus difficile à maintenir, ce qui dans les 2 cas n'est pas dans l'intérêt du client. Les développeurs adeptes d'*XP* n'optimisent donc leur code que lorsqu'il y a un réel besoin. Si votre client constate que les performances de l'application le gênent dans son activité, planifiez alors avec lui une *story* d'optimisation. Incluez-y des chiffres : évaluez par exemple le nombre minimum de transactions par seconde acceptable. Lorsque vous commencez à travailler sur la *story*, commencez par écrire un test de performances, et mesurez-les à chaque modification du code. Arrêtez-vous dès que vous avez atteint l'objectif fixé. Enfin, ne procédez pas à l'aveuglette : utilisez des outils de *profiling* comme *Xdebug* pour repérer le goulot d'étranglement.

### En résumé

Comme vous avez pu le constater, les pratiques d'*XP* se renforcent mutuellement : s'il est possible de n'en utiliser que quelques unes pour, par exemple, supplémenter *Scrum*, il convient de les choisir avec soin en tenant compte de leurs liens. Appliquer *XP* demande bien sûr un contexte favorable difficile à trouver. Les obstacles les plus fréquents sont, outre la résistance au changement, le blocage culturel, notamment vis à vis de la programmation en binôme, et la nécessité d'une discipline collective fort peu compatible avec un esprit *jeune pousse*.

## Bibliographie

- The art of agile developpement par Jim Shore et Shane Warden (ISBN : 978-0596527679).
- *The mythical man-month* par Frederick P. Brooks (ISBN : 978-0201835953).
- Clean code par Robert C. Martin (ISBN : 978-0132350884).


## Les outsiders

Deux autres méthodes font parler d'elles depuis peu : il s'agit de *Lean* et de *Kanban*. En voici une description rapide.

### Kanban

*Scrum* est finalement peu normé comparé à *XP* ; et bien *Kanban* l'est encore moins ! Les seules contraintes que *Kanban* impose sont de disposer d'un moyen de visualisation du flux de travail, et de limiter le nombre de tâches concurrentes. Cela se traduit le plus souvent par l'utilisation d'un tableau blanc doté de plusieurs colonnes, comme par exemple (cela devrait vous rappeler quelque chose) : *A faire*, *Dev*, *Test*, *Livré*. Toutes les tâches sont écrites sur des fiches qui sont placées dans le tableau. Mais le point important est que vous devez imposer des limites claires au nombre de tâches pouvant être présentes simultanément dans chaque colonne. Cela vous permet d'éviter de vous disperser en vous obligeant à établir des priorités claires entre les tâches. Bien sûr, si *Kanban* ne vous oblige à rien d'autre, ce n'est pas pour autant que vous ne devez pas inclure d'autres pratiques, comme pourquoi pas certaines pratiques techniques d'XP. Mais dans certains contextes, *Kanban* peut se suffire à lui-même. Prenons par exemple le cas d'une équipe ne faisant que de la maintenance et support de projets existants : ce type d'équipe travaille toujours plus ou moins dans l'urgence, réagissant au cas et par cas et travaillant en priorité sur les tâches les plus urgentes. *Kanban* peut leur permettre de visualiser plus facilement le travail en cours et surtout de fluidifier celui-ci.

### Lean

*Lean* n'est pas vraiment une méthode au sens strict : c'est une pensée, une école de gestion de la production qui a vu le jour chez *Toyota*, dans un contexte très industriel donc. Ses tenants recherchent la productivité et la qualité par un processus d'amélioration continue et d'élimination des gaspillages, le but étant d'arriver à une utilisation optimale des matériaux, ressources et main d'oeuvre. Ses principes ont été adoptés et adaptés au monde informatique, et rencontrent un franc succès en ces temps de crise et de réduction des coûts. Dans un contexte de projet de développement, *Lean* vise à identifier et éradiquer les gaspillages conduisant à l'insatisfaction du client, un manque de profits, ou à un manque de productivité des développeurs. On peut citer les gaspillages suivants :

- les défauts (*bugs*, code de mauvaise qualité, etc...),
- la surproduction (implémentation de fonctionnalités non nécessaires),
- l'attente (lenteur de l'application ou de sa maintenance, ou bien lenteur dans la communication entre les développeurs et le client),
- les procédures sans valeur ajoutée (par exemple le fait de remonter des métriques inutiles à la direction),
- les transports (comme la présence physique imposée à certaines réunions ou pour résoudre des problèmes logiciels sur site),
- la mauvaise gestion des ressources humaines.

Notez que ces éléments peuvent également se combiner entre eux (le fameux *effet domino*) et avoir des répercussions plus graves encore. Afin d'identifier ces gaspillages, *Lean* propose un certain nombre de stratégies, comme la cartographie des flux de valeur ajoutée. Cette méthodologie permet d'analyser le flux des services offerts par un service informatique à ses clients. Si certains de ces services ne possèdent pas de valeur ajoutée, ils doivent être éliminés.

Un autre concept fondamental de *Lean* est le flot : plus tôt le produit est livré au client, plus vite on aura ses retours, qui pourront être incorporés dans la version suivante. *Lean* pousse donc à travailler en itérations, et même à faire des itérations les plus courtes possibles. L'information doit être un flot continu et bidirectionnel entre le client et les développeurs, afin d'éviter le stress généré par une arrivée massive d'informations, comme par exemple les retours du client après 6 mois de développement isolé. Le flot permet aussi de retarder la prise de décisions. En effet, le développement est une activité pleine d'incertitudes et de choix possibles en termes d'architecture ou de technologies, retarder la prise de décisions le plus possible permet de prendre ces décisions en connaissance de cause, en se basant sur des faits plutôt que des hypothèses. Cela facilite l'adaptation au changement, et permet de rattraper ses erreurs plus aisément.

Par bien des côtés, le *Lean* appliqué au développement rejoint donc les principes de *Scrum* ou *XP*, et les complète même par des stratégies utiles pour améliorer ses pratiques. Il est donc intéressant de s'intéresser de près à ce courant de pensée, et de s'efforcer de le mettre en pratique au sein de ses projets, qu'ils respectent une méthodologie agile ou non.

## En conclusion

Il faut garder à l'esprit que toutes ces méthodes ne sont que des outils : ils ne vous garantissent évidemment pas le succès, mais sont là pour vous aider à travailler plus efficacement, en vous proposant certaines contraintes et manières de procéder. Aucune n'est meilleure qu'une autre, le choix doit se faire uniquement en fonction du contexte dans lequel vous travaillez : en tous les cas, il faut bien évidemment un contexte favorable. Le contexte contractuel est notamment très important : selon certains, les contrats dont le tarif est fixé à l'avance sont incompatibles avec les méthodes agiles. Il est pourtant normal que le client dispose d'une enveloppe budgétaire fixe, et il faut alors lui faire comprendre que le périmètre ne peut pas l'être aussi. En échange, l'équipe de développement s'engage à le satisfaire au mieux et à accepter les demandes d'évolution en cours de projet. Pour conclure, j'espère avoir réussi à vous montrer à quel point ces méthodes s'inspirent toutes de valeurs simples : la satisfaction du client et la communication. Adhérer à ces 2 principes élémentaires est déjà un premier pas vers le succès.


**RAPHAËL ROUGERON**

*Raphaël Rougeron travaille au sein du groupe Linagora (http://www.linagora.com) sur des projets web de développement et d'intégration de briques logicielles libres. Il est également le créateur du framework Stato (http://www.stato-framework.org), publié sous licence MIT.*

*Pour contacter l'auteur : goldoraf@gmail.com*

*Son blog : http://www.raphael-rougeron.com*


**Vous pouvez vous abonner :**

**Par téléphone au +33 975 180 358

**Par Fax au +48 22 244 24 59

**Par mail : *abo_fr@software.com.pl*

## Je souhaite m'abonner au magazine PHP Solutions

| 1 | Coordonnées postales : |
|---|---|
| Nom : | |
| Prénom : | |
| Adresse : | |
| | |
| | |
| Code postal : | |
| Ville : | Pays : |

**2 Je règle par :**

☐ Carte bancaire n° CB ____ ____ ____ ____

expire le ____ date et signature obligatoires

type de carte .....

code CVC/CVV ___

☐ Chèque (à envoyer à notre adresse postale)

À la ordre de :

Software Press Sp z o.o. SK, Bokserska 1, 02-682 Varsovie, Pologne

☐ Virement bancaire :

NORDEA BANK POLSKA S.A.

Banque guichet numéro de compte clé

1440 1101 0000 0000 1018 8113

IBAN : PL 55 1440 1101 0000 0000 1018 8113

Adresse Swift (Code BIC) : NDEAPLP2

Date et signature

# Transférer un fichier
## par le biais d'un formulaire HTML

Dans cet article, vous apprendrez à envoyer des fichiers vers un serveur web et à réceptionner les informations et les données des fichiers dans un script PHP. Vous verrez les opérations à réaliser pour activer cette fonctionnalité et pour la mettre en œuvre en toute sécurité.

**Cet article explique :**
- Comment télécharger un fichier de son disque dur, via un navigateur, vers un serveur web.

**Ce qu'il faut savoir :**
- Vous devez connaître les bases du langage PHP et de HTML.

Niveau de difficulté


De nombreuses applications web fournissent un mécanisme de téléchargement de fichiers, du disque du client vers celui du serveur web (galeries de photos, CMS, ...). Cet article explique comment créer un formulaire HTML avec des sélecteurs de fichiers, comment configurer le serveur web afin qu'il autorise le transfert, et comment récupérer les informations sur les fichiers, et les fichiers téléchargés, dans un script PHP.

### Créer un formulaire d'envoi de fichier

Un navigateur peut envoyer un ou plusieurs fichiers au serveur web par l'intermédiaire d'un formulaire HTML. La Figure 1 donne un exemple de formulaire HTML, généré par le Listing 1. Les étapes pour créer un formulaire avec envoi de fichier(s) sont les suivantes :

- définir les attributs de l'élément HTML `form`,
- ajouter un sélecteur de fichier par fichier à envoyer,
- ajouter un bouton d'envoi de données.

Le Listing 1 crée un formulaire HTML contenant un sélecteur de fichier. Un clic sur le bouton du sélecteur permet de parcourir l'arborescence du disque, afin de choisir le fichier à envoyer. Une fois un fichier sélectionné, son chemin sur le disque est affiché dans le champ texte. Quand l'internaute clique sur le bouton *Soumettre*, le fichier est envoyé par la méthode `POST` au script PHP *traite_formulaire.php* (Listing 2).

#### Elément form

L'élément `form` marque le début et la fin d'un formulaire HTML. Il inclut un élément par champ de formulaire :

- `select` pour les listes déroulantes à choix unique ou multiples,
- `textarea` pour les champs de saisie de texte multi-lignes,
- `button` pour obtenir un bouton personnalisé (image, texte, ...),
- `input` pour les champs de saisie de texte visible (*text*) ou masqué (*password*), les champs texte cachés (*hidden*), les cases à cocher (*checkbox*), les boutons radio (*radio*), les sélecteurs de fichiers (*file*), les boutons de soumission (*submit*) ou de réinitialisation du formulaire (*reset*) et les boutons auxquels des actions peuvent être associées en JavaScript (*button*). L'attribut `type` permet de choisir le type de contrôle de formulaire à afficher pour chaque élément `input`, les valeurs possibles de cet attribut sont celles données entre parenthèses.

L'attribut `action` indique l'adresse du script chargé de traiter les données du formulaire. Cette adresse peut être donnée en relatif (chemin donné par rapport à l'emplacement du chemin du formulaire HTML sur le serveur) ou en absolu (URL). Dans le Listing 1, le formulaire est envoyé au script *traite_formulaire.php* qui est situé dans le même répertoire sur le serveur que le fichier *formulaire.html*.

Un formulaire peut être envoyé par la méthode `HTTP GET` ou `POST`. Dans le premier cas les données sont envoyées dans l'en-tête de la requête HTTP, elles sont placées dans la *query string* (partie après le caractère point d'interrogation dans l'URL). Dans le second cas elles sont envoyées dans le corps de la réponse HTTP. L'attribut `method` de l'élément


**Figure 1.** *Formulaire d'envoi de fichier*

`form` indique la méthode `HTTP` à utiliser lors de la soumission des données au serveur, par défaut le formulaire est envoyé par la méthode `GET`. Les fichiers doivent toujours être envoyés dans le corps de la requête HTTP, l'attribut `method` de l'élément `form` doit donc avoir la valeur `POST`.

L'attribut `enctype` permet de spécifier l'encodage des valeurs envoyées au serveur, par défaut cet encodage est `application/x-www-form-urlencoded`. Pour envoyer un fichier au serveur il faut utiliser l'encodage `multipart/form-data`.

### Sélecteur de fichier

Pour créer un sélecteur de fichier, il faut donner la valeur `file` à l'attribut `type` de l'élément `input`. Le sélecteur comporte deux parties : un champ texte et un bouton dont l'intitulé peut varier selon le système d'exploitation et le navigateur. Le bouton permet de parcourir l'arborescence du disque dur du client afin de choisir le fichier à envoyer au serveur. Un clic sur le bouton du sélecteur ouvre une fenêtre de sélection de fichier. Une fois le fichier sélectionné, son chemin est placé dans le champ texte du sélecteur.

L'attribut `name` définit le nom du sélecteur, le script PHP utilisera ce nom pour accéder aux informations sur le fichier et à ses données. Le code ci-après crée un sélecteur de fichier dont le nom est `fichier` :

```
<input type='file' name='fichier' >
```

### Bouton d'envoi

Lorsque l'attribut `type` de l'élément `input` prend la valeur `submit` ou `image`, un bouton de soumission de formulaire est créé. Un clic sur ce bouton a pour effet d'envoyer les données du formulaire à l'adresse définie dans l'attribut `action` de l'élément `form`. Le texte du bouton est défini par l'attribut `value`. Le texte par défaut dépend de la version du navigateur utilisé. Dans l'exemple du Listing 1, le bouton est intitulé *Soumettre*, les données du formulaire sont envoyées au script `traite_formulaire.php`.

### Elément caché MAX_FILE_SIZE

PHP utilise deux mécanismes pour définir la taille maximale autorisée pour un téléchargement de fichier : les directives du fichier de configuration *php.ini*, présentées dans la section suivante, et un champ caché de formulaire dont le nom est `MAX_FILE_SIZE`. Ce champ caché est un élément `input` dont l'attribut `type` a la valeur `hidden`, l'attribut `name` a la valeur `MAX_FILE_SIZE` et l'attribut `value` contient le nombre d'octets autorisés. Le champ caché doit précéder le sélecteur de fichier dans le code HTML. Lorsque PHP reçoit la variable `MAX_FILE_SIZE`, il l'utilise pour déterminer si la taille du fichier envoyé est supérieure à celle autorisée dans le formulaire. Si le fichier dépasse la taille indiquée, un code d'erreur est placé dans le tableau des informations pour le fichier (les codes d'erreur sont présentés dans la section intitulée *Gérer les erreurs*).

**Listing 1.** *formulaire.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=UTF-
8">
        <title>Formulaire d'envoi de fichier</title>
        <link rel="stylesheet" type="text/css" href="style_form.css">
    </head>
    <body>
        <h1>Formulaire d'envoi de fichier</h1>
            <form action="traite_formulaire.php" method="post"
enctype="multipart/form-data">
            <div>
                <input type="file" name="fichier"><br>
                <input type="submit" name="envoi" value="Soumettre">
            </div>
            </form>
    </body>
</html>
```

Le site officiel de PHP indique que la taille maximale autorisée, définie avec `MAX_FILE_SIZE`, est purement informative pour le client. La note sous-entend que le navigateur pourrait utiliser cette information pour bloquer l'envoi de fichiers supérieurs à la taille souhaitée. Actuellement les principaux navigateurs ne bloquent pas l'envoi quand la taille du fichier dépasse la valeur définie dans le champ caché (tests réalisés avec *Firefox 3.5*, *Internet Explorer 8*, *Opera 10*, *Safari 4*). De plus, la fonctionnalité n'est décrite ni dans un RFC, ni dans les recommandations du W3C, il est donc peu probable que les navigateurs l'implémentent. L'ajout de ce champ caché dans un formulaire, n'est donc actuellement pas très utile. Il est important de noter également que la sécurité de l'application ne doit pas reposer sur ce champ caché dont la valeur peut être facilement modifiée avant l'envoi du formulaire. Pour toutes ces raisons, le champ caché `MAX_FILE_SIZE` n'est pas utilisé dans cet article.

## Vérifier les réglages du serveur web

Plusieurs directives des fichiers de configuration de PHP *php.ini* et Apache *httpd.conf* contrôlent le téléchargement de fichiers. Avant de développer des scripts de traitement de fichiers téléchargés sur un serveur, il faut s'assurer que le serveur autorise le téléchargement et vérifier les valeurs des directives limitant la taille des téléchargements.

Si vous n'administrez pas le serveur web, vous ne pourrez pas obtenir ou modifier les valeurs des directives du fichier de configuration Apache, et de certaines directives PHP.

### Configuration PHP

Exécutez le script `<?php phpinfo(); ?>` pour vérifier les valeurs des directives PHP décrites ci-après.

La directive `file_uploads` active ou désactive la fonctionnalité d'envoi de fichier du client vers le serveur. Pour autoriser le téléchargement la valeur doit être à `On` :

```
file_uploads = On
```

L'emplacement du répertoire où les fichiers envoyés par le navigateur sont stockés sur le serveur, est défini par la directive `upload_tmp_dir`. Le répertoire de téléchargement doit être accessible en écriture pour PHP. Dans la configuration ci-après, le répertoire où les fichiers sont stockés est le répertoire *tmp* situé à la racine du système de fichier. La première ligne montre cette directive pour un système windows, la seconde pour un système *linux/mac os X* :

```
upload_tmp_dir = "C:/tmp"
upload_tmp_dir = "/tmp"
```

La taille maximale de téléchargement dépend de deux directives : `post_max_size` et `upload_max_filesize`. La première définit la taille maximale autorisée pour les données postées. La seconde indique la taille maximale pour chaque fichier téléchargé, la valeur donnée doit être inférieure à celle définie dans `post_max_size`. Dans la configuration ci-après, PHP accepte jusqu'à 8 Mo de données par la méthode `POST`, chaque fichier téléchargé ne doit pas dépasser 3 Mo :

```
post_max_size = 8M
upload_max_filesize = 3M
```

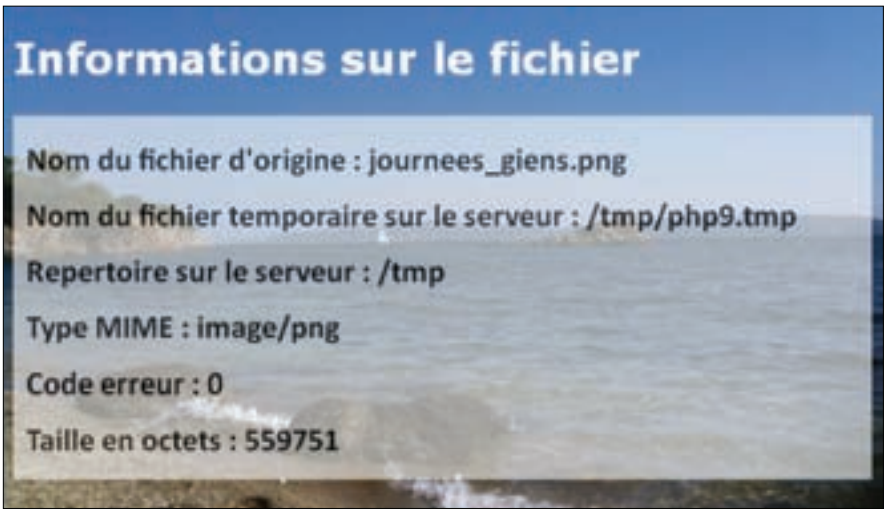



**Figure 2.** *Informations sur le fichier envoyé.*

Dans le cas de téléchargement de gros fichiers, il faut également veiller à ce que la limite maximale d'utilisation mémoire par le script PHP (directive `memory_limit`), le temps d'exécution maximal alloué au script (directive `max_execution_time`) et le temps de traitement des données en entrée (directive `max_input_time`) soient suffisants.

Si vous êtes administrateur, vous pouvez modifier les valeurs des directives directement dans le fichier *php.ini* et relancer le serveur web. Sinon, vous pouvez changer la valeur de certaines directives dans un fichier *.htaccess*. Il faut noter que la fonction `ini_set` ne permet pas de changer les valeurs des directives décrites dans cette section, en effet les directives `upload_max_filesize` et `post_max_size` ont le mode `PHP_INI_PERDIR` (modification possible dans les fichiers *php.ini*, *.htaccess* ou *httpd.conf*) et les directives `file_uploads` et `upload_tmp_dir` ont le mode `PHP_INI_SYSTEM` (modification possible dans les fichiers *php.ini* ou *httpd.conf*).

Les valeurs des directives de mode `PHP_INI_PERDIR` peuvent être fixées dans un *.htaccess*. Les directives booléennes sont paramétrées avec `php_flag`, celles qui prennent une valeur non booléenne sont affectées avec `php_value` :

```
php_value upload_max_filesize 15M
```

Lorsqu'une directive est paramétrée dans un fichier *.htaccess*, elle est définie pour tous les scripts PHP situés dans le même répertoire que le *.htaccess* ou dans des sous-répertoires. L'utilisation d'un *.htaccess* n'est possible que si le fichier de configuration du serveur Apache l'autorise, c'est-à-dire si la directive `AllowOverride` a la valeur `All` ou `Options` pour le répertoire racine du site web ou le répertoire contenant le *.htaccess*.

### Configuration Apache

Le fichier de configuration du serveur web Apache *httpd.conf* peut comporter une directive `LimitRequestBody` qui limite le nombre d'octets autorisés dans le corps de la requête HTTP. Cette limite peut aller de 0 (non limité) à 2 Go. Dans la configuration ci-après, Apache accepte de télécharger des fichiers dans une limite de 100 Ko :

```
LimitRequestBody 102400
```

Lorsque la taille du corps de la requête est supérieure à celle indiquée, un message d'erreur est retourné par Apache (statut 413) : *Request Entity Too Large*.

## Obtenir des informations sur le fichier téléchargé

Le tableau associatif *superglobal* `$_FILES` contient les informations concernant le ou les fichiers envoyés par le navigateur. Ce tableau comporte une case par fichier envoyé par le client. L'instruction ci-après affiche le contenu du tableau `$_FILES` :

```
print_r($_FILES);
```

Dans le cas de l'envoi du formulaire du Listing 1, l'instruction affiche :

```
Array (
    [fichier] => Array (
        [name] => journees_giens.png
        [type] => image/png
        [tmp_name] => /tmp/php9.tmp
        [error] => 0
        [size] => 559751 ) )
```

Le tableau associatif `$_FILES` a pour clés les noms donnés aux champs `input` du formulaire de type `file`. Par exemple, `$_FILES['fichier']` permet d'accéder aux informations concernant le fichier nommé `fichier` dans le formulaire d'envoi. La case `fichier` est un tableau associatif contenant toutes les informations sur le fichier. Les informations disponibles sont le nom du fichier envoyé par le client (*name*), la taille du fichier en octets (*size*), le type MIME du fichier (*type*),

**Listing 2.** *traite_formulaire.php*

```
<?php
$info = '';
if (isset($_FILES['fichier'])){
    $info = 'Nom du fichier d\'origine : '.$_FILES['fichier']['name'].'<br>';
    $info .= 'Nom du fichier temporaire sur le serveur : '.$_
FILES['fichier']['tmp_name'].'<br>';
    $info .= 'Repertoire sur le serveur : '.ini_get('upload_tmp_
dir').'<br>';
    $info .= 'Type MIME : '.$_FILES['fichier']['type'].'<br>';
    $info .= 'Code erreur : '.$_FILES['fichier']['error'].'<br>';
    $info .= 'Taille en octets : '.$_FILES['fichier']['size'].'<br>';
} else {
    $info = 'aucune information';
}
?>
<!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=UTF-
8">
        <title>Informations upload</title>
        <link rel="stylesheet" type="text/css" href="style_infos.css">
    </head>
    <body>
        <h1>Informations sur le fichier</h1>
        <div>
        <?php
          echo $info;
        ?>
        </div>
    </body>
</html>
```

**Tableau 1.** *Directives du fichier de configuration PHP*

| Nom de la directive | Configuration |
|---|---|
| file_uploads | Cette directive active (valeur `On`) ou désactive (valeur `Off`) la fonctionnalité d'envoi de fichier du client (navigateur) vers le serveur. |
| upload_tmp_dir | Cette directive permet de préciser l'emplacement du répertoire sur le serveur, dans lequel seront placés les fichiers envoyés par le client. Le chemin d'accès est une chaîne de caractères donnée entre quotes ou double quotes. |
| post_max_size | Cette directive définit la taille maximale acceptée pour l'ensemble des données envoyées par le client. Elle prend comme valeur un entier (nombre d'octets) ou un entier suivi d'une unité de mesure (ex : 8M pour 8 Mégaoctets). Les mesures disponibles sont K (Kilooctet), M (Mégaoctet) et G (Gigaoctet), |
| upload_max_filesize | Cette directive définit la taille maximale acceptée pour un fichier envoyé par le client. Sa valeur est définie comme pour la directive `post_max_size`, elle doit cependant être inférieure à celle définie dans `post_max_size`. |

**Tableau 2.** *Informations sur les fichiers téléchargés ($_FILES['nom_champ_input_file']['clé'])*

| Clé | Valeur |
|---|---|
| name | Nom du fichier sur le disque du client. |
| type | Type MIME du fichier dans la requête HTTP (exemple : image/gif). |
| size | Taille en octets du fichier envoyé. |
| tmp_name | Chemin du fichier temporaire téléchargé sur le serveur. |
| error | Code d'erreur du téléchargement. |

**Tableau 3.** *Codes d'erreur de téléchargement ($_FILES['nom_champ_input_file']['error'])*

| Code | Constante correspondante | Signification |
|---|---|---|
| 0 | UPLOAD_ERR_OK | Pas d'erreur au téléchargement. |
| 1 | UPLOAD_ERR_INI_SIZE | Le fichier envoyé dépasse la taille maximale autorisée définie dans le php.ini (directive `upload_max_filesize`). |
| 2 | UPLOAD_ERR_FORM_SIZE | Le fichier envoyé est supérieur à la taille définie dans le formulaire HTML, dans le champ caché `MAX_FILE_SIZE`. |
| 3 | UPLOAD_ERR_PARTIAL | Le serveur n'a pas reçu la totalité du fichier. |
| 4 | UPLOAD_ERR_NO_FILE | Aucun fichier n'a été téléchargé. |
| 6 | UPLOAD_ERR_NO_TMP_DIR | Le répertoire temporaire de téléchargement sur le serveur est manquant. |
| 7 | UPLOAD_ERR_CANT_WRITE | Le fichier n'a pu être enregistré sur le serveur, échec à l'écriture. |

le nom du fichier temporaire dans lequel le fichier a été stocké (*tmp_name*) et depuis la version 4.2.0 de PHP le code d'erreur associé au téléchargement (*error*). Ainsi, le code ci-après affiche la taille du fichier téléchargé :

```
echo $_FILES['fichier']['size'];
```

Le Listing 2 affiche toutes les informations du fichier téléchargé par le formulaire du Listing 1. La Figure 2 montre le résultat de l'exécution de ce script.

## Gérer les erreurs

Les principaux problèmes rencontrés lors d'un téléchargement sont recensés dans cette section, ainsi que les traitements à réaliser pour les corriger.

### Tableau des fichiers vide

Si vous obtenez un tableau vide lorsque vous exécutez `print_r($_FILES)`, vérifiez les points ci-après. PHP ne remplit pas le tableau `$_FILES` lorsqu'une des conditions est vraie :

- la directive `file_uploads` est à `Off`,
- l'attribut `method` de l'élément `form` est absent ou n'a pas la valeur `POST`,
- l'attribut `enctype` de l'élément `form` est absent ou n'a pas la valeur `multipart/form-data`,
- le sélecteur de fichier n'a pas d'attribut `name`,
- le fichier a une taille supérieure à celle définie dans la directive `post_max_size`.

### Codes d'erreurs

PHP attribue un code d'erreur à tout fichier téléchargé, ce code est accessible avec la clé `error` dans le tableau associatif contenant les informations d'un fichier. Par exemple, le code suivant affiche le code d'erreur pour le fichier correspondant au sélecteur du formulaire du Listing 1 :

```
echo $_FILES['fichier']['error'];
```

Si le fichier est téléchargé correctement, le code d'erreur contient la valeur 0 (constante `UPLOAD_ERR_OK`). Sinon, il contient un des codes d'erreur :

- `UPLOAD_ERR_INI_SIZE` (valeur 1) : taille du fichier téléchargé supérieure à celle définie dans la directive `upload_max_filesize`, dans ce cas `error` a la valeur 1 et `size` la valeur 0,
- `UPLOAD_ERR_FORM_SIZE` (valeur 2) : le fichier téléchargé est supérieur à la taille définie en octets dans le champ caché de formulaire dont le nom est `MAX_FILE_SIZE`. Ce champ est décrit dans la section intitulée *Créer un formulaire d'envoi de fichier.*
- `UPLOAD_ERR_PARTIAL` (valeur 3) : le serveur n'a pas reçu la totalité du fichier.
- `UPLOAD_ERR_NO_FILE` (valeur 4): aucun fichier téléchargé.
- `UPLOAD_ERR_NO_TMP_DIR` (valeur 6) : pas de répertoire temporaire pour stocker le fichier.
- `UPLOAD_ERR_CANT_WRITE` (valeur 7, depuis PHP 5.1.0) : le fichier téléchargé ne peut pas être enregistré sur le disque.

**Listing 3.** *traite_formulaire_2.php*

```
<?php

// inclusion des fonctions
require 'outils.php';

// initialisation du message d'erreur
$erreur = "";
// extensions autorisees
$tab_ext = array('pdf');
// le formulaire a ete poste
if (isset($_POST['envoi'])) {
    // verifier si telechargement OK
    $erreur = verifUpload('fichier');
    if (empty($erreur)){
        // verification extension
        $ext = getExtension($_FILES['fichier']['name'], $tab_ext);
        if (empty($ext)){
            $erreur = "Le fichier envoye n'est pas au format : ".
implode(',', $tab_ext);
        } else {
            // definir le chemin du repertoire de stockage
            $rep_upload = '/upload/'.$ext;
            $chemin_rep_dest = dirname($_SERVER['SCRIPT_
FILENAME']).$rep_upload;
            // deplacer le fichier
            $erreur = deplaceFichier($_FILES['fichier']['tmp_name'],
$chemin_rep_dest, 'fichier', $ext);
        }
    }
} else {
    $erreur = 'aucune information';
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/
html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>Traitement du fichier</title>
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <h1>Traitement du fichier</h1>
        <div>
        <?php
            // le formulaire a ete poste et le fichier correctement envoye
            if (isset($_POST['envoi']) && empty($erreur)) {

                echo '<div class="reussite">Le fichier a ete sauve sur le
disque</div>';
            // erreurs
            } else {
                echo "<div class='erreur'>$erreur</div>";
            }
        ?>
        </div>
    </body>
</html>
```

– `UPLOAD_ERR_EXTENSION` (valeur 8, depuis PHP 5.2.0) : téléchargement interrompu.

Des erreurs peuvent également survenir lorsque l'espace mémoire alloué dans le fichier de configuration est insuffisant (directive `memory_limit` du *php.ini*), ou lorsque les durées d'exécution de script et de traitement des données sont insuffisantes (directives `max_input_time` et `memory_limit`).

Le script *outils.php* (Listing 4) contient des fonctions générales pour le téléchargement de fichiers. La fonction `getErreurMsg` retourne un message d'erreur en fonction du code d'erreur qu'elle reçoit en paramètre. Si le téléchargement s'est déroulé correctement, elle retourne une chaîne vide. Cette fonction sera utilisée dans les prochaines sections de cet article.

## Récupérer le fichier

Vous avez vu, dans la section intitulée *Créer un formulaire d'envoi de fichier*, comment envoyer un fichier vers le serveur à partir d'un navigateur. Les fichiers envoyés sur le serveur sont stockés dans un répertoire qui contient les fichiers temporaires. L'emplacement de ce répertoire est défini par la directive PHP `upload_tmp_dir` dans le fichier *php.ini*. Les fichiers sont effacés du disque à la fin de l'exécution du script. Vous allez maintenant apprendre à récupérer ces fichiers temporaires pour les stocker dans un répertoire final sur le serveur, afin de les sauver de manière permanente et de pouvoir les manipuler.

Le Listing 3 traite le fichier envoyé par le formulaire du Listing 1. Pour tester cet exemple, modifiez l'attribut action de l'élément `form` dans le formulaire du Listing 1, afin qu'il poste maintenant les données au script *traite_formulaire_2.php* (Listing 3). Le script n'accepte que les fichiers au format PDF.

Le script *traite_formulaire_2.php* s'assure que le formulaire a bien été posté et si c'est le cas vérifie alors que le fichier a été correctement téléchargé sur le serveur. Il contrôle ensuite le type de fichier afin de s'assurer qu'il s'agit bien d'un fichier PDF et enfin sauve le fichier dans un répertoire dédié au stockage permanent des fichiers téléchargés. Le script utilise des fonctions utiles pour tous les scripts qui réalisent des téléchargements. Ces fonctions ont été placées dans le fichier *outils.php*, elles seront réutilisées dans le dernier exemple de l'article (Listing 5).

### Vérifier l'envoi du fichier

Des erreurs peuvent survenir lors du transfert, elles peuvent avoir des causes multiples, comme vous l'avez vu dans la section intitulée *Gérer les erreurs*. La fonction `getErreurMsg` définie dans le script *outils.php* (Listing 4) permet de vérifier que le fichier a été correctement téléchargé et stocké sur le serveur.

Pour des raisons de sécurité, il faut également s'assurer que le fichier à déplacer est un fichier issu d'un téléchargement. Deux approches sont possibles pour réaliser cette vérification. La première solution consiste à vérifier que la case `error` du tableau `$_FILES`, correspondant au fichier téléchargé, est bien égale à la constante `UPLOAD_ERR_OK` (entier 0). Si c'est le cas, aucune erreur n'est survenue. Dans l'exemple, les informations sur le fichier envoyé sont obtenues à partir du sous-tableau `$_FILES['fichier']`, le champ d'envoi dans le formulaire étant nommé `fichier`. Le code suivant affiche un message de réussite dans le cas d'un téléchargement correctement exécuté :

```
if ($_FILES['fichier']['error'] === 0){
    echo 'Envoi ok';
}
```

La seconde solution consiste à utiliser la fonction PHP `is_uploaded_file`.Cette fonction prend en paramètre le chemin d'accès à un fichier et retourne un booléen : la valeur `true` si le fichier passé en paramètre a bien été envoyé sur le serveur par la méthode `POST`, la valeur `false` sinon. L'argument donné en paramètre

est le nom du fichier temporaire sur le serveur. Celui-ci a été téléchargé dans le répertoire défini par la directive `upload_tmp_dir` du fichier de configuration *php.ini*. La case `tmp_name` du tableau `$_FILES` correspond donc au chemin d'accès du fichier temporaire sur le serveur. Le code ci-après affiche un message de réussite si le fichier a été téléchargé par la méthode `POST` :

```
if (@is_uploaded_file($_FILES['fichier']
['tmp_name'])){
echo 'Envoi ok';
}
```

Les vérifications sont effectuées dans le Listing 3 par la fonction `verifUpload` (Listing 4). Cette fonction prend en paramètre le nom donné au champ du formulaire qui contient le fichier. Dans l'exemple la fonction est donc appelée avec la chaîne de caractères `fichier`. La fonction `verifUpload` vérifie que le tableau `$_FILES` contient des informations pour le champ de formulaire demandé, si ce n'est pas le cas elle stocke un message d'erreur dans la variable `$erreur`. Cette variable, initialisée à vide, est retournée par la fonction. Si le tableau des informations est renseigné, la fonction vérifie le code d'erreur. Si la fonction `getErreurMsg`, décrite dans la section dédiée aux erreurs, retourne une chaîne non vide, alors l'erreur est stockée dans la variable `$erreur`. La fonction `is_uploaded_file` est utilisée pour vérifier que le fichier provient d'un téléchargement par la méthode `POST`. A l'issue de toutes ces vérifications, la variable `$erreur` contient soit une chaîne vide (succès), soit un message d'erreur. La chaîne retournée est stockée dans le script du Listing 3, s'il y a un message d'erreur alors le Listing 3 affiche une page HTML contenant le message d'erreur. Sinon, le format du fichier est vérifié, comme expliqué dans la section suivante.

### Vérifier le format du fichier téléchargé

Deux approches sont utilisées pour vérifier le format d'un fichier téléchargé. La première se base sur l'information envoyée par le navigateur dans le champ *Content-Type* de l'en-tête de la requête HTTP, la seconde considère l'extension du fichier. Ces deux informations sont stockées dans le tableau fourni pour chaque fichier téléchargé. Ces informations peuvent être utilisées pour indiquer à un utilisateur que le fichier qu'il a envoyé ne correspond pas au type attendu par l'application, mais elles ne sont en aucun cas une mesure de sécurité suffisante pour se protéger d'un utilisateur malveillant. La section suivante intitulée *Sécurité* expliquera plus précisément les problèmes de sécurité rencontrés lors de l'envoi de fichier par

**Listing 4.** *outils.php*

```php
<?php
/**
 * Initialise un message d'erreur en fonction du code d'erreur.
 * @param  $code_erreur int code d'erreur du tableau $_FILES
 * @return $erreur string texte d'erreur
 */
function getErreurMsg($code_erreur) {
    $erreur = '';
    switch($code_erreur) {
        case UPLOAD_ERR_OK :
            // succes
            $erreur = '';
            break;
        case UPLOAD_ERR_INI_SIZE :
        case UPLOAD_ERR_FORM_SIZE :
            $erreur = "La taille du fichier est trop grande";

            break;
        case UPLOAD_ERR_PARTIAL :
            $erreur = "Le serveur n'a pas recu la totalite du fichier";

            break;
        case UPLOAD_ERR_NO_FILE :
            $erreur = "Aucun fichier telecharge";
            break;
        case UPLOAD_ERR_NO_TMP_DIR :
        case UPLOAD_ERR_CANT_WRITE :
            $erreur = "Impossible de stocker le fichier";
            break;
        case UPLOAD_ERR_EXTENSION :
            $erreur = "Telechargement interrompu";
            break;
        default :
            $erreur = "Erreur de telechargement";
            break;
    }
    return $erreur;
}

/**
 * Verifie que l'extension du fichier est autorisee et la retourne.
 * @param  $nom_fichier string nom du fichier
 * @param  $tab_ext tableau d'extensions autorisees
 * @return string extension trouvee
 */
function getExtension($nom_fichier, $tab_ext){
    // recuperer l'extension du fichier telecharge
    $ext_fichier = strtolower(substr($nom_fichier, -3, 3));
    // verifier que le type du fichier envoye par formulaire est autorise
    if (!in_array($ext_fichier, $tab_ext)){
        $ext_fichier = '';
    }
    return $ext_fichier;
}

/**
 * Genere un nom unique de fichier
 * @param  $prefixe string prefixe pour le nom
 * @param  $ext string extension
 * @param  $chemin_rep_dest string chemin du repertoire de destination
 * @return string chemin d'acces au fichier
 */
function genereNom($prefixe, $ext, $chemin_rep_dest){
    // creer un nom de fichier unique : prefixe + timestamp courant + ext
    $nom = $prefixe . '_' . time() . '.' . $ext;
    // chemin sur le serveur ou le fichier sera place
    $chemin_fichier = $chemin_rep_dest.'/'.$nom;
    return $chemin_fichier;
}

/**
 * Verifie le telechargement et l'extension.
 * @param  $nom_champ string nom du champ du formulaire
 * @return string erreur rencontree
 */
function verifUpload($nom_champ){
    $erreur = '';
    // aucune information sur le telechargement
    if (!isset($_FILES[$nom_champ])){
        $erreur = "Champ $nom_champ : tableau de donnees vide";
    }
```

**Listing 4.** *outils.php – suite*

```
    else {
        // verification du code d'erreur
        $erreur = getErreurMsg($_FILES[$nom_champ]['error']);
        $erreur = (!empty($erreur))? "Champ $nom_champ : $erreur" : "";
        // le fichier a bien ete telecharge
        if (!@is_uploaded_file($_FILES[$nom_champ]['tmp_name'])){
            $erreur .= ' (telechargement incorrect)';
        }
    }
    return $erreur;
}

/**
 * Deplace le fichier temporaire
 * @param  $chemin_src string chemin du fichier a deplacer
 * @param  $chemin_rep_dest string chemin de destination
 * @param  $prefixe prefixe pour le nom unique a generer
 * @param  $type extension du fichier
 * @param  $chemin_fichier string chemin du fichier dans le repertoire de
destination
 * @return string erreur rencontree
 */
function deplaceFichier($chemin_src, $chemin_rep_dest, $prefixe, $type,
&$chemin_fichier = null){
    $erreur = '';
    // verifier qu'il est possible d'ecrire dans le repertoire de
sauvegarde des fichiers
    if (!is_writable($chemin_rep_dest)) {
        $erreur = 'Repertoire de sauvegarde inaccessible en ecriture';

    } else {
        $chemin_fichier = genereNom($prefixe, $type, $chemin_rep_dest);
        // verifier si le fichier n'est pas present sur le serveur
        if (file_exists($chemin_fichier)){
            $erreur = 'le fichier existe deja';
                } else {
            // sauver le fichier dans le repertoire final sur le serveur
            if (!@move_uploaded_file($chemin_src, $chemin_fichier)) {

                // erreur de deplacement du fichier
                $erreur = 'Erreur lors du stockage du fichier sur le
disque';
            }
        }
    }
     return $erreur;
}
?>
```

la méthode `POST`. Pour l'instant, il est important de noter que le type MIME du fichier donné dans les informations du tableau (case `$_FILES['fichier']['type']`), et l'extension dans le nom du fichier envoyé (case `$_FILES['fichier']['name']`) ne sont pas vérifiées par le serveur. Ces données fournies par le navigateur sont facilement modifiables et il ne faut donc pas s'y fier.

L'exemple du Listing 3 n'autorise que le téléchargement de fichiers au format PDF. Tout autre type de fichier ne devra donc pas être sauvegardé sur le serveur de manière permanente. Afin de vérifier que le fichier transféré dans le répertoire de téléchargement du serveur est bien du type attendu, le script se base sur l'extension du fichier reçu. Le nom du fichier (case `$_FILES['fichier']['name']`) et la liste des extensions autorisées sont fournis à la fonction `getExtension` du script *outils.php*. Celle-ci extrait l'extension du fichier et vérifie qu'elle fait partie de celles listées dans le tableau passé en paramètre. L'extension du fichier téléchargé est obtenue grâce à la fonction PHP `substr`, qui retourne les trois derniers caractères du nom du fichier. La chaîne ainsi extraite est ensuite transformée en lettres minuscules, afin d'effectuer une comparaison sans tenir compte de la casse. Si l'extension fait partie de la liste, elle est retournée par la fonction, sinon la chaîne vide est retournée. Le script du Listing 3 stocke un message d'erreur si l'extension n'est pas valide et l'affiche dans la page web. Une fois l'extension validée, il reste à renommer et déplacer le fichier temporaire.

Il faut noter que dans cet exemple simple, seules les extensions de trois caractères sont prises en compte. La fonction pourrait être adaptée pour extraire des extensions comportant plus de caractères (*.jpeg*, *.tiff*, ...).

### Définir le répertoire de destination

Une fois le téléchargement et le format vérifiés, il faut fixer le répertoire dans lequel le fichier sera transféré de manière définitive et renommer le fichier avant de le déplacer. Pour des raisons de sécurité, il est impératif de renommer le fichier envoyé. La section suivante, dédiée à la sécurité, expliquera plus précisément les implications de sécurité lors de l'utilisation du nom d'origine du fichier. Elle indiquera également des méthodes pour protéger le répertoire des accès directs par URL.

Dans le Listing 3, le fichier PDF envoyé par le formulaire est sauvegardé dans le sous-répertoire `pdf` du répertoire `upload`, qui est lui-même situé au même niveau sur le disque que le script de traitement du formulaire. Le chemin d'accès à ce répertoire est obtenu en se basant sur la variable PHP super-globale `$_SERVER`. La case `$_SERVER['SCRIPT_FILENAME']` de ce tableau correspond au chemin d'accès sur le disque du script en cours d'exécution. La fonction PHP `dirname` permet d'obtenir la partie répertoires d'accès jusqu'au fichier passé en paramètre, sans le nom du script lui-même. Cela permet d'obtenir le chemin complet sur le disque sans le nom du script exécuté. Le répertoire de destination est donc la concaténation de ce chemin et de la chaîne */upload/pdf*. Une fois le chemin du répertoire défini, il reste à renommer le fichier et à le déplacer dans ce répertoire.

### Renommer et déplacer le fichier

La dernière étape consiste à déplacer le fichier du répertoire contenant les fichiers temporaires vers le répertoire dédié au stockage permanent des fichiers téléchargés. Elle est réalisée avec la fonction `deplaceFichier` du script *outils.php*. Cette fonction prend en paramètre le chemin du fichier à déplacer (`$_FILES['fichier']['tmp_name']`), le chemin du répertoire de destination ainsi que les informations pour renommer le fichier (préfixe et type de document). Le dernier argument est optionnel, il permet de stocker le chemin final d'accès au fichier et sera utilisé dans le Listing 5.

Avant de déplacer un fichier, il est important de vérifier que le répertoire de destination des fichiers téléchargés sur le serveur est bien accessible en écriture. En effet si les droits d'écriture ne sont pas donnés, les fichiers ne pourront être stockés et une erreur se produira. La fonction PHP `is_writable` permet de s'assurer que le répertoire peut être utilisé pour y stocker les fichiers envoyés. Si ce n'est pas le cas, un message d'erreur est retourné.

Le nom donné au fichier envoyé ne doit pas être celui d'origine. Il doit être choisi de manière à ce qu'il soit unique. Ceci permet d'éviter que le fichier soit écrasé lors d'un téléchargement ultérieur par un fichier auquel on aurait attribué le même nom. Une manière de procéder est de rajouter un *timestamp* au

préfixe reçu en argument, ceci est réalisé en utilisant la fonction PHP `time`. Le *timestamp* est le nombre de secondes écoulées depuis le 1er janvier 1970 jusqu'à l'instant d'exécution de la fonction. Il est également possible d'utiliser un nombre aléatoire généré par la fonction `rand`. Quelle que soit la méthode choisie, il est préférable de vérifier l'existence du fichier dans le répertoire de destination, avant de déplacer le fichier.

La fonction PHP `move_uploaded_file` permet d'effectuer le transfert de fichier sur le serveur. Elle reçoit en premier paramètre le chemin d'accès au fichier temporaire sur le disque (case `$_FILES['fichier']['tmp_name']`) et en second paramètre le chemin de destination (répertoire de destination et nouveau nom du fichier). Si un fichier de même nom existe dans le répertoire de destination, il sera écrasé par celui qui est déplacé. La fonction retourne un booléen, avec la valeur `true` en cas de succès et la valeur `false` sinon. Ce peut être le cas si le fichier, dont le déplacement est demandé, n'est pas issu d'un téléchargement correct par la méthode `POST`. En cas d'erreur, elle produit également un message d'alerte PHP (*warning*). Il est préférable de bloquer ce message grâce à l'opérateur de contrôle d'erreur `@` précédant le nom de la fonction, comme dans l'exemple du Listing 4.

Une fois le fichier déplacé, un message de réussite est affiché par le script du Listing 3.

## Sécurité

Le stockage et l'utilisation sur le serveur web de fichiers envoyés par un internaute posent des problèmes de sécurité. Cette section explique les implications de sécurité et les bonnes pratiques de sécurité à suivre pour implémenter un téléchargement de fichiers sécurisé.

### Ne pas se fier au type MIME

Lorsque le navigateur envoie le fichier, son type MIME est envoyé dans le champ *Content-Type* de l'en-tête HTTP. Cette information est stockée dans la clé `type`. L'instruction ci-dessous affiche le type MIME du fichier reçu, lorsque le nom donné au champ du formulaire est `fichier` :

```
echo $_FILES['fichier']['type'];
```

Le script pourrait vérifier que le type MIME est `image/png`, lorsque l'application n'autorise que la réception d'images au format PNG. Ceci permet d'indiquer à un utilisateur que le fichier reçu n'a pas le format attendu. Cette vérification n'est d'aucune utilité pour protéger l'application d'un utilisateur malveillant car elle est transmise par le client, et peut

**Listing 5.** *upload_images.php*

```
<?php

/**
 * Verifie la validite des images envoyees
 * @param $img array donnees sur les images telechargees (reference)
 * @return $erreur string
 */
function verifImages(&$img) {
    $erreur = '';
    // pour chaque image telechargee
    foreach($_FILES as $fichier => $details) {
        // verifier si image envoyee correctement
        $erreur_upload = verifUpload($fichier);
        // pas d'erreur de telechargement
        if (empty($erreur_upload)) {
            // recuperer des informations sur l'image (largeur, hauteur et type)
            if (list($img[$fichier]['width'], $img[$fichier]['height'], $type_img)
= @getimagesize($_FILES[$fichier]['tmp_name'])) {

                // recuperer l'extension correspondant au type de l'image
(avec le prefixe .)
                $img_ext = image_type_to_extension($type_img);
                // verifier le format d'image autorise (uniquement les png)
                if ($img_ext != '.png') {
                    $erreur .= "Champ $fichier : format d'image interdit
(autorise : png)<br>";
                }
            // la fonction getimagesize      n'a pas pu s'executer
            } else {

                $erreur .= "Champ $fichier : le fichier n'est pas une image
valide<br>";
            }
        } else {
         $erreur .= $erreur_upload.'<br>';
        }
    }
    return $erreur;
}

/**
 * Transfere les images telechargees du repertoire temporaire
 * vers un repertoire final sur le serveur
 * @param $img array donnees sur les images telechargees (reference)
 * @param $chemin_rep_dest string chemin sur le serveur du repertoire
d'images
 * @return $erreur string
 */
function sauveImages(&$img, $chemin_rep_dest) {
    $erreur = '';
    // pour chaque image telechargee
    foreach($_FILES as $fichier => $details) {
        $chemin_image = null;
        // sauver l'image dans le dossier final sur le serveur
        $erreur = deplaceFichier($_FILES[$fichier]['tmp_name'], $chemin_
rep_dest, $fichier, 'png', $chemin_image);
        // stocker le nom de l'image sur le serveur dans le tableau $img
        $img[$fichier]['nom_img'] = basename($chemin_image);
        // l'image a ete sauvee correctement
        if (empty($erreur)) {
         // creer une petite image a partir de l'image telechargee
            $erreur = creerPetiteImage($img[$fichier], $chemin_image);
        }
    }
    return $erreur;
}

/**
 * Cree une petite image a partir d'une image dont le chemin est fourni.
 * @param $img array donnees sur l'image d'origine
 * @param $chemin_image string chemin sur le serveur de l'image d'origine
 * @return $erreur string
 */
function creerPetiteImage($img, $chemin_image) {
    $erreur = '';
    $ressource = null;
    // creer une ressource image a partir de l'image d'origine
    if ($ressource = imagecreatefrompng($chemin_image)) {
        // reduire de 30%
        $nouvelle_largeur = floor($img['width'] * 0.3);
        $nouvelle_hauteur = floor($img['height'] * 0.3);
```

**Listing 5.** *upload_images.php – suite*

```
        // creer une ressource image vide avec les nouvelles dimensions
        if ($thumb = imagecreatetruecolor($nouvelle_largeur, $nouvelle_hauteur)) {
            // copier le contenu de l'image dans l'image vide
            if (imagecopyresized($thumb, $ressource, 0, 0, 0, 0, $nouvelle_largeur, $nouvelle_hauteur,
$img['width'], $img['height'])) {
                // sauver la petite image sur le disque
                $chemin_petite_img = dirname($chemin_image).'/petite_'.$img['nom_img'];
                if (!imagepng($thumb, $chemin_petite_img)) {
                    $erreur = "Impossible de sauver la petite image";
                }
            } else {
                $erreur = "Impossible de copier l'image";
            }
        } else {
            $erreur = "Impossible de copier l'image";
        }
    } else {
        $erreur = "Impossible de creer une petite image";
    }
    return $erreur;
}

// ========== SCRIPT PRINCIPAL ============
// inclusion de fonctions
require 'outils.php';

// initialisation du message d'erreur
$erreur = "";
// si le formulaire a ete poste
if (isset($_POST['envoi'])) {
    // les 2 fichiers sont dans le tableau $_FILES
    if (isset($_FILES['fichier_1']) && isset($_FILES['fichier_2'])) {
        // initialisation du tableau contenant des informations sur les images
        $img = array();
        // verifier que l'envoi de chaque fichier image s'est effectue correctement
        $erreur = verifImages($img);
        // pas d'erreur de telechargement sur le serveur
        if (empty($erreur)) {
            // definir le chemin du repertoire de stockage
            $chemin_rep_dest = dirname($_SERVER['SCRIPT_FILENAME']).'/upload/images';
            // transferer les donnees vers le repertoire final
            $erreur = sauveImages($img, $chemin_rep_dest);
        }
    } else {
        $erreur = "Tableau de donnees vide";
    }
}
?>

<!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=UTF-8">
        <title>Formulaire d'envoi d'image</title>
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <h1>Envoi de fichier image (format PNG)</h1>
        <?php
        // le formulaire a ete poste et les images correctement envoyees
        if (isset($_POST['envoi']) && empty($erreur)) {
            echo '<div class="reussite">Les images ont ete sauvees sur le disque : </div>';
            // afficher les images reduites crees a partir des images telechargees
            echo '<div><img src="upload/images/petite_'.$img['fichier_1']['nom_img'].'" alt="petite_image_1">';
            echo '<img src="upload/images/petite_'.$img['fichier_2']['nom_img'].'" alt="petite_image_2"></div>';
        // le formulaire n'a pas ete poste ou des erreurs sont survenues
        } else {
            // afficher les eventuels messages d'erreur
            echo "<div class='erreur'>$erreur</div>";
            // afficher ou reafficher le formulaire
            echo '<div>
                  <form action="'.$_SERVER['PHP_SELF'].'" method="post" enctype="multipart/form-data">
                      <div>Fichier 1 : <input type="file" name="fichier_1"></div>
                      <div>Fichier 2 : <input type="file" name="fichier_2"></div>
                      <div><input type="submit" name="envoi" value="Soumettre"></div>
                  </form>
                  </div>';
        }
        ?>
    </body>
</html>
```

# bulletin d'abonnement

phpsolutions

Merci de remplir ce bon de commande et de nous le retourner par fax : **+48 22 244 24 59**
ou par courrier :
**Software-Press Sp. z o.o. SK**
Bokserska 1, 02-682 Varsovie, Pologne
Tél. **0 975 180 358**
E-mail : **abo_fr@software.com.pl**

Prénom/Nom .....

Entreprise .....

Adresse .....

.....

Code postal .....

Ville .....

Téléphone .....

Fax .....

Je souhaite recevoir l'abonnement à partir du numéro .....

.....

En cadeau je souhaite recevoir .....

.....

E-mail (indispensable pour envoyer la facture) .....

.....

## PRIX D'ABONNEMENT À PHP SOLUTIONS : 35 €

Je règle par :

☐ **Carte bancaire n° CB**

⊔⊔⊔⊔ ⊔⊔⊔⊔ ⊔⊔⊔⊔ ⊔⊔⊔⊔

**code CVC/CVV** ⊔⊔⊔⊔

**expire le** _____ **date et signature obligatoires**

**type de carte** (MasterCard/Visa/Diners Club/Polcard/ICB)

☐ **Chèque (à envoyer à notre adresse postale)**
**À la ordre de :**
**Software-Press Sp z o.o. SK**

☐ **Virement bancaire :**
**Nom banque :**
**Société Générale Chasse/Rhône**
**banque guichet numéro de compte clé Rib**
**30003 01353 00028010183 90**
**IBAN : FR76 30003 01353 00028010183 90**
**Adresse Swift (Code BIC) : SOGEFRPP**

**Abonnez-vous et obtenez un cadeau !**

donc être facilement modifiée avant l'envoi au serveur.

### Attention aux commentaires d'images

Lorsque les fichiers téléchargés sont des images, il est possible de vérifier que c'est bien une image qui a été reçue en utilisant la fonction `getimagesize`. Cette fonction retourne un tableau contenant des informations sur l'image dont le nom est passé en argument.

```
$fic = $_FILES['fichier']['tmp_name'];
$infos = getimagesize($fic);
echo $infos['mime'];
```

Le script peut vérifier que le type MIME retourné par la fonction `getimagesize` fait partie de ceux autorisés. Ceci permet de s'assurer que c'est bien une image qui a été téléchargée. Cette vérification, bien que plus sûre que celle qui reposait sur l'information facilement modifiable donnée dans l'entête HTTP, n'est pas suffisante. En effet, la plupart des formats d'images permettent d'ajouter un commentaire à l'image. Le fichier contient donc bien une image, et un type MIME valide, mais peut contenir du code PHP.

Par exemple, si le commentaire GIF `<?php phpinfo(); ?>` est ajouté à une image GIF test.gif au moment de la sauvegarde, alors cette image a un type MIME `image/gif`, mais elle contient un code PHP qui permet d'afficher les réglages du fichier de configuration PHP sur le serveur. Si cette image est renommée *test.php* avant son envoi au serveur, alors le contenu du commentaire sera interprété par PHP lorsque le fichier *test.php* sera demandé. PHP ignore les données binaires de l'image, elles sont affichées en haut de la page de résultat, il exécute et affiche le résultat de la fonction `phpinfo`.

Vérifier le type MIME de l'image est utile mais insuffisant si d'autres précautions ne sont pas prises. Il faut renommer le fichier téléchargé et faire en sorte que PHP ne cherche pas à interpréter le contenu du fichier.

### Ne pas se fier aux extensions

L'extension contenue dans le nom du fichier reçu peut être analysée afin de n'autoriser que certains types de fichiers. Deux techniques de filtrage existent : liste blanche ou liste noire. L'approche par liste noire définit ce qui est interdit (*php, cgi, ...*), celle par liste blanche définit ce qui est autorisé (*gif, jpg, jpeg, png*). L'approche par liste noire est plus permissive que celle par liste blanche, il ne faut jamais l'utiliser.

La vérification de l'extension pose plusieurs problèmes. Premièrement, l'extension donnée au fichier peut être incorrecte. L'information est transmise par le client, elle peut être facilement modifiée. Deuxièmement, certaines extensions qui ne paraissent pas dangereuses peuvent l'être en fonction des réglages du serveur web. En effet, celui-ci peut être paramétré pour que PHP analyse, par exemple, les fichiers dont l'extension est `html`. La directive `AddType` du fichier de configuration Apache permet d'associer une ou plusieurs extensions à PHP, dans l'exemple ci-après les fichiers dont l'extension est `php` ou `js` sont interprétés par PHP :

```
AddType application/x-httpd-php .php .js
```

Si vous n'administrez pas le serveur web, vous ne pouvez pas définir la liste des extensions interprétées par PHP. Cette liste est susceptible d'évoluer à tout moment sur le serveur si l'administrateur effectue une mise à jour du fichier de configuration. L'analyse de l'extension du fichier ne garantit donc pas que le contenu du fichier soit du type attendu, ni qu'il ne sera pas interprété par PHP.

Enfin le dernier problème concerne les fichiers contenant plusieurs extensions. Lorsque c'est le cas, le serveur Apache ne tient pas toujours compte de leur ordre. Par exemple, le fichier test.php.aaa sera interprété par PHP. L'extension `aaa` n'étant pas connue, c'est celle `php` qui est retenue. Si un utilisateur malveillant envoie ce fichier sur le serveur et que le fichier est placé dans le répertoire `upload`, alors en entrant l'URL *http://nom_serveur/upload/test.php.aaa* il obtiendra l'exécution du script PHP sur le serveur. Si ce script contient un shell PHP, l'utilisateur malveillant pourrait exécuter des commandes sur le serveur. Il faut noter qu'une approche de validation par liste noire laisserait passer le fichier car l'extension `aaa` ne fait pas partie de celles qui sont interdites.

### Attention au .htaccess

Il est possible de définir les extensions associées à PHP dans un fichier *.htaccess* en utilisant la directive `AddType`. Le fichier *.htaccess* ci-après associe l'extension *png* à PHP :

```
AddType application/x-httpd-php .png
```

Si le fichier est placé dans le répertoire contenant les fichiers téléchargés, alors l'URL *http://nom_serveur/upload/image.png* provoquera l'analyse par PHP du fichier image.png. Si ce fichier contient du code PHP dans le commentaire d'images, alors il sera exécuté. La vérification de l'extension et du type du fichier avec `getimagesize` n'offrira aucune protection dans cet exemple. Il faut veiller à ce que l'internaute ne puisse pas télécharger un fichier *.htaccess* dans le répertoire stockant les fichiers téléchargés.

### Ne pas utiliser le nom d'origine

Le nom d'origine du fichier, contenu dans la clé `name` du tableau `$_FILES`, ne doit jamais être utilisé pour renommer le fichier téléchargé. Laisser la possibilité à l'internaute de définir le nom d'un fichier qui sera placé sur le serveur web est une faille de sécurité. Ceci pourrait lui permettre d'écraser un fichier placé dans le répertoire contenant les fichiers téléchargés. En effet, si un fichier de même nom existe dans le répertoire de destination, la fonction `move_uploaded_file` écrase le fichier. Il devient ainsi possible à un utilisateur malveillant de remplacer un fichier *.htaccess* protégeant le répertoire. De plus, ceci pourrait permettre de placer un script dans un répertoire du serveur web et de l'exécuter en entrant l'URL du script dans le navigateur. Par exemple, si le fichier téléchargé est placé dans un répertoire `upload` situé à la racine du site web, et que l'utilisateur envoie le fichier *test.php*, il peut l'exécuter en entrant l'URL *http://nom_serveur/upload/test.php*. Si test.php contient un shell PHP, alors des commandes peuvent être exécutées sur le serveur web par un utilisateur malveillant, par le biais de ce script.

La vérification du type MIME ou de l'extension avant l'utilisation du nom du fichier est insuffisante pour se protéger, il faut impérativement donner au fichier un nom différent de celui d'origine et vérifier que le répertoire de stockage ne contient pas un fichier de même nom.

### Interdire l'accès direct au répertoire

Les fichiers téléchargés devraient être situés dans un répertoire qui n'est pas directement accessible par un internaute. Ceci permet d'éviter qu'un utilisateur malveillant provoque l'exécution d'un fichier malicieux qu'il a préalablement téléchargé. Ceci peut être réalisé en stockant les fichiers dans un répertoire en dehors du répertoire contenant le site web, ou en les stockant dans un répertoire du site web dont un fichier *.htaccess* interdit l'accès. Il est important dans ce cas de placer le fichier *.htaccess* au dessus du répertoire contenant les fichiers téléchargés, afin d'éviter que ce fichier ne soit écrasé. Interdire l'accès direct au répertoire implique d'une part de faire le lien entre le fichier d'origine et le nom utilisé pour stocker les informations, ceci est réalisé le plus souvent en utilisant une base de données. Et d'autre part il faut écrire un script chargé de récupérer et d'envoyer les données. Ce script peut utiliser la fonction `readfile` pour lire le fichier dans le répertoire contenant les fichiers téléchargés. Le chemin du fichier transmis à la fonction `readfile` doit être nettoyé afin de ne pas permettre l'accès à n'importe quel fichier sur le disque dur par une faille de traversée de


**Figure 3.** *Affichage des images téléchargées.*

répertoires. Ceci peut être réalisé en utilisant la fonction `basename`.

Une autre approche consiste à placer le répertoire dans le site web et à définir dans un *.htaccess* les extensions autorisées. L'utilisateur peut alors accéder au fichier avec l'URL, mais seuls les fichiers dont les extensions sont autorisées peuvent être demandés. Le *.htaccess* ci-après n'autorise que l'accès direct aux fichiers dont l'extension est *gif* ou *png*. Les fichiers qui ont une double extension sont interdits.

```
deny from all
<Files ~ "^\w+\.(gif|png)$">
order deny,allow
allow from all
</Files>
```

Il faut veiller à ce que ce fichier *.htaccess* ne puisse pas être écrasé en le plaçant dans un répertoire au dessus de celui où sont stockés les fichiers, et il faut s'assurer qu'aucun fichier *.htaccess* ne puisse définir que l'extension *gif* ou *png* est associée à PHP.

### Manipuler les fichiers

Le déplacement du fichier, du répertoire temporaire de téléchargement vers le répertoire contenant tous les fichiers téléchargés, doit être réalisé avec la fonction `move_uploaded_file`. Cette fonction vérifie préalablement que le fichier est issu d'un téléchargement correct par la méthode `POST`.

Il est important de ne jamais réaliser une inclusion de fichier téléchargé dans un code PHP avec `include` ou `require`. Vérifier que le fichier n'existe pas dans le répertoire de téléchargement, avant d'effectuer le déplacement, évite qu'un téléchargement écrase un fichier existant.

### Limiter la taille autorisée

Il est important de fixer une taille limite de téléchargement raisonnable, afin d'éviter les attaques de déni de service par saturation de l'espace disque. Pour fixer cette taille, il faut utiliser exclusivement les directives du *php.ini* `post_max_size` et `upload_max_filesize`. La limitation de la taille ne doit pas passer par le champ caché de formulaire `MAX_FILE_SIZE`, en effet sa valeur peut être facilement modifiée par un internaute, avant l'envoi au serveur.

### Limiter le nombre de téléchargements

Le déni de service pourrait être obtenu par un grand nombre de téléchargements de fichiers inférieurs à la taille autorisée. Pour se protéger de ces attaques, il faut limiter le nombre de téléchargements autorisés pour un utilisateur. Il est préférable d'autoriser les envois de fichiers uniquement dans des applications où les utilisateurs s'authentifient. Si les téléchargements sont réalisés par un utilisateur authentifié, il suffit de stocker la taille des téléchargements et le nombre de téléchargements réalisés au cours d'une journée par cet utilisateur sur le serveur, et de ne plus enregistrer les fichiers envoyés lorsque les limites autorisées sont dépassées.

## Exemple

L'exemple du Listing 5 permet d'envoyer deux fichiers et de les sauvegarder sur le serveur. Le même script affiche le formulaire et effectue son traitement après envoi. Afin de simplifier l'exemple du Listing 5, ce dernier autorise uniquement l'envoi de fichiers d'images au format *png*. Il sera facile de l'adapter afin d'autoriser l'envoi et le traitement de fichiers d'images aux formats *gif* et *jpeg*.

Le script affiche le formulaire permettant d'envoyer deux fichiers lorsqu'il n'y a pas encore eu d'envoi vers le serveur, c'est à dire lorsque la variable `$_POST['envoi']` n'a pas encore été initialisée. Il ré-affiche le formulaire lorsqu'une erreur s'est produite, en dessous du message d'erreur correspondant. Une fois le formulaire posté au serveur et les images correctement sauvegardées, un message de réussite s'affiche à l'écran, ainsi qu'une image aux dimensions réduites de chacune des images téléchargées sur le serveur (Figure 3). Les fonctions spécifiques au traitement du formulaire de l'exemple sont définies dans le Listing 5. Ce dernier fait également une inclusion des fonctions outils du Listing 4.

### Vérifier la validité des images

Après envoi du formulaire, la fonction `verifImages` s'assure que chacune des images téléchargées a été correctement reçue et a un format valide. Dans l'exemple seules les images au format *png* sont autorisées.

La boucle `foreach` sur le tableau associatif `$_FILES` permet de parcourir les informations relatives à chacun des fichiers téléchargés. Dans l'exemple les deux champs du formulaire s'appellent `fichier_1` et `fichier_2`. Le tableau `$_FILES` a donc deux cases, la case `$_FILES['fichier_1']` et la case `$_FILES['fichier_2']`, chacune contenant un tableau d'informations lié au téléchargement du fichier correspondant. La boucle `foreach` permet de parcourir automatiquement toutes les cases du tableau `$_FILES`, il est donc possible de traiter un nombre illimité de champs de formulaire du type `file`.

La vérification du téléchargement correct de chacun des fichiers est effectuée par la fonction `verifUpload`, utilisée également dans le Listing 3 pour vérifier l'envoi d'un fichier PDF. En cas d'erreur, la fonction retourne un message non vide, ce message est stocké dans la variable `$erreur_upload`. Ce message est ajouté à la variable `$erreur` qui stocke l'ensemble des erreurs de téléchargement rencontrées au cours de l'exécution de la boucle. Cette liste d'erreurs, retournée par la fonction `verifImages` est affichée dans le navigateur et le formulaire est affiché à nouveau. Si lors de la vérification du téléchargement d'un fichier, la fonction `verifUpload` retourne une chaîne vide, alors les vérifications de type de l'image peuvent être réalisées en utilisant les fonctions PHP `getimagesize` et `image_type_to_extension`.

La fonction `getimagesize` prend en argument un chemin d'accès à un fichier image et retourne un tableau de sept cases comportant des informations sur cette image. Si le fichier passé en paramètre n'est pas une image valide ou bien qu'une erreur se produit, le booléen `false` est retourné et une alerte PHP est lancée. Pour empêcher le message d'alerte de s'afficher à l'écran, l'opérateur de contrôle d'erreur `@` est utilisé. Dans l'exemple, il est nécessaire de récupérer les dimensions de

l'image, utilisées par la suite pour créer des copies réduites de chaque fichier transféré, et le type de l'image, pour interdire les formats autres que PNG. Les trois premières cases du tableau retournées par la fonction `getimagesize` fournissent ces informations : la première case contient la largeur de l'image en pixels, la seconde case sa hauteur en pixels, et la troisième le type de l'image (une constante correspondant aux formats *png*, *gif*, ...). La fonction PHP `list` permet de placer le contenu des cases d'un tableau dans des variables passées en paramètre (une case par variable). La largeur et la hauteur de l'image sont placées dans deux cases du tableau `$img[$fichier]`, le type est placé dans la variable `$type_img`. Les autres données retournées par `getimagesize` sont inutiles dans l'exemple présent et ne sont donc pas conservées. Le tableau `$img` stocke les informations pour chaque image téléchargée, les clés de ce tableau associatif sont les noms des champs de formulaire de type `file`. A la fin de l'exécution de la fonction, le tableau contiendra donc les dimensions de chaque image téléchargée. La hauteur en pixels du second fichier téléchargé pourra être obtenue avec `$img['fichier_2']['height']`. Pour ne pas perdre les dimensions des images une fois la fonction exécutée, le tableau `$img` est passé en paramètre par référence (signe `&`). La fonction reçoit le tableau vide et le remplit au cours des itérations de la boucle. Une fois l'appel à la fonction `verifImages` réalisé, la variable `$img`, située dans la partie principale en dehors de la fonction, contient toutes les informations stockées au cours des itérations.

Une fois les informations sur l'image récupérées, la fonction `image_type_to_extension` permet de récupérer l'extension de l'image dans une chaîne de caractères. Elle prend en paramètre une constante correspondant à un type d'image. Par défaut, le caractère point est retourné en début de chaîne, avant les caractères de l'extension elle-même (exemple : *.gif*, *.png*). Le test suivant dans le code permet de s'assurer que le type réel de l'image est donc bien identique au type d'image autorisé dans cet exemple, c'est à dire le format *png* (*.png*). Si une erreur survient pour l'un ou l'autre des fichiers envoyés, ou pour les deux, un message relatif à l'erreur est retourné. Ce message d'erreur et le formulaire seront retournés au navigateur.

### Déplacer les images

Après vérification de la validité des deux images téléchargées sur le serveur, celles-ci sont déplacées une par une vers le répertoire de stockage définitif des images téléchargées. Le chemin du répertoire de destination est initialisé à partir du chemin d'accès au script courant sur le serveur, suivi du nom du répertoire des téléchargements, le répertoire `upload`, et du nom du répertoire de stockage des images nommé `images`.

La fonction `sauveImages` transfère les images. Elle prend en paramètre le tableau `$img` qui a été rempli dans la fonction `verifImages`. Une autre donnée sera ajoutée dans `$img` lors du traitement de l'image, la variable `$img` est donc encore une fois passée par référence. Le second paramètre passé à la fonction `sauveImages` est le chemin d'accès au répertoire de sauvegarde des images.

La fonction `deplaceFichier`, déjà vue dans le Listing 3, est utilisée pour effectuer le déplacement des images téléchargées vers le répertoire de destination. Cette fonction renomme l'image et définit le chemin définitif de l'image sur le serveur. Ce chemin est placé dans la variable `$chemin_image` qui est passée par référence à la fonction `deplaceFichier`. Afin de conserver le nom du fichier, la fonction `basename` est utilisée sur le chemin et le nouveau nom de l'image est stocké dans le tableau `$img` pour chacun des fichiers.

### Créer et afficher des images réduites

Une fois les images correctement déplacées, des images aux dimensions réduites sont créées à partir des fichiers téléchargés. Ces images aux dimensions plus petites seront ensuite affichées dans le navigateur, en dessous du message de réussite du téléchargement. La fonction `creerPetiteImage` crée et stocke les images réduites. Elle prend en paramètre le tableau d'information pour le fichier couramment traité (dimensions et noms de l'image) et le chemin de l'image d'origine.

La fonction PHP `imagecreatefrompng` crée une ressource pour l'image source au format *png* (celle téléchargée). Des fonctions PHP sont également disponibles pour les formats *gif* et *jpeg*. La fonction prend en paramètre le chemin de l'image sur le disque. Elle retourne le booléen `false` en cas d'erreur. Cette ressource va être utilisée pour obtenir une image réduite. Les dimensions de la nouvelle image sont calculées à partir des dimensions contenues dans le tableau `$img` (réduction de 30% en largeur et en hauteur).

L'image réduite est créée avec la fonction PHP `imagecreatetruecolor`. Elle prend en paramètre les dimensions de l'image à créer (largeur et hauteur en pixels). La ressource retournée est une image pour l'instant vide. L'image source est copiée, redimensionnée et placée dans la nouvelle image en utilisant la fonction `imagecopyresized`. Cette dernière prend en paramètres dans l'ordre : la ressource de l'image de destination, la ressource de l'image source, quatre paramètres de coordonnées (positions en x et y du point de destination et du point source), la nouvelle largeur de l'image à créer, la nouvelle hauteur, la largeur de l'image source et enfin la hauteur de l'image source. La fonction `imagepng` sauve la nouvelle image créée sur le disque, au format *png*. Elle prend en paramètre la ressource d'image et le chemin de sauvegarde de l'image. Le nom de l'image est celui de l'image d'origine, préfixé par le mot `petite`. Les images réduites sont sauvées dans le même répertoire que les images d'origine. Elles sont affichées à l'issue de l'exécution du script si aucune erreur n'est survenue.

## Conclusion

Vous savez à présent envoyer des fichiers du client vers le serveur, configurer le serveur pour qu'il accepte les téléchargements, et mettre en œuvre une stratégie de protection de vos applications. Une fois le fichier stocké sur le serveur, vous pouvez le manipuler en utilisant les connaissances acquises dans les précédents articles de ce magazine, décrivant la manipulation de fichiers et de répertoires. Dans le prochain numéro vous apprendrez à réaliser des téléchargements de fichiers distants à partir d'un script PHP.

### Sur Internet

- *http://www.php.net/manual/fr/features.file-upload.php* – Manuel PHP,
- *http://www.la-grange.net/w3c/html4.01/interact/forms.html* – Traduction de la recommandation officielle du W3C sur les formulaires HTML.

**CÉCILE ODERO, MAGALI CONTENSIN**

*Cécile Odero est spécialisée dans la conception et le développement d'applications web en PHP. Elle travaille au CNRS comme ingénieur en développement et déploiement d'applications.*
*Contact : cecile.odero@gmail.com*
*Magali Contensin, auteur du livre Bases de données et Internet avec PHP et MySQL, est chef de projet en développement d'applications au CNRS. Elle enseigne depuis plus de dix ans le développement d'applications web à l'Université.*
*Contact : http://magali.contensin.online.fr*

b-pack™
advanced business software

RECRUTE

web 2.0

SGBD

oracle
mysql
mssql

Ajax

HTML
CSS

Cloud

windows
linux

PHP

XML

Apache
IIS

REST*

Web
SSO

(*) SOA, Interfaces, webservices, SOAP

**Editeur de solutions d'optimisation des processus et de gestion des achats.**

b-pack développe les applications des grandes entreprises du secteur privée et public pour la gestion des processus stratégiques finances et achats.

Vous êtes passionné par les nouvelles technologies ? Rejoignez-nous, et participez à la création de la nouvelle génération de logiciels d'entreprises.

Retrouvez la liste des postes ouverts sur:
http://www.b-pack.fr/societe/carriere


Paris

Immeuble Technologies
84-88 bd de la M° Marchand
92400 Courbevoie

Aix-en-provence

220 Rue Denis Papin
Héliosis Batiment A
13857 Aix-en-Provence

Atlanta

303 Perimeter Center North
Suite 300
Atlanta, GA 30346

contact em@il : job@b-pack.com

# Dans le prochain numéro de *phpsolutions*

Le périodique *phpsolutions* est publié par
Software Press Sp. z o.o. SK
Bokserska 1, 02-682 Varsovie, Pologne
Tél. 0975180358, Fax. +48 22 244 24 59
www.phpsolmag.org

**Président de Software Press Sp. z o.o. SK :** Paweł Marciniak

**Directrice de la publication :** Ewa Łozowicka

Imprimerie, photogravure :
ArtDruk *www.artdruk.com*
Imprimé en Pologne/Printed in Poland

Abonnement (France métropolitaine, DOM/TOM) :
1 an (soit 6 numéros) 35 €

**Dépôt légal :** à parution
ISSN : 1731-4593

**Distribution :** MLP
Parc d'activités de Chesnes, 55 bd de la Noirée
BP 59 F - 38291 SAINT-QUENTIN-FALLAVIER CEDEX


**Rédacteur en chef :** Łukasz Bartoszewicz
**Assistante da la rédaction :** Martyna Stobiecka
**Préparation du CD :** Andrzej Kuca
**Maquette :** Agnieszka Marchocka
**Couverture :** Agnieszka Marchocka

**DTP :** Sławomir Sobczyk *Studio2W@gmail.com*

**Composition :** Sławomir Sobczyk
**Correction :** Aymeric Lagier

**Bêta-testeurs :** Fabrice Gyre, Brice Favre, Valérie Viel, Aymeric Lagier, Christophe Milhau, Alain Ribault, Stéphane Guedon, Eric Boulet, Mickael Puyfages, Christian Hernoux, Isabelle Lupi, Antoine Beluze, Timotée Neullas, Yann Faure, Adrien Mogenet, Jean-François Montgaillard, Turmeau Nicolas, Jonathan Marois, Wilfried Ceron, Wajih Letaief, François Van de Weerdt, Jeremy Rafflin, Eric Vincent.

Les personnes intéressées par la coopération sont priées de nous contacter :
editor@phpsolmag.org

**Abonnement :** *abo_fr@software.com.pl*
**Fabrication :** Andrzej Kuca *andrzej.kuca@software.com.pl*
**Diffusion :** Ilona Lepieszka *ilona.lepieszka@software.com.pl*
**Publicité :** *publicite@software.com.pl*



**En vente dès mai !**

## POUR LES DÉBUTANTS

### Télécharger un fichier depuis un script PHP

Vous avez appris, dans les précédents articles de cette série sur les fichiers, comment lire et écrire dans des fichiers situés sur le disque dur du serveur web et comment manipuler les fichiers reçus par le biais d'un formulaire HTML. PHP permet également de télécharger des fichiers situés sur des serveurs distants, afin de les stocker, les analyser et intègre leurs données dans le résultat envoyé au navigateur. Il est ainsi possible de récupérer automatiquement des pages web externes, des images, ... Cet article présente trois méthodes alternatives de téléchargement de fichiers distants, depuis un script PHP.

## FICHE TECHNIQUE

### Gestion du multilingusime dans le développement web

Le multilinguisme des sites web est devenu un sujet incontournable. Or, cette notion peut s'avérer bien plus complexe qu'il n'y paraît. Nous allons voir dans un prochain article des exemples et techniques d'implémentation de fonctionnalités liées au multilinguisme.

## SÉCURITÉ

### Tester et nettoyer les données grâce à FilterPHP

La sécurité dans PHP est un point qui est très important dans une application. Dans cet article, nous allons voir que la classe FilterPHP est plutôt bien adaptée aux problèmes courants. FilterPHP a le mérite d'être vraiment simple à comprendre, à implémenter et propose de nombreux filtres pour contrôler et assainir des variables. Vous verrez comment créer une classe vous permettant de sécuriser vos données entrantes et sortantes.

## OUTILS

### Découvrez CuteFlow

De nos jours, des pressions internes et externes nous imposent d'évoluer constamment. C'est de toute évidence le moyen le plus efficace pour stimuler le dynamisme d'une entreprise. Pour cette raison, un système de flux de travail, permettant la circulation et la validation des tâches à accomplir. Dans cet article, nous vous présentons tout d'abord les fonctionnalités du logiciel, puis, son installation, ensuite, sa configuration, par la suite nous expliquons la circulation et la validation des documents.

## Et de nombreuses autres articles à ne pas manquer !

**La rédaction se réserve le droit de modifier le contenu de la revue.**


WEB Gazelle
Génialement Simple !
Cet homme
essaie de mettre son
site Internet à jour
Cet homme
a mis son site
Internet à jour avec
WebGazelle CMS 2.0
WebGazelle CMS 2.0 rencontre l'AJAX
www.webgazelle.net
Webgazelle.net, une marque de
COGNIX
systems
WebGazelle(R) CMS est une solution de gestion de contenu pour site Internet