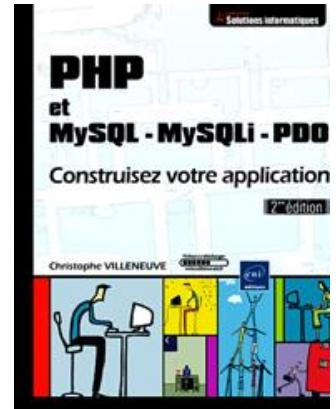


# PHP et MySQL - MySQLi - PDO

Construisez votre application [2ième édition]

Christophe VILLENEUVE



## Résumé

Ce livre sur **PHP** est destiné à toute personne qui désire **se lancer dans le développement web** avec ce langage. Il détaille **pas à pas** le développement d'une application de gestion d'un carnet d'adresses sur Internet. La conception de cette application prend en compte différents formats de bases de données (**MySQL**, **MySQLi** et **PDO**) et différentes versions de **PHP**.

Dans un premier temps, l'auteur choisit de décrire les **principales fonctions de PHP** en prenant des exemples facilement compréhensibles. Il décrit ensuite **pas à pas** les différentes **étapes du développement** en s'aidant des exemples de la première partie (accès sécurisés, gestion du carnet d'adresses, gestion des mots de passe, gestion des administrateurs, affichage et exportation des données...). Un chapitre complémentaire détaille des **notions plus avancées** comme les contrôles de sécurité, le suivi de la navigation des visiteurs...

Cette **nouvelle édition** de l'ouvrage met l'accent sur deux points : le premier concerne internet (**sécurité captcha**, transmissions des données entre sites...), l'autre point concerne la **programmation objet**, de sa découverte à l'utilisation des **espaces de noms** (namespace). L'application étudiée est déclinée en trois versions, avec une version supplémentaire en mode objet, entièrement téléchargeables sur cette page.

## Les chapitres du livre :

Avant-propos - L'environnement de développement - La préparation du développement - La saisie et l'affichage - Les résultats - Fonctionnalités supplémentaires - Programmation orientée objet - Annexe

## L'auteur

**Christophe Villeneuve** se passionne pour l'informatique depuis de nombreuses années. Autodidacte, son parcours professionnel a toujours évolué dans le monde du développement et du conseil. Son intérêt pour PHP a fait de lui un membre actif de l'AFUP (Association Française des Utilisateurs de PHP) et avec ce livre, il met en exergue son expérience pour guider le lecteur de façon efficace dans la construction d'applications PHP.

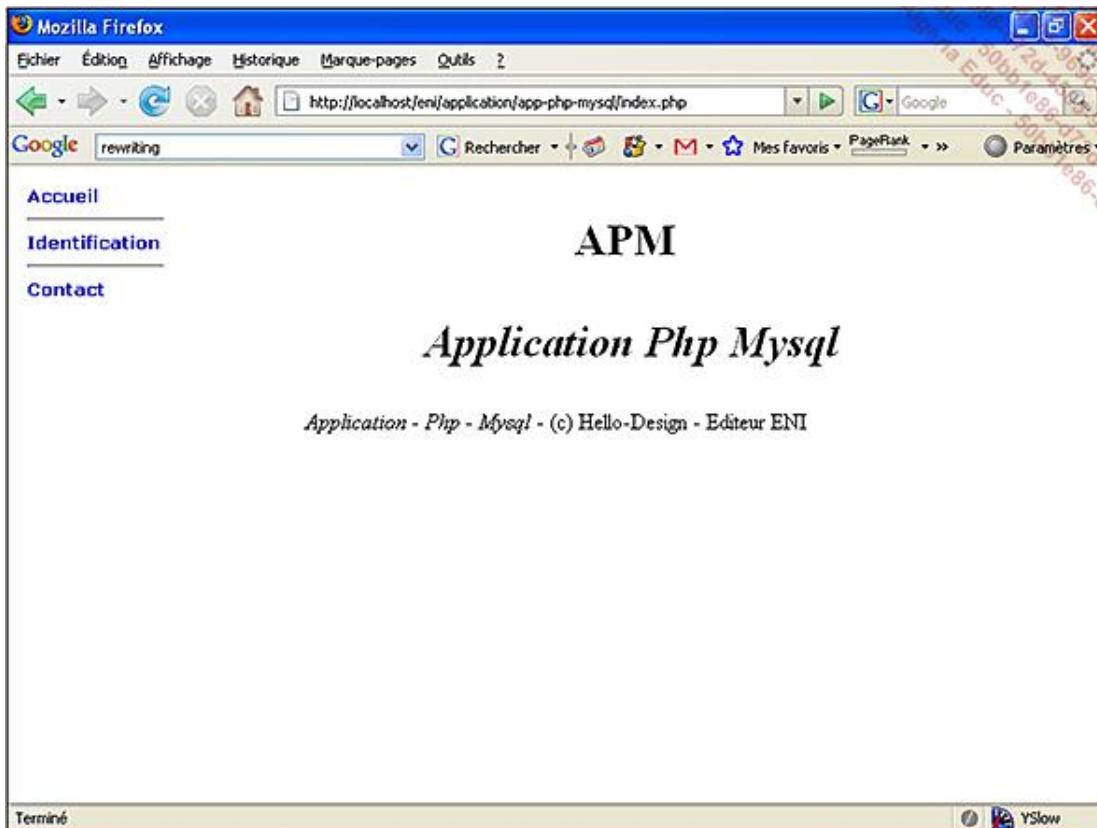
*Ce livre numérique a été conçu et est diffusé dans le respect des droits d'auteur. Toutes les marques citées ont été déposées par leur éditeur respectif. La loi du 11 Mars 1957 n'autorisant aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective", et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, "toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayant cause, est illicite" (alinéa 1er de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal. Copyright Editions ENI*

# Avant-propos

Ce livre est destiné à toutes les personnes qui désirent se lancer dans la programmation en environnement Web (Internet) ou qui développent sur d'autres langages évolués que PHP et veulent développer des applications dynamiques et évolutives pour Internet.

## 1. Description de l'application à réaliser

Tout au long des pages de ce livre, nous allons réaliser une application qui gère un carnet d'adresses. Le carnet d'adresses sera accessible sur Internet pour un ou plusieurs utilisateurs disposant des droits suffisants : le carnet d'adresses sera donc partagé.



Pour chaque contact du carnet d'adresses, outre les informations classiques (nom, prénom, adresse, téléphone, photo...), le titulaire d'un compte autorisé pourra ajouter des informations supplémentaires comme les comptes de tchat, les sites internet préférés... Chaque information sera liée à une rubrique. Chaque titulaire d'un compte autorisé pourra personnaliser la liste des rubriques.

Il sera aussi possible de localiser sur une carte la position des contacts.

Chaque titulaire d'un compte pourra, avec cette application, transmettre les données enregistrées vers d'autres applications : les exportations proposées correspondent aux formats CSV, PDF, XML.

## 2. Différents objectifs de réalisation

Pour réaliser cette application, différents objectifs doivent être atteints :

- Il faut concevoir une structure modulaire, permettant ainsi à n'importe quel utilisateur de pouvoir créer un enregistrement dans une base de données.
- Une fois la structure créée, elle doit permettre la recherche d'informations existantes, l'ajout d'autres données, leur modification, leur effacement à partir d'un masque de saisie et leur récupération.

Login	Niveau	Email	Date creation	dernier passage	Edit
test	user	webmaster@hello-design.fr	25-06-2007	14-01-2008	<a href="#">Cliquez ici</a>
admin	admin	livre@hello-design.fr	01-08-2007	17-01-2008	<a href="#">Cliquez ici</a>

- Enfin, l'application doit pouvoir permettre de récupérer les informations de la base de données pour les afficher à l'écran, les imprimer ou les exporter pour une autre utilisation.

### 3. Préparation et déroulement du développement

Pour réaliser cette application, le livre est organisé ainsi :

## **Chapitre 1 : Environnement de développement**

Ce chapitre vous donnera les bases du PHP (notamment comment établir une connexion avec une base de données). Il présentera les outils nécessaires à la réalisation du projet sur un serveur Apache.

## **Chapitre 2 : Préparation du développement**

Avant de commencer le projet, nous préparerons le travail en découvrant un certain nombre de possibilités offertes avec le langage PHP. Nous aurons la possibilité d'en ressortir une ou plusieurs notions pouvant nous permettre de mener à bien notre projet.

## **Chapitre 3 : Saisie et affichage**

Cette partie concerne la réalisation de l'application proprement dite. Grâce aux différentes préparations que nous avons pu réaliser, nous allons pouvoir réutiliser :

- les accès sécurisés pour définir les droits des utilisateurs,
- la gestion du carnet d'adresses,
- les paramétrages du carnet d'adresses,
- la gestion des mots de passe,
- la gestion des administrateurs.

## **Chapitre 4 : Les résultats**

Sur les éléments saisis dans notre application, nous devons pouvoir effectuer des recherches et des tris. Nous devons aussi permettre la communication avec d'autres applications autorisant ainsi des traitements différents.

## **Chapitre 5 : Fonctionnalités supplémentaires**

Une fois l'application réalisée nous ajouterons des protections supplémentaires pour aider le serveur et faciliter les contrôles. Nous localiserons les contacts sur des cartes et nous visualiserons la navigation des visiteurs.

La programmation orientée objet est une autre méthode de programmation. Nous verrons l'ensemble des fonctions standards et comment les mettre en place dans notre cas pratique.

## **4. Les versions utilisées**

Dans cet ouvrage, nous vous proposons la même application, mais avec trois approches différentes pour vous permettre d'étudier différentes bases de données avec le langage PHP dans ses différentes versions.

Certains formats de base de données sont des standards, mais d'autres sont de plus en plus connus et ce livre vous permettra de mieux les connaître et de les utiliser :

- **PHP et MySQLi**

Version pour PHP 5.xx et supérieur et MySQL 5 et supérieur.

- **PHP et MySQL**

Version pour PHP 4.3.x ; 5.xx et supérieur et MySQL 4 ; 5 et supérieur.

- **PHP et PDO**

Version pour PHP 5.2.x et supérieur et MySQL 5 et supérieur.

- **PHP et POO**

Version pour PHP 5.xx et supérieur et MySQL 4 ; 5 et supérieur.

## 5. L'environnement de développement

Quel que soit votre système d'exploitation, il est nécessaire d'installer un environnement Apache/PHP/phpMyAdmin. Par exemple : Wampserver, Mamp ou Xampp.

Vous devez créer une base de données sous le nom de "ouvrage" et importer le fichier se trouvant dans le dossier SQL.

## Télécharger les exemples du livre

Afin de pouvoir réaliser l'ensemble des manipulations de cet ouvrage, vous devez télécharger les fichiers d'exemple fournis sur le site des Editions ENI.

Pour cela, accédez au site des Editions ENI : [www.editions-eni.fr](http://www.editions-eni.fr) et dans la zone **Moteur de recherche**, saisissez la référence ENI du livre **SO25PHP** et validez. Cliquez sur le titre du livre puis sur le lien de téléchargement.

Le dossier qui vous est fourni comprend deux sous-dossiers (application et source) organisés comme ceci :

- application
  - app-php-mysql
  - app-php-mysqli
  - app-php-pdo
  - app-poo-mysqli
  - sql
- ouvrage
  - sql
  - Différents dossiers du livre

Le dossier **application** comprend les scripts pour initialiser la base de données SQL (que vous devez installer dans phpMyAdmin) et quatre dossiers contenant chacun une application avec une base de données différente. Ces trois applications sont complètement indépendantes l'une de l'autre.

Le dossier **ouvrage** comprend la base de données SQL et tous les dossiers qui seront étudiés dans le livre.

Tout au long de cet ouvrage, des exemples sont réalisés pour expliquer comment se servir des fonctions PHP. Nous ne pouvons pas utiliser l'ensemble des fonctions qui existent car elles sont trop nombreuses, mais nous pouvons étudier les principales.

Le code source se trouve dans le fichier source.rar.

Chaque fonction et exemple se trouvent regroupés dans un dossier séparé "source" avec exactement le nom du thème les concernant.

Dans chaque application d'exemple (app-php-xxx), nous découvrons la décomposition suivante :

- admin : ce dossier contient le fichier de connexion vers la base de données. Il est mis à part car c'est le plus important. Il est nécessaire de le protéger avec les fichiers .htaccess et .htpasswd.
- dl : ce dossier est destiné à stocker les fichiers qui seront téléchargés.
- images : nous stockons dans ce dossier les images et photos qui sont utilisées dans le carnet d'adresses personnel de chaque personne qui possède un compte.
- include : ce dossier contient les fichiers qui sont appelés lors de la navigation. Ils sont composés de :

Config : contient les configurations en générales

Fct : fichier comprenant les fonctions générales

Fct\_captcha.inc.php : fichier comprenant la réalisation du captcha

Fct\_debug : fichier comprenant les outils de débbugages

Fct\_log : fichier pour les logs

Fct\_upload : fichier pour les envois de fichiers (upload)

Fpdf : fichier pour créer des fichiers PDF avec le dossier FONT

GoogleMapAPI : la classe Google Map

- log : ce dossier mémorise les logs des différents passages des visiteurs.

Les autres fichiers se trouvent à la racine du dossier. Ils sont utilisés pour notre application. Chaque fichier possède une découpe spécifique au menu :

admin-xxx : menu administrateur

carnet-xxx : menu votre carnet

compte : information sur le compte

contact : page de contact

deconnection : quitter le compte

download : fichier utilisé pour forcer les téléchargements

export-xxx : menu exportation

footer : pied des pages internet

head : en-tête des pages internet

identification : page pour saisir les codes d'accès pour accéder à la partie privée

login : gestion et vérifications des codes d'accès

pass-xxx : changer le mot de passe ou recevoir un autre mot de passe

rubrique : menu Rubrique

Les lignes admin-xxx et carnet-xxx se décomposent de la façon suivante :

Add : ajouter

View : visualiser

View-edit : éditer la ligne que nous avons visualisée

Après l'installation d'un des trois environnements, vous devez lancer votre navigateur et saisir l'une des adresses suivantes :

- <http://localhost/votreDossier/application/app-php-mysql>
- <http://localhost/votreDossier/application/app-php-mysqli>
- <http://localhost/votreDossier/application/app-php-pdo>
- <http://localhost/votreDossier/application/app-poo-mysqli>

Si l'application affiche des messages d'erreurs, vous devez vous reporter à l'ouvrage pour effectuer les modifications nécessaires.

Concernant les accès à la partie privée, nous vous rappelons les codes d'accès :

Pour la partie Administrateur :

Login : admin

Password : Admin

Pour un compte utilisateur :

Login : test

Password : test

Ces codes d'accès peuvent être modifiés comme vous le souhaitez.



## Introduction

Avant de commencer un développement, nous devons connaître les bases du langage intranet/internet pour réaliser correctement une page HTML et par conséquent une application dynamique en PHP.

## Page fixe/page dynamique

Quel que soit le micro-ordinateur que vous possédez, un système d'exploitation (OS) est présent dessus pour vous permettre de réaliser de nombreuses tâches sous la forme de logiciels ou d'applications. Grâce au micro-ordinateur nous pouvons aussi accéder à l'univers d'Internet et naviguer sur le Web avec un navigateur (browser).

Les navigateurs les plus répandus actuellement sont : Mozilla Firefox, Safari, Konqueror, Opéra, Internet Explorer.

Cette plate-forme Internet que nous utilisons régulièrement évolue tous les jours avec de nombreux moyens et de nombreuses techniques, et par le biais d'informations et/ou d'actualités, de site personnels ou de sites professionnels.

Cette masse d'informations est générée par toutes sortes de sites Internet, qui sont représentés sous la forme de sites vitrines ou de sites dynamiques.

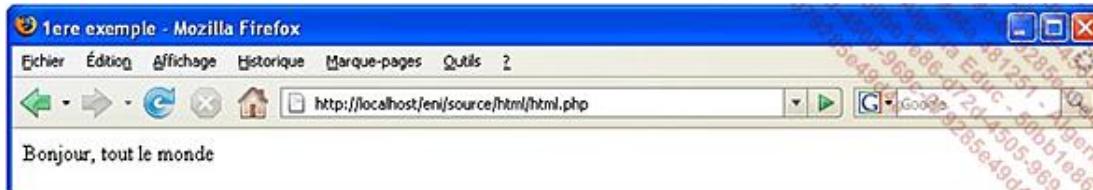
Le site vitrine est un site dont les pages sont fixes et souvent développées en HTML.

Il est possible de réaliser une page fixe au format HTML comme ceci :

```
html/html.php
```

```
<html>
<head>
<title>1ere exemple</title>
</head>
<body>
<?php
echo "Bonjour, tout le monde";
?>
</body>
</html>
```

Pour obtenir le résultat suivant :



Le site dynamique permet d'afficher certaines informations sur une partie ou la totalité d'une page de navigation.

Nous allons effectuer un petit exemple en affichant un message "Bonjour tout le monde" avec une partie dynamique permettant :

- de proposer un lien pour accéder à la suite ;
- de recommencer la manipulation.

Voici le premier écran lorsque nous arrivons sur la page Internet :



Les lignes du programme se décomposent comme ceci :

```
html/html2.php
```

```
<html>
<head>
<title>2eme exemple</title>
```

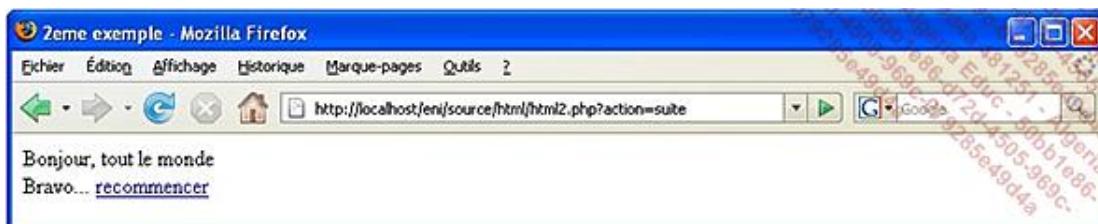
```

</head>
<body>
<?php
echo "Bonjour, tout le monde<br>";

if (isset($_GET['action']) && $_GET['action']=="suite")
{
    echo "Bravo... ";
    echo "<a href=html2.php?action=recommence>recommencer</a>";
}
if ( (isset($_GET['action']) && $_GET['action']=="recommence")
    || empty($_GET['action']))
{
    echo "Etape 1... ";
    echo "<a href=html2.php?action=suite>Suite</a>";
}
?>
</body>
</html>

```

Lorsque nous cliquons sur le lien **Suite**, nous obtenons l'écran suivant :



► L'autre possibilité d'un site dynamique est d'afficher des informations venant d'une base de données. Ce point sera étudié dans ce chapitre Présentation d'une base de données (ou BDD).

# Le langage PHP

Le langage PHP est un langage de script exécuté côté serveur, c'est-à-dire que le navigateur Internet n'affiche que le résultat de l'exécution du script dans la page.

Le but du langage PHP est de permettre de réaliser des pages Internet sous la forme de pages dynamiques.

Il permet avant tout de :

- valider les formulaires ;
- éviter de répéter les lignes de contrôle ;
- programmer en ligne de commande ;
- écrire des applications clientes graphiques.

Par ailleurs, avec PHP il est possible de réaliser des applications plus complexes que nous allons seulement énumérer car de nombreux kits existent déjà : forum, blog, livre d'or, newsletter, compteur de visiteurs, chat, système d'actualités, jeux de stratégies et réflexions, etc.

Le langage ne se limite pas à la production de pages HTML. Il est possible de générer à la volée des images graphiques, des fichiers PDF, des animations Flash...

 Le langage PHP permet aussi de communiquer avec tous les autres langages (XML, Java, Ajax...).

## 1. Afficher une page

Pour accéder à une page Internet, nous utilisons un navigateur. Lorsque nous affichons un site Internet, nous nous connectons en fait à un serveur qui utilise un langage, par exemple le langage PHP.

Pour afficher la page d'un site Internet, nous saisissons directement dans la barre de navigation du navigateur l'adresse du site Internet. Lorsque celle-ci a été saisie, elle va nous envoyer directement sur le serveur qui héberge le site Internet. Par exemple : <http://www.editions-eni.fr>

Lorsque nous développons un site Internet, il est nécessaire de disposer d'un serveur sur notre ordinateur permettant ainsi de simuler la connexion. Nous saisissons alors dans notre barre de navigation une adresse de ce type : <http://127.0.0.1/> ou <http://localhost/>

## 2. Détails de l'utilisation

L'utilisation du langage PHP comporte certaines règles à respecter. Les balises (ou tags) sont représentées ainsi :

```
<?php  
...  
?>
```

Par exemple pour afficher le message : "Bonjour, tout le monde", voici la page Internet au format HTML :

debut/exemple.htm

```
<html>  
<head>  
</head>  
<body>  
Bonjour, tout le monde  
</body>  
</html>
```

Pour obtenir ceci :



Avec le langage PHP, voici le même exemple que précédemment :

debut/exemple.php

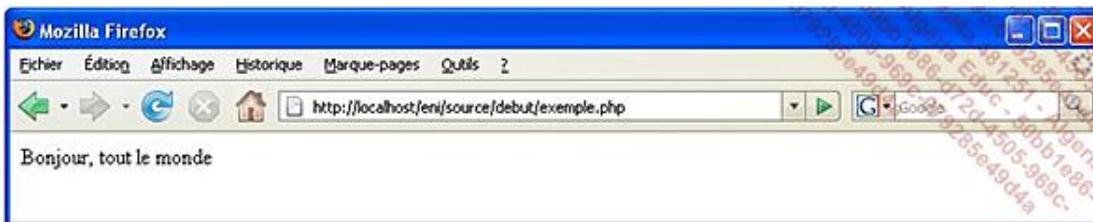
```
<html>
<head>
</head>
<body>
<?php
echo "Bonjour, tout le monde";
?>
</body>
</html>
```

La fonction principale utilisée par PHP pour afficher un message correspond à la fonction **Echo** :

### **Echo**

Affiche une chaîne de caractères.

Pour obtenir ceci :



 Bien sûr, il existe d'autres fonctions pour afficher un texte que nous ne verrons pas dans ce livre. Signalons par exemple PRINT, PRINT\_R.

Une autre fonction qui sera très utilisée dans l'ouvrage, il s'agit de :

### **Die**

Alias de la fonction exit.

### **Exit**

Termine le script courant en affichant un message si nécessaire.

À la fin de chaque ligne en PHP, nous mettons le symbole ";" (point virgule). Ce symbole est tiré du langage de programmation C. Lorsque la fonction est exécutée, et avant d'interpréter la fonction suivante, nous signalons la fin d'exécution pour la ligne en cours.

Sans ce symbole à la fin de chaque ligne, nous rencontrerions quelques soucis de fonctionnement.

## **3. Les extensions des fichiers**

Les fichiers qui sont sauvegardés sur l'espace disque possèdent l'extension PHP pour être traités par l'hébergeur.

Il est toujours possible d'utiliser les extensions .php3, .php4, .php5 mais cela est déconseillé car c'est une information qui peut être utile pour les personnes qui viendraient visiter votre site Internet.

Si nous parlons extension des fichiers, il est possible d'effectuer une distinction entre les fichiers principaux et les

sous-fichiers.

Les sous-fichiers sont des fichiers comprenant des routines, des classes, des fonctions ou toutes sortes d'informations importantes et qui peuvent être utilisés par plusieurs fichiers principaux. Pour repérer facilement tous les fichiers et les identifier, nous pouvons utiliser l'extension **.inc.php**.

L'ajout **.inc** correspond au principe **include**.

Grâce à cette extension supplémentaire, nous pouvons repérer plus facilement les fichiers principaux et les fichiers qui servent d'inclusions.

# Présentation d'une base de données (ou BDD)

## 1. Introduction

La base de données (BDD) est un système permettant d'enregistrer les données venant de différentes sources (formulaires, informations diverses...).

Nous allons communiquer avec une base de données en utilisant un langage dynamique, le langage PHP.

Un des formats les plus répandus et le plus connu est actuellement le format SQL et plus précisément le format MySQL. Bien sûr, il existe d'autres types de bases de données : PostgreSQL, DB2, Oracle, MS Access (ODBC).

Depuis la version 4.1 de PHP, est apparu le format MySQLi. C'est une extension de MySQL. Ce format est de plus en plus utilisé et fortement conseillé car il risque de devenir un standard. Il existe des outils pour convertir le format MySQL vers le format MySQLi.

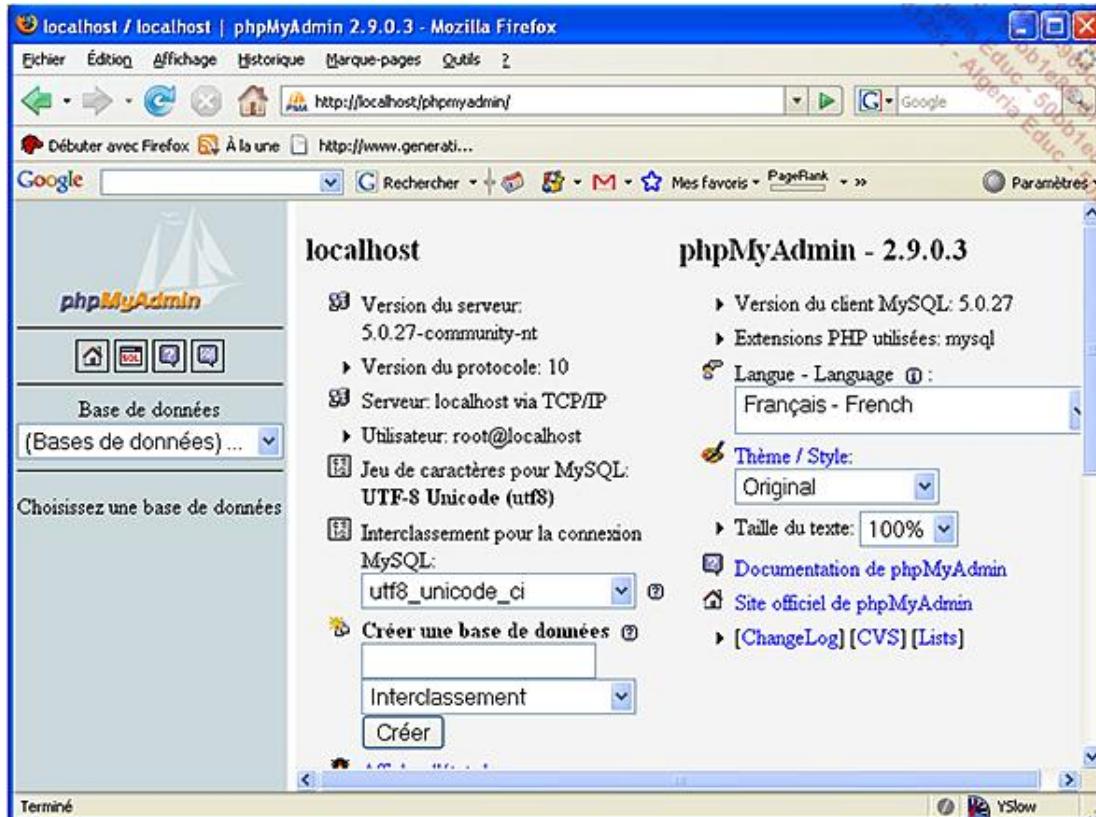
Avec les versions actuelles de PHP, un nouveau format est apparu : il s'agit du format PDO qui permet d'utiliser une base de données comme un objet. Il sera étudié un peu plus loin dans cet ouvrage.

Le principe de fonctionnement entre un langage de programmation, comme ici le langage PHP, et une base de données est le suivant :

- le langage PHP effectue les actions que nous lui demandons ;
- celui-ci va interroger la base de données pour mémoriser les informations ;
- la base de données effectue le travail ;
- quand la base de données a terminé, elle retourne une information signalant que tout s'est correctement déroulé.

Dans cet ouvrage, nous allons utiliser la base de données au format SQL et plus précisément MySQL.

Pour communiquer avec la base de données nous utiliserons phpMyAdmin comme ci-après :



La base de données se présente sous une forme structurée (nom de la base de données, nom des tables, nom des

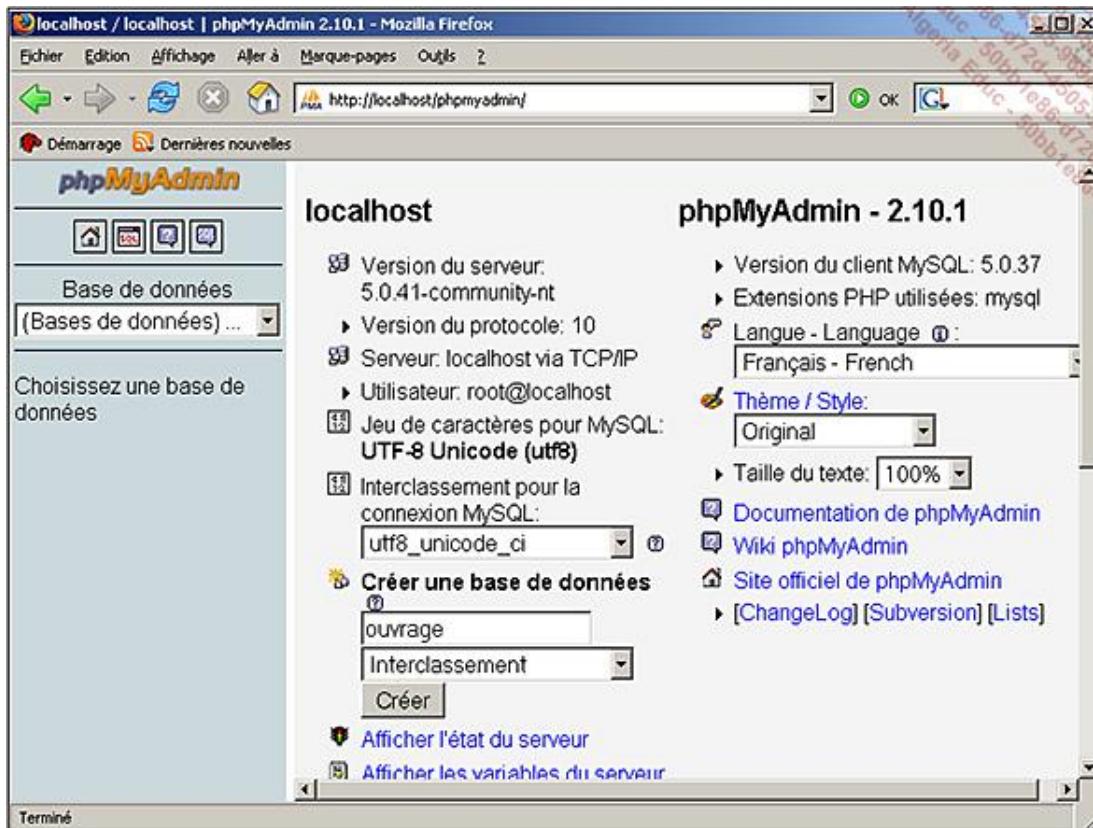
champs pour chaque table). Les données remplissent les champs des tables.

C'est ainsi qu'est utilisée la base de données MySQL, à partir de phpMyAdmin.

## 2. Créer une base de données

Quand vous choisissez un hébergeur gratuit ou payant, celui-ci vous attribue d'office un nom de base de données, souvent c'est le nom du compte mais certains hébergeurs peuvent vous en imposer un de leur choix.

Par contre sur votre ordinateur, c'est-à-dire en local, vous pouvez choisir le nom de la base de données. Ici, nous allons utiliser comme nom **ouvrage**, comme ci-après :



 Tout au long de ce livre, nous allons étudier les différentes fonctions et points importants du langage PHP. Nous effectuerons les tests avec la base de données qui utilise le nom **ouvrage**.

Après avoir créé la base de données, nous avons deux possibilités :

- Importer le fichier SQL qui décrit la structure de la ou des tables.
- Créer une ou plusieurs tables avec des noms de champs.

## 3. Importer un fichier SQL

Pour importer un fichier MySQL déjà existant, nous allons utiliser l'interface qui nous est proposée par phpMyAdmin comme ci-après :

localhost / localhost / ouvrage | phpMyAdmin 2.10.1 - Mozilla Firefox

Structure SQL Rechercher Requête Exporter Importer Opérations Priviléges Supprimer

**Import**

Fichier à importer

Emplacement du fichier texte  Parcourir... (Taille maximum: 2 048Kio)

Jeu de caractères du fichier: utf8

Ces modes de compression seront détectés automatiquement : aucune, gzip, zip

Importation partielle

Permettre l'interruption de l'importation si la limite de temps est sur le point d'être atteinte. Ceci pourrait aider à importer des fichiers volumineux, au détriment du respect des transactions.

Nombre d'enregistrements (requêtes) à ignorer à partir du début  
0

Terminé

Pour obtenir le résultat suivant :

localhost / localhost / ouvrage | phpMyAdmin 2.10.1 - Mozilla Firefox

Structure SQL Rechercher Requête Exporter Importer Opérations Priviléges Supprimer

**Tables**

Table	Action	Enregistrements	Type
exemple	<input type="checkbox"/>      	13	InnoDB
1 table(s)	Somme	13	InnoDB

Tout cocher / Tout décocher Pour la sélection :

Version imprimable Dictionnaire de données

**Créer une nouvelle table sur la base ouvrage**

Nom:  Nombre de champs:

Exécuter

Terminé

Voici le source du fichier ouvrage.SQL qui a été importé pour obtenir la structure de cette base de données :

SQL / ouvrage . SQL

CREATE

```
TABLE 'exemple' (
  'id' int(5) NOT NULL auto_increment,
  'nom' varchar(20) NOT NULL,
  'prenom' varchar(20) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Le fichier SQL se compose d'un certain nombre de lignes :

- `id` : numéro d'identification de ligne ;
- `nom` : champ qui correspond au nom de famille ;
- `prénom` : champ qui correspond à votre prénom ;
- `PRIMARY KEY` : clef primaire ;
- `auto_increment` : compteur automatique sur la colonne c'est-à-dire `id`.

Nous étudierons ultérieurement comment remplir ces champs avec des données.

À la fin de cet ouvrage, nous allons être amenés à utiliser une autre table mais celle-ci possède des données. Voici comment se présente le fichier SQL :

```
CREATE TABLE 'actualite' (
  'id' tinyint(5) NOT NULL auto_increment,
  'titre' varchar(100) NOT NULL,
  'lien' varchar(255) NOT NULL,
  'description' text NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
```

Cette table sera utilisée pour l'utilisation des Flux RSS, donnant la possibilité d'émettre des bulletins d'information vers d'autres sites Internet ; nous y insérerons quelques actualités qui sont très importantes :

```
INSERT INTO 'actualite' ('id', 'titre', 'lien', 'description') VALUES
(1, 'Edition ENI', 'http://www.editions-eni.com/', 'Le site des Editions ENI'),
(2, 'Hello-Design', 'http://www.hello-design.fr', 'Auteur du livre que vous avez
dans les mains'),
(3, 'Actualité PHP et MySQL française', 'http://www.nexen.net',
'Soyez informé des actualités francaises venant du PHP'),
(4, 'Portail de la communauté francophone', 'http://www.phpteam.net/',
'Site permettant de progresser en PHP sous la forme de Tutoriaux'),
(5, 'Le site incontournable', 'http://www.elroubio.net/',
'Venez découvrir Le site du pere de elephant');
```

## 4. Créer un fichier SQL

Pour créer un fichier SQL, nous pouvons utiliser l'interface de phpMyAdmin :

- Soit en mode assisté.
- Soit en utilisant la partie SQL.

Nous allons reprendre le même nom de base de données que nous avons utilisé précédemment, c'est-à-dire **ouvrage**. Puisque celle-ci a été créée, nous devons maintenant créer une table avec un certain nombre de lignes qui correspondent aux noms des champs.

Nous avons choisi **exemple** comme nom d'une table qui contiendra 3 champs : id, nom, prénom.

Comme nous le montre la capture écran, nous nous retrouvons dans une interface permettant de créer nos noms de champs :

- Champ : nom du champ qui nous servira ultérieurement.
- Type : permet de définir quel genre de textes, de chiffres ou de dates sont autorisés dans ce champ.
- Taille : le nombre de caractères autorisés.

Il existe d'autres paramètres pour définir un champ par exemple :

- Null ou non null : cela autorise les (non) valeurs Null dans la colonne, c'est-à-dire que cela permet d'enregistrer une ligne sans que la valeur soit renseignée.
- Auto-incrémentation.

- Verrouiller la clef primaire.

La définition de la clef primaire est assez importante, nous devons y penser lors de la création de la table, mais elle peut toujours être modifiée ou supprimée plus tard.

L'option **auto-increment** consiste à ajouter +1 à la dernière valeur dans le champ. C'est très utile :

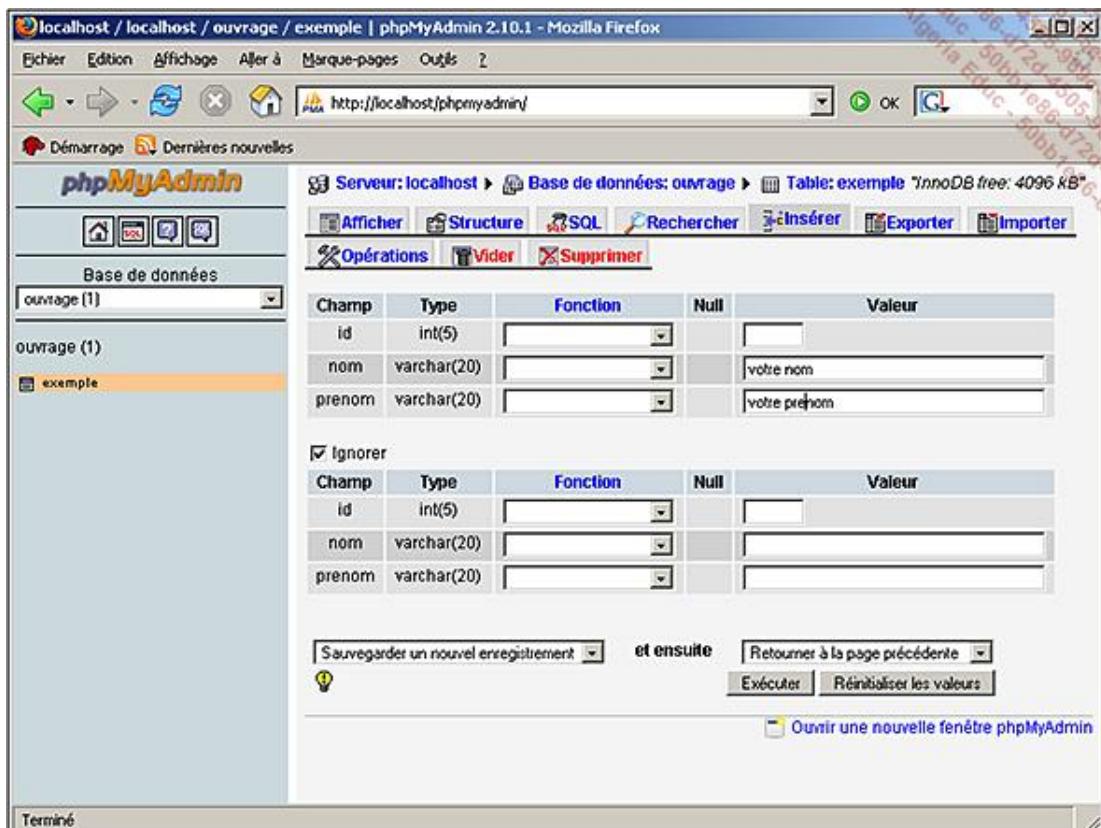
- lorsque nous rencontrons des tables assez complexes,
- pour éviter de générer un compteur en langage PHP car c'est la base de données qui va effectuer l'opération.

Quand vous avez terminé la définition de tous les champs, vous pouvez valider la saisie des champs en cliquant sur le bouton **Sauvegarder**.

 Il est toujours possible d'ajouter un ou plusieurs champs ultérieurement. Pour cela, revenez sur la table et insérez les nouveaux champs.

## 5. Insérer des données

Pour insérer des données, choisissez le menu **Insérer** :



Nous allons juste remplir le champ nom et le champ prénom. Le champ id, qui correspond au compteur de lignes va se remplir tout seul.

Il est tout à fait possible d'ajouter autant de lignes que nous désirons.

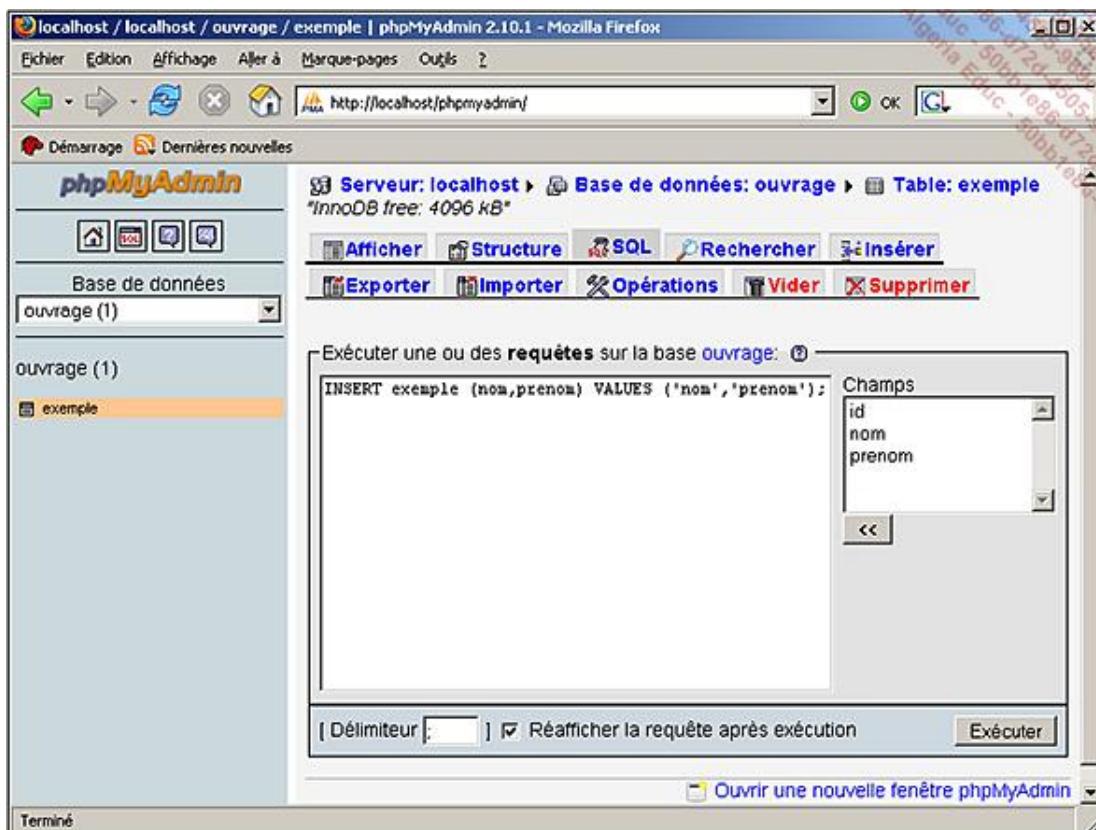
 Lorsque nous insérons des données, nous pouvons être de nouveau obligés d'intervenir sur les données ou sur la structure. Concernant les données, nous pouvons modifier ou supprimer une ligne. Pour la partie structure nous avons un peu plus de possibilités, par exemple ajouter de nouveaux champs, supprimer des champs, modifier des noms de champs.

## 6. Manipulations SQL

Avec l'interface phpMyAdmin, nous avons la possibilité d'utiliser les fonctions SQL standards :

- INSERT : insertion de données.
- UPDATE : mise à jour de données.
- DELETE : suppressions de données.

Pour saisir ces différentes fonctions, nous utilisons une partie de l'interface phpMyAdmin (menu **SQL**) :



Pour insérer des données, nous pouvons saisir :

```
INSERT exemple (nom,prenom) VALUES ('nom','prenom');
```

Ici, nous insérons un nom et un prénom. La ligne recevra le numéro 2 qui correspond à la ligne suivante.

Pour mettre à jour les données, nous pouvons saisir :

```
UPDATE exemple SET nom='essai' WHERE id=1;
```

Ici nous mettons à jour le champ nom de la ligne 1.

Pour visualiser les données, nous pouvons saisir :

```
SELECT * FROM exemple ;
```

Ici, nous demandons d'afficher la totalité du contenu de la table exemple.

Pour supprimer les données d'une ligne, nous pouvons saisir :

```
DELETE * FROM exemple WHERE id=2;
```

Ici nous supprimons la ligne 2 de la table exemple.



Ce ne sont que quelques instructions parmi les plus utilisées. Il existe des instructions de filtre, de tri... Tout ceci sera étudié dans le livre.

---

## 7. Accès à la base de données avec PHP

Nous ne pouvons pas autoriser toutes les personnes qui possèdent un compte sur notre site Internet à utiliser l'interface phpMyAdmin pour compléter la base de données.

C'est pourquoi le langage PHP possède une série de fonctions permettant d'effectuer la communication entre l'interface du navigateur, son langage et la base de données.

Une méthodologie existe pour effectuer la relation, dans l'ordre :

- `mySQL_connect` : Connexion au serveur de données.
- `mySQL_select_db` : Sélection de la base de données.
- `mySQL_query` : Exécution d'une requête.
- `mySQL_xxx` : Exploitation des résultats.
- `mySQL_free_result` : Libération du résultat.
- `mySQL_close` : Fermeture de la connexion.

## Le serveur

Une fois qu'il est décidé d'utiliser un langage et une base de données, nous avons besoin d'un outil pour faire fonctionner ces deux applications. Nous allons utiliser le serveur Apache qui permettra de faire fonctionner notre site Internet en local.

Dans la mesure où la majorité des solutions d'hébergement gratuites et payantes utilisent ce serveur, nous n'avons pas à nous inquiéter sur la compatibilité entre notre configuration et l'hébergement retenu.

 Rappel : un site Internet développé en PHP ne stocke aucune information chez le client (visiteur). Les informations d'exécution sont mémorisées sur le serveur et c'est le serveur qui va effectuer le travail pour les convertir aux normes HTML en vigueur. Par conséquent notre site Internet tournera sur un serveur distant.

---

Les possibilités de paramétrage et de configuration d'un serveur Apache sont une de ses fonctionnalités phares. Elles permettent d'installer et d'héberger plusieurs sites Internet sur un même serveur.

Il est possible de protéger un dossier pour éviter de se faire voler les codes d'accès. Les fichiers utilisés sont **.htaccess** et **.htpasswd**. Ces fichiers seront étudiés plus loin dans l'ouvrage (cf. chapitre Fonctionnalités supplémentaires - HTAccess).

Nous pouvons aussi contrôler et gérer les fonctions URL de redirections qui seront étudiées plus loin dans l'ouvrage (cf. chapitre Fonctionnalités supplémentaires - Redirection d'adresse).

Le serveur Apache peut vérifier si le chemin de la page Internet existe ou pas lorsque le visiteur veut saisir directement la page à la main.

Comme vous l'aurez compris, le serveur Apache permet d'effectuer la majorité du travail pour laisser la place au langage et à la base de données, de l'espace pour exécuter les fonctions principales.

# Les outils

## 1. Les logiciels

Pour réaliser notre application, nous avons besoin d'outils (logiciels). Ils sont variés et nombreux : le plus rapide consiste à utiliser un éditeur de texte (par exemple notepad ou le bloc notes de votre système d'exploitation) mais on peut citer quelques outils destinés à la réalisation de développements en PHP, par exemple :

- Eclipse avec son module PHP Eclipse ;
- ZendStudio ;
- PHP Edit.

Il existe d'autres logiciels mais ils ne sont pas destinés exclusivement au langage PHP.

Lors du développement d'un site Internet, il est nécessaire de pouvoir débuguer l'application pour repérer et corriger les quelques problèmes qui peuvent exister.

Les outils pouvant être utilisés sont par exemple :

- Advanced PHP Debugger (APD) ;
- Xdebug ;
- PHP debugger DBG.

## 2. Environnement prédefini

Pour utiliser le langage PHP, il est nécessaire d'avoir une station de travail qui comprendra la gestion du langage dynamique, une base de données (MySQL) et un serveur proprement dit (Apache).

Il est possible d'installer séparément ces trois applications, mais lorsque nous commençons à vouloir programmer avec le langage PHP, il peut se poser quelques problèmes de paramétrage ou de configuration. La solution consiste à installer un environnement déjà prédefini.

Pour installer et configurer tout ceci, sans énormément de connaissances, il existe déjà des environnements semi-configurés avec un exécutable qui s'occupera de tout installer. Nous pouvons citer les plus connus :

- WAMP (Windows Apache MySQL PHP) : disponible sur le site Internet <http://www.wampserver.com>.
- MAMP (Mac Apache MySQL PHP) : disponible sur le site Internet <http://www.mamp.info>.
- XAMPP (linuX Apache MySQL PHP) : disponible sur le site Internet <http://www.apachefriends.org/en/xampp.html>.



Ce sont trois environnements parmi ceux qui peuvent exister sur le marché. Bien sûr, il en existe d'autres qui fonctionnent aussi très bien.

## 3. Organisation des fichiers PHP

Lorsque nous utilisons un langage dynamique comme le PHP, nous nous retrouvons très rapidement avec de nombreux fichiers informatiques et souvent cela peut devenir un peu lourd à gérer. C'est pourquoi, il est préférable de préparer une structure commune pour les dossiers :

- Admin : contient les fichiers de paramétrage et de connexion à la base de données. Il faudra sécuriser ce

dossier avec les fichiers .htaccess et .htpasswd.

- Include : contient les fichiers comprenant l'extension .inc. Nous y retrouverons la gestion des fonctions, toutes les tâches qui sont amenées à être répétées.
- Images : contient toutes les images qui seront affichées sur notre site Internet.
- Media : contient toutes les vidéos.
- Forum : contient le forum.

Nous pouvons bien sûr créer d'autres dossiers ou sous-dossiers. Le but est toujours de pouvoir s'y retrouver.

## Introduction

Dans ce chapitre, nous définirons un certain nombre d'éléments ou de concept qui seront nécessaires au développement de notre application. Quelques exemples simples nous permettront de mettre en évidence les bases indispensables à sa bonne réalisation tant du point de vue du langage que de la base de données, de sa structure et de la sécurité.

## Les feuilles de style

Avant de commencer la réalisation d'un projet ou d'un site Internet, il est important de préparer une maquette sous la forme d'un squelette que l'on habillera avec des feuilles de style pour obtenir le projet final. Les feuilles de styles sont appelées CSS (Cascading Style Sheet) et permettent de structurer une page écrite en HTML.

Nous allons apprendre à préparer une feuille de style pour donner un style personnalisé à notre application. Cette partie ne concerne pas directement le développement, mais nous ne devons pas perdre de vue que pour réaliser un site Internet sympathique à l'œil, il est nécessaire de mettre un minimum de présentation (utilisation de couleurs sur le texte, personnalisation des liens hypertextes...).

Au lieu d'insérer un style de caractère pour chaque page Internet, nous allons donc regrouper la totalité de ces styles dans une feuille CSS. Ces feuilles de styles nous serviront à définir une certaine police de caractère (ou typo), une taille et une couleur. Elles possèdent d'autres possibilités comme la personnalisation des bordures d'un tableau, la définition d'un fond d'écran...

Avec PHP, il est possible d'utiliser cette même feuille de style pour l'ensemble du site Internet, en définissant un en-tête commun. Ainsi le site Internet aura la même présentation pour toutes ses pages.

### Exercice

Pour nous exercer, affichons un texte "Bonjour, tout le monde" avec une feuille de style externe. Le fichier css.css est composé comme ceci :

```
css/css.php
```

```
texte
{
font-family: Verdana, Arial, sans-serif;
font-size: 20pt;
color: #000000;
font-weight:bold;
}
```

Nous définissons une police de caractères appelée **texte**, de taille 20, de couleur noire et en gras.

Ensuite, nous devons préparer notre fichier php. Il va se composer comme ceci :

- Appel du fichier css.css.
- Affichage de la balise que nous avons choisie.
- Affichage du message.
- Fermeture de la balise.

Nous obtenons le code source suivant :

```
css/index.php
```

```
<?php
echo "<link rel='stylesheet' href='$fichier' type='text/css'>";
echo "<span class=texte>";
echo "Bonjour, tout le monde";
echo "</span>";
?>
```

Tous les mots clés utilisés ici seront expliqués dans l'ouvrage.

En visualisant le résultat, nous obtenons ceci :



Nous pouvons proposer à l'utilisateur un choix multiple pour la police de caractères et par conséquent pour une présentation.

css/exemple2.php

```
<?php

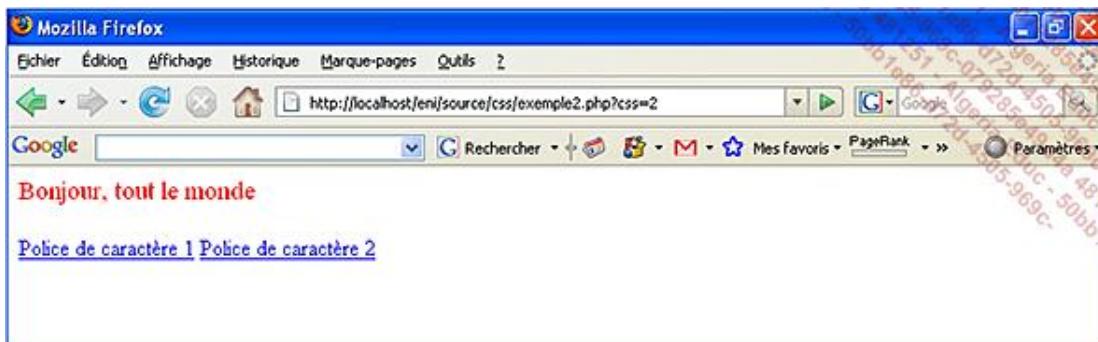
if(isset($_GET['css'])) && $_GET['css']=='1'
$fichier="css.css";
else
$fichier="css2.css";

echo "<link rel='stylesheet' href='$fichier' type='text/css'>";

echo "<span class=texte>";
echo "Bonjour, tout le monde";
echo "</span>";

echo "<br><br>";
echo "<a href='exemple2.php?css=1'>Police de caractère 1</a> ";
echo "<a href='exemple2.php?css=2'>Police de caractère 2</a> ";
?>
```

Ici, nous donnons la possibilité à notre visiteur de choisir le style de caractères qu'il souhaite.



Après son choix, la page Internet se recharge pour se mettre en relation avec la bonne feuille de style.



Attention pour chaque feuille de style à bien utiliser les mêmes balises sinon les présentations risquent de ne pas correspondre à vos choix.

---

# Les bases du langage PHP

Pour tous les langages informatiques nous devons connaître certaines bases. Ces bases sont utilisées régulièrement et le langage PHP n'échappe pas à la règle. C'est pourquoi dans cette partie nous allons voir les notions importantes qui sont :

- les variables ;
- les conditions ;
- les boucles.

## 1. Les variables

Une variable correspond à une valeur qui mémorise provisoirement une donnée numérique (chiffre) ou du texte (donnée alphabétique). Cette information sera utile pour exécuter une opération précise.

Les variables que nous utilisons sont destinées à une seule page de notre site, c'est pourquoi leur durée de vie est très courte.

Lorsqu'un langage permet l'utilisation d'une variable, nous pouvons nous permettre d'exécuter certaines manipulations.

Nous allons mémoriser dans la variable **a** le mot **Editions** et dans la variable **b** le mot **ENI**.

Nous affichons chacune des variables à l'écran puis les deux variables sur une seule ligne.

variable/exemple1.php

```
<?php
$a="Editions";
$b="ENI";

echo "$a<br>";
echo "$b<br>";
echo "$a $b<br>";
?>
```

Nous pouvons générer une nouvelle variable en enregistrant les variables **a** et **b** dans une autre variable **c**.

```
<?php
$c=$a." ".$b;
echo "$c<br>";
?>
```

Pour obtenir le résultat suivant :



Nous venons de voir l'utilisation de variables avec du texte, mais il est possible d'utiliser les variables avec des chiffres.

Les chiffres permettent d'effectuer les fonctions de calcul de base (addition, soustraction, multiplication, division) et des calculs complexes.

Illustrons avec l'addition l'une des possibilités que PHP peut nous offrir. Nous allons réaliser une addition dans une variable **a** :

```
<?php  
$a = 1+1;  
  
echo "a : $a<br>";  
?>
```

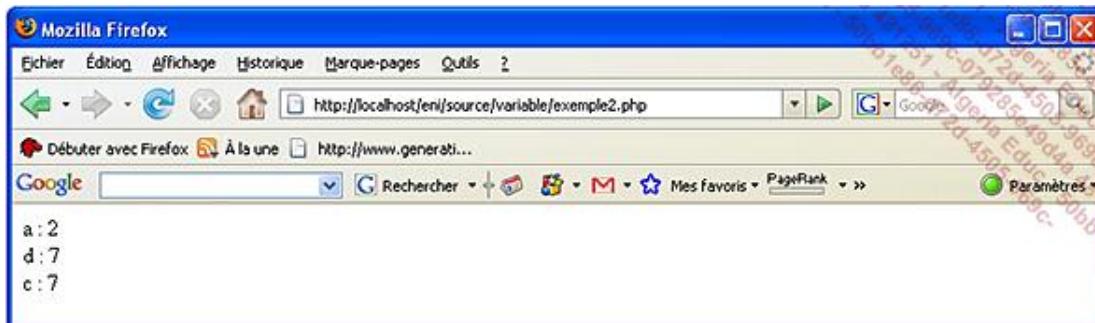
Le langage PHP nous offre la possibilité d'effectuer des additions basiques, c'est-à-dire décomposer l'addition variable par variable. Comme ceci :

```
<?php  
$d = 5;  
$d = $d + $a;  
echo "d : $d<br>";  
?>
```

Cependant nous pouvons effectuer ces additions sous une autre forme : au lieu de répéter deux fois la même variable, nous pouvons demander au langage PHP d'effectuer l'ajout de la deuxième variable dans la première. Ici, nous allons ajouter et prendre en compte ce nouveau résultat dans une variable **c** et nous ajoutons la valeur de **a**.

```
<?php  
$c = 5;  
$c += $a;  
echo "c : $c<br>";  
?>
```

Nous obtenons le résultat suivant :



 Ce qui vient de vous être montré sera utilisé un peu plus loin dans l'ouvrage, entre autres pour la création d'une clef aléatoire (cf. ce chapitre, Le squelette de l'application - Les fonctions).

## 2. Les conditions

### If...else

Les conditions servent dans le langage informatique à déterminer si le résultat obtenu est égal ou pas à une autre valeur. Ce qui nous permet d'obtenir pour résultat une valeur Vrai ou Faux.

La syntaxe employée pour les conditions est :

**If**

condition

**Else**

sinon

Dans cette fonction, nous pouvons effectuer certains tests de comparaison :

**==** : Egal à  
**>** : est supérieur à  
**<** : est inférieur à  
**>=** : est supérieur ou égal à  
**<=** : est inférieur ou égal à  
**!=** : est différent de

Nous allons utiliser ces conditions lors d'un petit exemple :

condition/exemple1.php

```
<?php
$a="Editions ENI";

if ($a == "Editions ENI")
{
    echo "Vrai";
}
else
{
    echo "Faux";
}
?>
```

Notre exemple nous retourne le résultat Vrai.

Bien sûr nous pouvons effectuer plusieurs tests dans une même condition, la syntaxe utilisée est alors :

```
AND correspond au terme ET
OR correspond au terme OU
&& correspond au mot AND
|| correspond au mot OR
```

condition/exemple2.php

```
<?php
$a="Editions ENI";
$b="hello";

if ($a == "Editions ENI" && $b=="hello")
{
    echo "Vrai";
}
else
{
    echo "Faux";
}
?>
```

Nous obtenons un résultat Vrai car nous regardons si la variable **a** possède bien la valeur **Editions ENI** et si la variable **b** possède bien la valeur **hello**.

Nous venons de voir l'utilisation des conditions simples, c'est-à-dire avec deux résultats possibles. Il est possible d'effectuer des tests en cascade c'est-à-dire d'utiliser plusieurs conditions les unes en dessous des autres avec la fonction elseif.

**Elseif** : sinon si

Pour illustrer cette fonction, nous allons, à partir d'un formulaire, déterminer si la valeur que nous avons saisie correspond à l'une des quatre tranches que nous allons déterminer.

Ces tranches iront de 25 en 25, c'est-à-dire :

- 0 à 25 : tranche 1 ;

- 26 à 50 : tranche 2 ;
- 51 à 75 : tranche 3 ;
- 75 à 99 : tranche 4.

Nous prenons une variable et nous mettons une valeur au hasard pour voir le résultat :

condition/exemple3.php

```
<?php
$tranche=2;
if ($tranche<=0)
    echo "Tranche 0<br>";
elseif ($tranche==1)
    echo "Tranche 1<br>";
elseif ($tranche==2)
    echo "Tranche 2<br>";
elseif ($tranche==3)
    echo "Tranche 3<br>";
elseif ($tranche==4)
    echo "Tranche 4<br>";
elseif ($tranche==5)
    echo "Tranche 5<br>";
elseif ($tranche==6)
    echo "Tranche 6<br>";
elseif ($tranche==7)
    echo "Tranche 7<br>";
elseif ($tranche==8)
    echo "Tranche 8<br>";

elseif ($tranche==9)
    echo "Tranche 9<br>";
else
    echo "Hors tranche<br>";
?>
```

Le résultat montre qu'il s'agit de la Tranche 2.

### **Switch**

Dans l'exemple précédent nous voyons que les lignes se répètent. Nous pouvons effectuer ces tests par l'intermédiaire d'une autre instruction :

### **Switch**

Teste la variable.

### **Case**

Valeur de la variable.

### **Break**

Sort du test.

condition/exemple4.php

```
<?php
$tranche=2;

switch ($tranche)
{
case 0:    echo "Tranche 0<br>"; break;
case 1:    echo "Tranche 1<br>"; break;
case 2:    echo "Tranche 2<br>"; break;
```

```

case 3: echo "Tranche 3<br>"; break;
case 4: echo "Tranche 4<br>"; break;
case 5: echo "Tranche 5<br>"; break;
case 6: echo "Tranche 6<br>"; break;
case 7: echo "Tranche 7<br>"; break;
case 8: echo "Tranche 8<br>"; break;
case 9: echo "Tranche 9<br>"; break;
default:
    echo "Hors tranche";
    break;
}
?>

```

Nous obtenons les mêmes résultats mais nous avons une bien meilleure clarté de notre code.

### 3. Les boucles

L'utilisation des boucles est présente dans tous les langages de programmations. Cette technique est très utile pour répéter une action, pour réaliser un compteur ou un système d'affichage de lignes de résultats.

Nous allons aborder les boucles les plus répandues dans le langage PHP.

#### For

L'instruction **for** permet d'effectuer une boucle en spécifiant le nombre de répétitions.

**For** : boucle limitée

L'instruction nous permet de déterminer la position de départ, suivie de la position de fin et pour terminer un pas d'incrément.

Nous allons afficher dans une boucle les chiffres de 1 à 10 avec un pas de 1 :

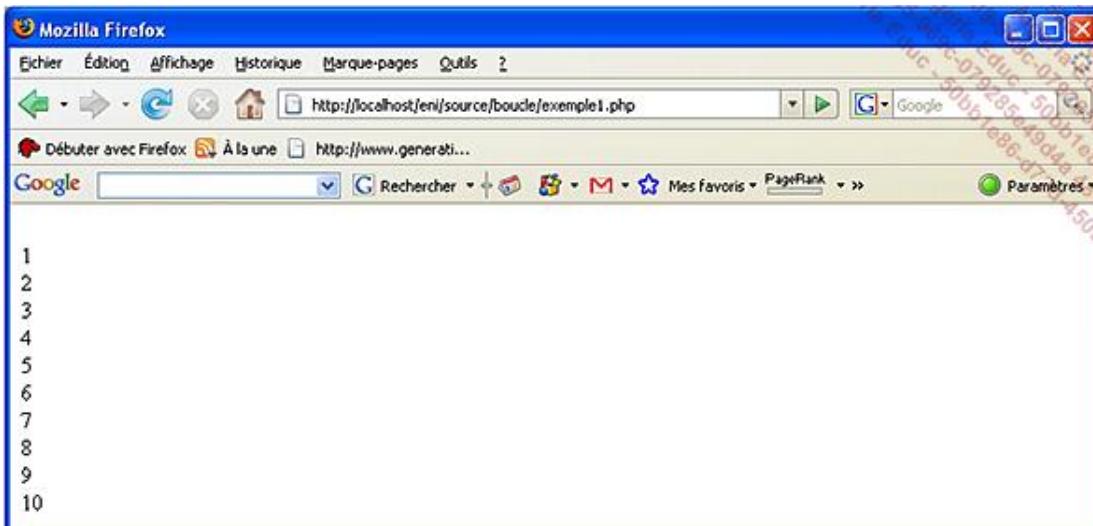
boucle/exemple1.php

```

<?php
for ($i=0;$i<=10;$i++)
{
    echo "$i<br>";
}
?>

```

Pour obtenir le résultat suivant :



#### while

Cette instruction **while** permet d'effectuer le même travail que précédemment, sauf qu'ici, nous n'avons pas de

position de fin.

### **While**

Boucle sans limite.

Elle nous sera utile pour afficher un résultat venant d'une table, car nous ne pouvons pas connaître le nombre de lignes provenant du choix de notre visiteur ; c'est pourquoi cette instruction affichera le résultat ligne par ligne, tant qu'il reste des lignes à afficher.

condition/exemple2.php

```
<?php
$i=0;
while ($i<=10)
{
    echo "$i<br>";
    $i++;
}
?>
```

Dans notre exemple, nous avons inséré un compteur pour permettre d'afficher le système de compteur.

### **Foreach**

Cette instruction permet d'itérer sur les éléments d'un tableau. Elle peut nous être utile pour effectuer une recherche dans un tableau dynamique.

### **Foreach**

Boucle tableau.

condition/exemple3.php

```
<?php
$arr = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
foreach ($arr as &$i)
{
    echo "$i<br>";
}
?>
```



L'utilisation de cette boucle sera étudiée un peu plus loin dans le livre (cf. dans ce chapitre - Les tableaux).

# La base de données

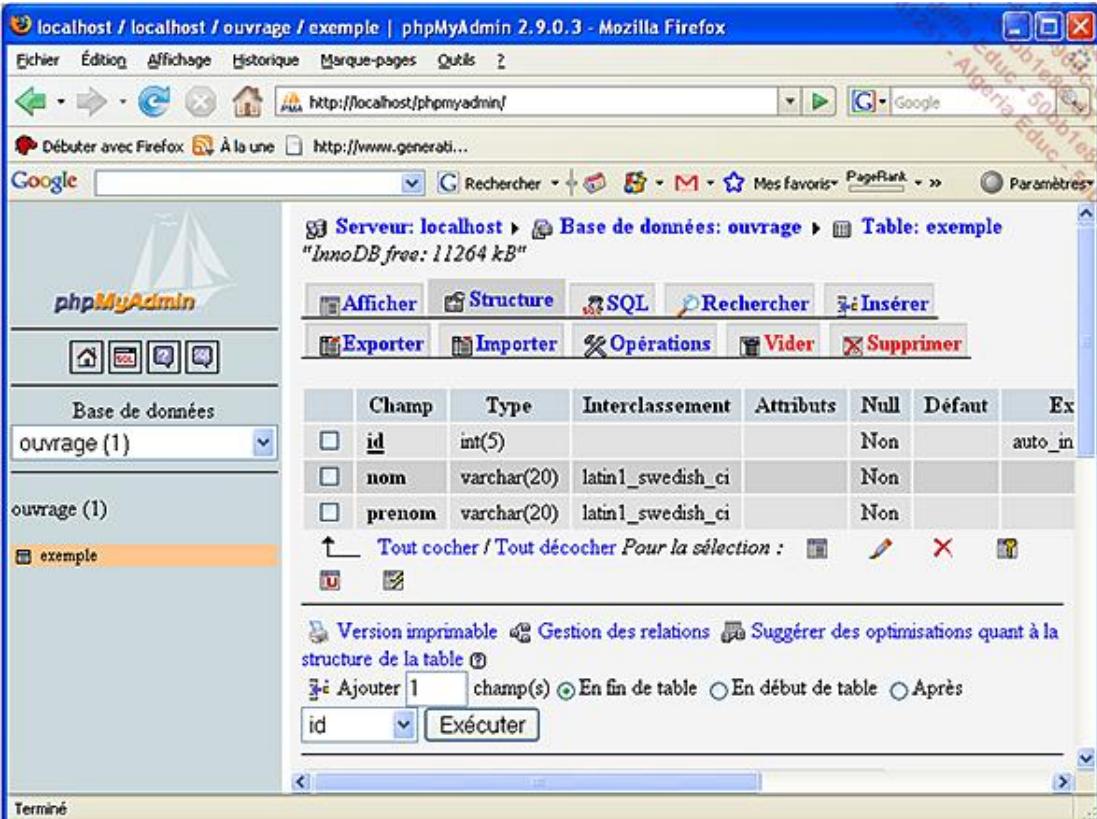
Nous allons aussi avoir besoin pour notre application d'une base de données. Nous allons réutiliser le même fichier SQL **ouvrage.sql** que nous avons étudié dans le chapitre précédent en effectuant les appels à la base de données par MySQLi et MySQL.

Le format MySQLi correspond à une extension de MySQL. Actuellement, nous pouvons dire que ce nouveau format risque de devenir le successeur de MySQL car il a été optimisé pour répondre aux différentes contraintes de charge et de rapidité liées à Internet.

## 1. Connexion

Quel que soit le format que nous allons choisir, nous allons utiliser la même interface phpMyAdmin, puisque c'est la même société qui propose le format MySQL et MySQLi.

Lorsque nous importons notre fichier ouvrage.sql, nous pouvons voir le résultat dans l'interface phpMyAdmin :



The screenshot shows the phpMyAdmin interface for the 'exemple' table in the 'ouvrage' database. The table structure is as follows:

Champ	Type	Interclassement	Attributs	Null	Défaut	Ex
<input type="checkbox"/> <b>id</b>	int(5)			Non		auto_in
<input type="checkbox"/> <b>nom</b>	varchar(20)	latin1_swedish_ci		Non		
<input type="checkbox"/> <b>prenom</b>	varchar(20)	latin1_swedish_ci		Non		

Below the table, there are buttons for 'Exporter', 'Importer', 'Opérations', 'Vider', and 'Supprimer'. The 'Opérations' button is highlighted. There is also a section for adding a new column to the table.

Lorsque notre fichier base de données est créé, nous devons par l'intermédiaire du langage PHP préparer la connexion à notre base de données se trouvant sur le serveur pour être certains que la communication sera correctement réalisée.

BaseDeDonnee/exemple-mysqli.php

```
<?php
$serveur="localhost";
$user="root";
$password="";
$bdd="ouvrage";

$connex = mysqli_connect($serveur, $user, $password, $bdd);
if (mysqli_connect_errno())
{
echo ("Echec de la connexion : ". mysqli_connect_error());
```

```
die ("Code d'erreur : ".mysqli_connect_errno());
}
?>
```

Pour effectuer la connexion en MySQLi, nous allons utiliser la fonction :

### **Mysqli\_connect**

Ouvre la connexion au serveur MySQLi.

Nous insérons un petit test qui consiste à repérer si une erreur apparaît, auquel cas nous affichons un message ; les fonctions utilisées :

### **Mysqli\_connect\_error**

Retourne le message d'erreur de connexion MySQLi.

### **Mysqli\_connect\_errno**

Retourne le code erreur de la connexion MySQLi.

Si vous tenez à utiliser le format MySQL classique pour effectuer la même connexion, nous obtenons ceci :

BaseDeDonnee/exemple-mysql.php

```
<?php
$serveur="localhost";
$user="root";
$password="";
$bdd="ouvrage";
$connex = mysql_connect($serveur, $user, $password, $bdd);
if (mysql_errno()) die ("Echec de la connexion : ". mysql_error());

mysql_select_db($bdd,$connex);
?>
```

Dans cet exemple nous utilisons les fonctions suivantes :

### **Mysql\_connect**

Ouvre la connexion au serveur MySQL.

### **Mysql\_error**

Retourne le numéro d'erreur.

### **Mysql\_select\_db**

Sélectionne une base de données.

---

 Ces deux exemples montrent la connexion au serveur et à la base de données **ouvrage**. Nous étudierons un peu plus en détail toutes ces lignes un peu plus loin (cf. dans ce chapitre - Les bases de données).

---

Nous allons placer ce fichier dans le dossier **admin**. C'est un fichier très sensible car il contient tous les codes nécessaires pour se connecter au serveur et à la base de données.

Lorsque nous basculerons notre application sur Internet, nous devrons protéger le dossier qui contient les paramètres de connexion par l'intermédiaire des fichiers .htaccess et .htpasswd.

### **.htaccess**

Contient les directives de protection.

### **.htpasswd**

Contient les identifiants et mots de passe des personnes qui peuvent accéder à ce dossier.

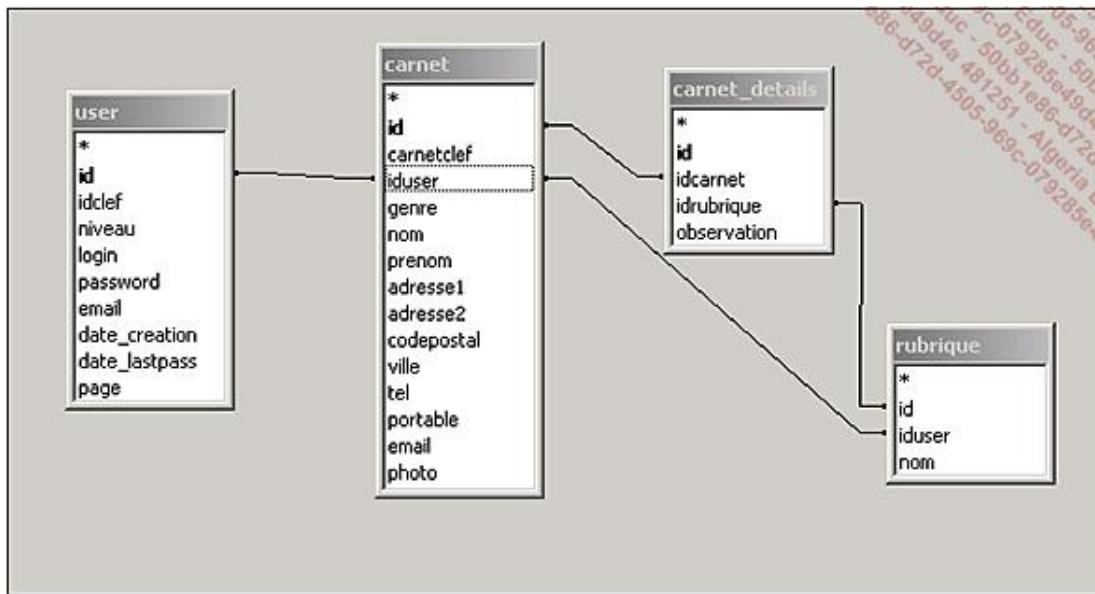
Ces protections de dossiers seront étudiées aussi dans une autre partie de l'ouvrage (cf. chapitre Fonctionnalités supplémentaires - HTAccess). Il faut déjà prendre conscience des intentions malhonnêtes de certains visiteurs.

## 2. Structure de la base de données

Pour revenir à notre application, nous allons importer notre fichier SQL appelé ENI.SQL avec l'interface phpMyAdmin comme nous l'avons étudié dans le chapitre précédent.

Le source du fichier ne sera pas détaillé ici. Il est juste disponible avec les autres fichiers en téléchargement.

Cependant, nous allons étudier la structure de la base SQL, permettant ainsi de nous approcher le plus possible de notre application. Ce fichier correspond à l'ensemble des tables et des données qui seront utilisées dans notre application "Carnet d'adresses".



La table **user** représente tous les possesseurs de comptes. Elle enregistre les comptes des personnes qui sont autorisées à posséder un carnet d'adresse. Nous allons y trouver certains champs importants :

- **idclef** : une clef cryptée nous permet de ne pas faire apparaître en clair le numéro de ligne afin d'éviter, si l'on modifie un caractère de ce champ, de tomber sur l'enregistrement suivant ou précédent pour lequel nous n'avons pas les droits d'accès.
- **niveau** : il nous sera utile pour déterminer le niveau d'accès du possesseur du compte.
- **page** : il nous permet lors de l'identification du possesseur du compte de l'envoyer sur une page spécifique.

La table **carnet** est destinée à mémoriser les informations du possesseur du compte ; cette table enregistrera à partir d'un formulaire tous les contacts des personnes qui possèdent un compte sur notre site Internet. Il pourra avoir dans un carnet plusieurs enregistrements personnels. Les champs importants de la table sont :

- **carnetclef** : ce champ sera utilisé pour sécuriser les données et permet donc de ne pas voir le numéro de la ligne comme 1 idclef (que nous avons vu quelques lignes au-dessus).
- **iduser** : ce champ mémorise le numéro de ligne du possesseur du compte et ne sera visible pour aucun visiteur.

La table **rubrique** représente la liste des différentes rubriques que tous les possesseurs de comptes pourront personnaliser.

Cette liste correspond à différents choix possibles pour classer certaines informations. Par ailleurs étant donné que ces listes sont destinées à une seule personne, c'est-à-dire le possesseur du compte, elles seront différentes pour tous les possesseurs de comptes.

La table **carnet\_details** sert à mémoriser une ou plusieurs informations pour un des contacts du possesseur de compte. Elle est associée avec la table carnet.

Ainsi, le titulaire du compte possède un carnet dans lequel il peut mettre tous ses contacts, et de plus pour chaque contact, il aura la possibilité d'inscrire une ou plusieurs informations.

Voici le résultat une fois ce fichier importé :

The screenshot shows the phpMyAdmin interface for the 'eni' database. The left sidebar lists the database 'eni (4)' and its tables: 'carnet', 'carnet\_details', 'rubrique', and 'user'. The 'user' table is currently selected, indicated by an orange highlight. The main panel displays a table with the following data:

Table	Action	Enregistrements	Type
carnet		36	InnoDB
carnet_details		33	InnoDB
rubrique		4	InnoDB
user		3	InnoDB
4 table(s)	Somme	76	InnoDB

Below the table, there are buttons for 'Tout cocher / Tout décocher' and 'Pour la sélection :'. At the bottom, there are links for 'Version imprimable' and 'Dictionnaire de données'.

Le fichier SQL est disponible en téléchargement.

# Le squelette de l'application

## 1. Les includes

PHP possède une fonctionnalité très utilisée : les includes. C'est une fonctionnalité très simple, mais fort utile et surtout très puissante. Elle permet d'insérer une page dans une autre page.

Nous allons voir que vous ne pouvez pas ignorer cette fonction : par exemple dans notre site Internet, nous avons un menu et une présentation identique pour toutes les pages. Grâce à cette fonction, nous n'avons pas besoin de recopier cette présentation pour toutes les pages Internet.

L'autre avantage est en rapport avec les modifications et évolutions futures ou pendant la période de développement du site Internet : nous n'aurons à effectuer qu'une seule fois les corrections pour qu'elles s'appliquent sur l'ensemble du site Internet.

Pour effectuer l'appel d'un fichier extérieur, nous devons créer ce fichier extérieur :

Fichier page.php

```
<html>
<head></head>
<body>
Les Editions ENI
</body>
</html>
```

Pour afficher cette page, nous allons faire appel à une autre page :

Fichier index.php

```
<?php
include "page.php";
?>
```

Il existe d'autres fonctions qui permettent également d'appeler des fichiers externes :

- **include** et **require** : exécute le fichier spécifié comme un argument.
- **include\_once** et **require\_once** : mêmes fonctions que ci-dessus avec la particularité de ne s'exécuter qu'une seule fois. C'est très utile pour éviter les messages de redéfinitions de fonction.

Nous utiliserons de préférence la fonction **require\_once**.

## 2. Les pseudo-frames

Les pseudo-frames (appelés FRAME dans le langage HTML) sont très souvent utilisées dans le langage dynamique. Elles découpent l'écran en plusieurs parties, permettant ainsi de gérer séparément l'actualisation d'une ou plusieurs parties.

L'autre particularité des pseudo-frames est qu'elles peuvent être indexées par les moteurs de recherche sans soucis de présentation.

Pour exécuter les pseudo-frames, la meilleure solution consiste, comme nous l'avons vu dans les pages précédentes, à utiliser les fonctions include ou require.

Pour illustrer l'utilisation des pseudo-frames et en guise d'exemple de base, nous allons créer un fichier avec un en-tête (head.inc.php) et un pied de page (footer.inc.php).

 Pour mémoire, l'extension **.inc.php** concerne des fichiers includes qui ne seront pas pris en compte comme des fichiers principaux.

frame/head.inc.php

```
<?php
echo "Frame : Partie en haut de l'écran";
?>
```

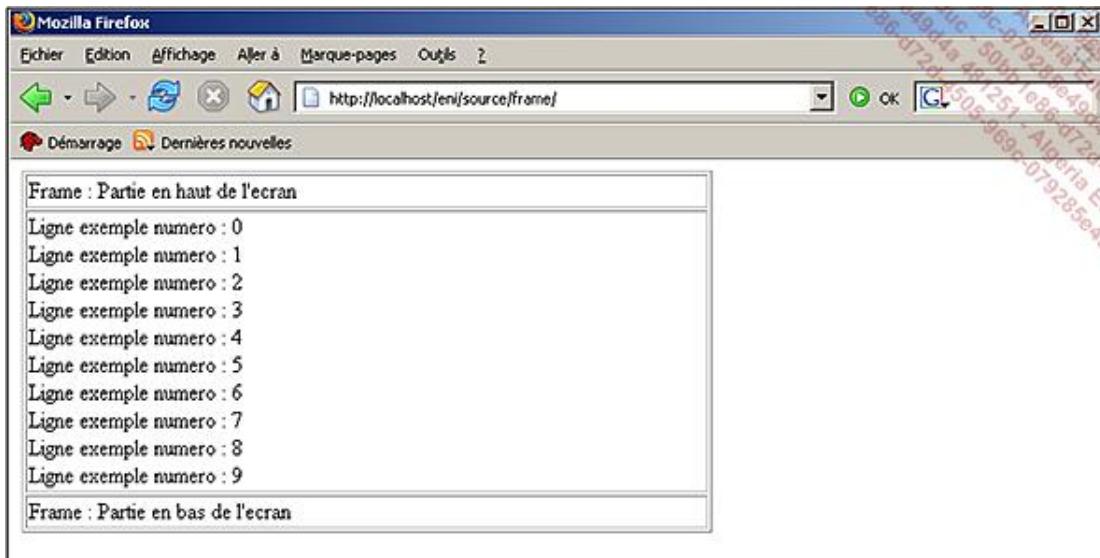
frame/footer.inc.php

```
<?php
echo "Frame : Partie en bas de l'écran";
?>
```

frame/index.php

```
<html>
<body>
<table border="1" width="500">
  <tr>
    <td> <?php require('head.inc.php'); ?> </td>
  </tr>
  <tr>
    <td>
<?php
  $i=0;
  $fin=10;
  while($i != $fin)
  {
  echo "Ligne exemple numero : ".$i."<br>";
  $i++;
  }
  ?> </td>
  </tr>
  <tr>
    <td> <?php require('footer.inc.php'); ?> </td>
  </tr>
</table>
</body>
</html>
```

Voici le résultat :



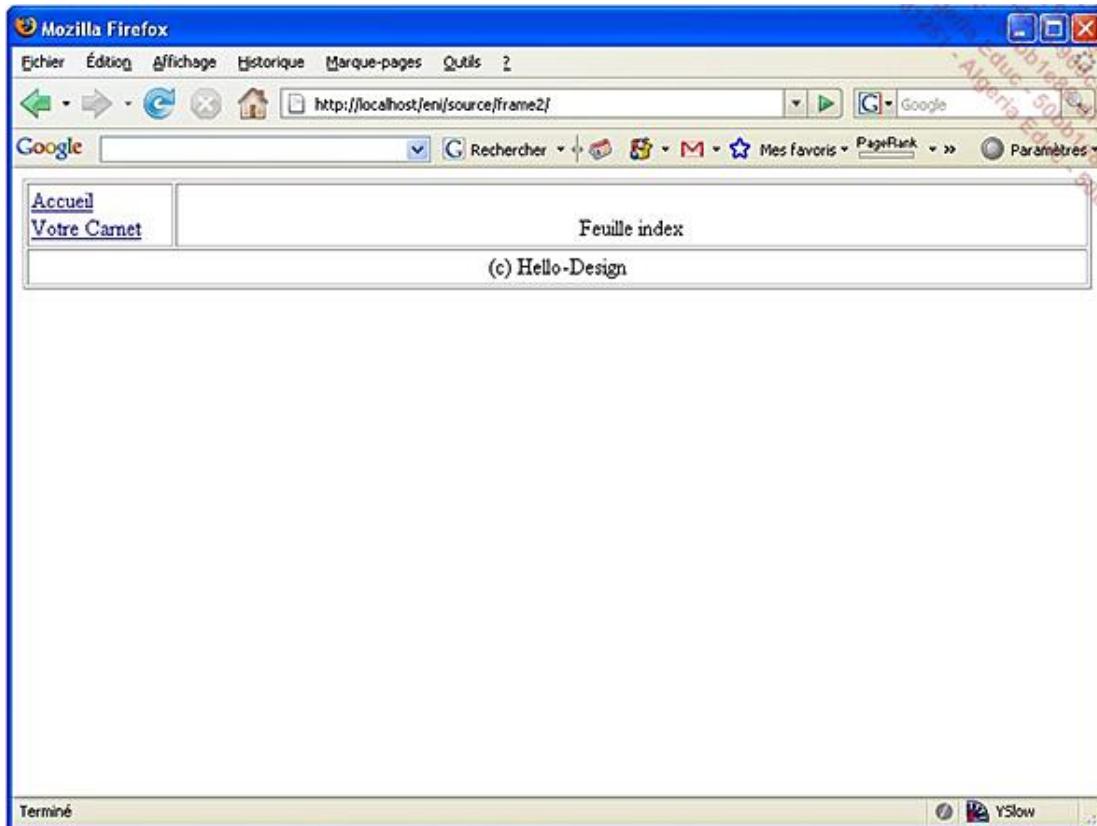
Pour concevoir des pseudo-frames il suffit de créer un tableau HTML classique et dans chaque partie du tableau d'inclure une relation avec les fichiers PHP.

L'avantage d'utiliser cette technique est que le lien URL sera différent pour chaque page.

Dans notre application, nous allons utiliser la technique des pseudo-frames pour permettre d'afficher et de gérer le menu de notre application. Notre menu se composera d'un titre et de sous-menus.

Lorsque nous accédons au site, aucun sous-menu n'est affiché, mais suivant le choix du visiteur, le sous-menu sera affiché ou pas. Par ailleurs, ce sous-menu disparaîtra au changement de menu.

Voici l'écran de début :

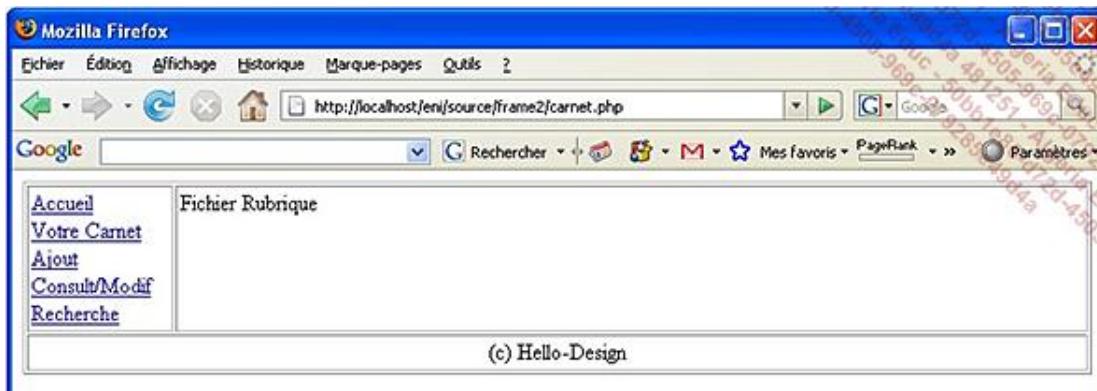


Le code de notre fichier head.inc.php se présente comme ceci :

frame2/head.inc.php

```
<table border="1" cellspacing="2" cellpadding="2" width="100%">><tr>
<td width=100 valign=top>
<a href=index.php>Accueil</a><br>
<a href=carnet.php>Votre Carnet</a><br>
<?php if (isset ($menu) && $menu=="carnet") { ?>
<a href=#>Ajout</a><br>
<a href=#>Consult/Modif</a><br>
<a href=#>Recherche</a><br>
<?php } ?>
</td><td valign=top>
```

Si nous choisissons de sélectionner le menu **Votre carnét**, une nouvelle présentation apparaît comme ceci :



Tout d'abord, vous pouvez noter que l'url a changé et correspond à notre choix. Lors de l'affichage de la page, de nouvelles rubriques sont apparues. Voici le code source exemple :

frame2/carnet.php

```
<?php
$menu="carnet";
require_once ("head.inc.php");
echo "Fichier Rubrique";
require_once ("footer.inc.php");
?>
```

Sur la première ligne une variable nous indique que nous sommes dans la page carnet. Ensuite nous allons afficher notre menu et en reprenant l'exemple précédent, head.inc.php, nous voyons que si le menu carnet est bien sélectionné notre application va afficher les nouvelles rubriques.

Si nous resélectionnons le menu **Accueil**, nous allons voir disparaître le sous-menu qui venait d'apparaître.



Le menu complet est disponible en téléchargement.

### 3. Les fonctions

Les fonctions permettent aux programmeurs de réaliser un extrait de code qui peut se répéter de nombreuses fois. Nous pouvons aussi utiliser une fonction pour isoler une partie de notre application assez compliquée qui sera ainsi plus facile à faire évoluer. On peut aussi les appeler des sous-programmes.

Les fonctions sont très utilisées pour, par exemple, la gestion d'un caddie ou d'un panier (site marchand). Mais elles sont aussi utilisées pour le formatage de champ ou le contrôle de champs, de dates...

Il est possible pour une fonction de faire appel à une autre fonction (voir exemple 2).

#### Return

Termine la fonction et retourne une valeur.

Exemple d'une fonction simple qui affichera comme résultat le message **Bonjour** :

fonctions/exemple1.php

```
<?php
function retourne_message ($text)
{
    return $text;
}

echo retourne_message ("Bonjour");
?>
```

Pour revenir à notre application, nous allons avoir besoin d'un petit générateur de clé automatique.

Ce générateur de clé sera utilisé pour obtenir un minimum de sécurité et garder une clef d'identification qui sera utilisée pour les SESSIONS (voir plus loin). À chaque identification, cette clef sera recalculée automatiquement.

Ce générateur de clef sera décomposé en deux fonctions dans un exemple que nous utiliserons pour notre application.

La première fonction (ci-après) permet de récupérer le timestamp UNIX en microsecondes ; si plusieurs personnes effectuent l'affichage de la même page au même moment, cette fonction permettra de retourner des valeurs différentes et il y aura donc moins de risques de conflit au niveau de la sécurité.

fonctions/exemple2.php

```
<?php
function make_mms()
{
    list($usec, $sec) = explode(' ', microtime());
```

```
    return (float) $sec + ((float) $usec * 100000);
}
?>
```



**explode** : coupe une chaîne en segments. Cette fonction permet de couper une chaîne en segments (en parties) sous la forme d'un tableau de chaîne ; **microtime** : retourne le timestamp Unix avec les microsecondes ; **float** : retourne une valeur numérique.

La deuxième fonction, qui est très utile, permet de récupérer une clef d'identification d'une longueur de 20 caractères. Elle va nous servir de fonction principale :

```
<?php
function recuper_clef()
{
$idcle="";
$taille = 20;
$lettres = "abcdefghijklmnopqrstuvwxyz0123456789";
srand(make_mms());
for ($i=0;$i<$taille;$i++)
{
$idcle.=substr($lettres,(rand()%strlen($lettres)),1);
}
return $idcle;
}
?>
```



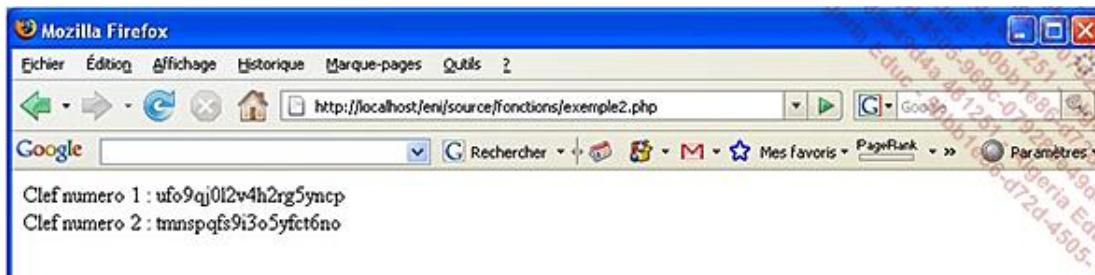
**srand** : permet de générer un nombre aléatoire avec le résultat provenant de l'autre fonction (ci-dessus) **microtime** ; **substr** : retourne un segment de chaîne.

Maintenant, il nous reste à afficher les valeurs pour vérifier que nos deux fonctions travaillent parfaitement.

Afficher les deux clefs pour vérifier :

```
<?php
echo "Clef numero 1 : ".recuper_clef()."<br>";
echo "Clef numero 2 : ".recuper_clef()."<br>";
?>
```

Voici le résultat :



# Les formulaires

## 1. GET, POST, REQUEST et FILES

GET et POST sont des méthodes standards du HTML. Elles existent depuis de nombreuses années en PHP. Elles sont utilisées à l'aide de `$_GET` et `$_POST`.

La fonction FILE se présente en PHP sous la forme `$_FILES`. Elle est utilisée pour l'envoi de fichiers à partir d'un formulaire vers le serveur, par exemple une photo, un document...

 La variable `$_REQUEST` correspond à l'incorporation des fonctions GET, POST, COOKIE. C'est une variable globale et nous vous déconseillons de l'utiliser. Le paramètre `$_FILES` n'est plus inclus dedans depuis la version de PHP 4.3.

L'utilisation de ces variables permet de recevoir des données venant d'un formulaire, d'un lien...

Différence entre `$_GET` et `$_POST` : la première affiche les informations en clair directement dans la barre URL de votre navigateur. La deuxième transmet les mêmes informations mais elles ne seront pas affichées dans la barre de navigation :

Voici comment utiliser ces variables :

```
<form name="saisie" method="POST" action="exemple1.php">
...
</form>
```

### a. Envoi des données à partir d'un formulaire

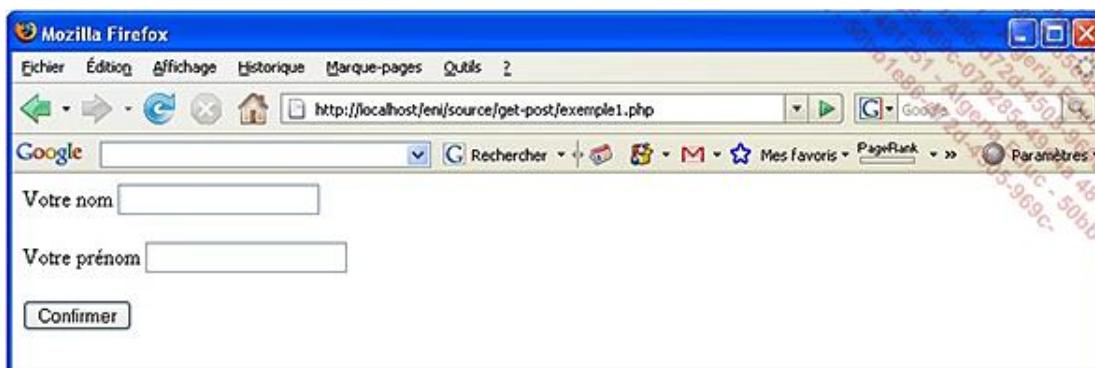
#### \*\$\_POST

Pour récupérer les données venant de notre formulaire nous allons utiliser `$_POST` si nous avons envoyé les données de notre formulaire par la méthode POST.

Exemple : soit un formulaire comprenant deux champs : **nom** et **prenom**.

Après la saisie de notre formulaire, nous afficherons après un "bonjour" le nom et le prénom que nous venons de saisir.

Voici l'écran de saisie :



Pour afficher ces champs, utilisons le formulaire suivant :

```
get-post/exemple1.php
```

```
<html>
<body>
<form name="saisie" method="POST" action="exemple1.php">
Votre nom <input name="nom" type="text"><br /><br />
Votre prénom <input name="prenom" type="text"><br /><br />
  <input name="Confirmer" type="submit" value="Confirmer">
```

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

Lors de la validation des données, nous allons afficher les champs avec les paramètres de validation de PHP.

```
<?php
if (sizeof($_POST) > 0)
{
echo "Bonjour : ".$_POST['prenom']." ".$_POST['nom']."<br />";
}
?>
```

Le source ci-dessus montre une particularité : **sizeof**. Cette fonction nous informe si notre formulaire possède au moins des données qui ont été envoyés.



**sizeof** : alias de la fonction **count** qui compte le nombre d'éléments envoyés par le formulaire.

Le contenu de `$_POST` est testé pour voir si notre formulaire a bien été envoyé.

 Cet exemple est disponible en téléchargement.

### **\*\$\_GET**

Si nous utilisons la méthode GET, nous allons récupérer les valeurs de données saisies à l'aide de `$_GET` :

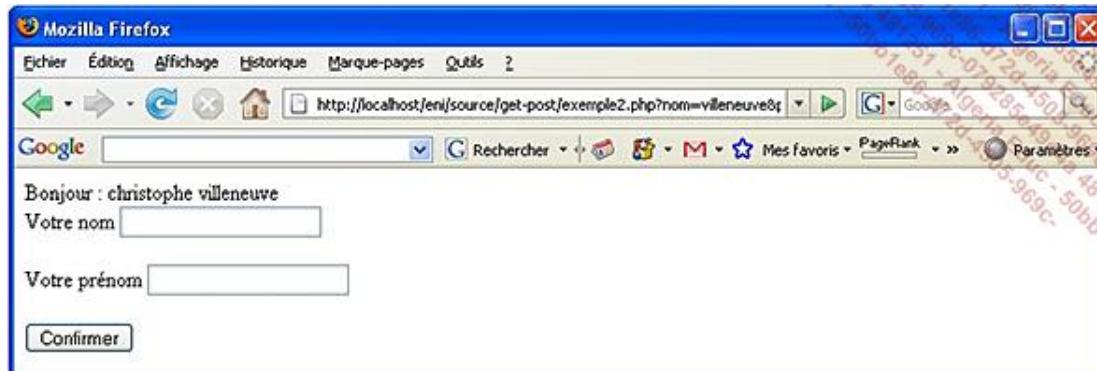
```
<form name="saisie" method="GET" action="exemple2.php">
...
</form>
```

Nous allons utiliser le même exemple que ci-dessus mais avec ce nouveau code.

get-post/exemple2.php

```
<?php
if (sizeof($_GET) > 0)
{
echo "Bonjour : ".$_GET['prenom']." ".$_GET['nom']."<br />";
}
?>
<html>
<body>
<form name="saisie" method="GET" action="exemple2.php">
Votre nom <input name="nom" type="text"><br /><br />
Votre prénom <input name="prenom" type="text"><br /><br />
<input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>
```

Le résultat sera identique mais les données saisies à partir du formulaire seront affichées dans la barre d'adresse de votre navigateur sur comme l'écran ci-après :





La variable `$_GET` possède une autre particularité. Elle permet de récupérer les données venant d'un lien (URL).

Pour afficher la totalité des données contenues dans l'envoi, il est possible d'utiliser les fonctions `print_r($_GET)` ou `print_r($_POST)`.

### Print\_r

Affiche les informations lisibles pour une variable.

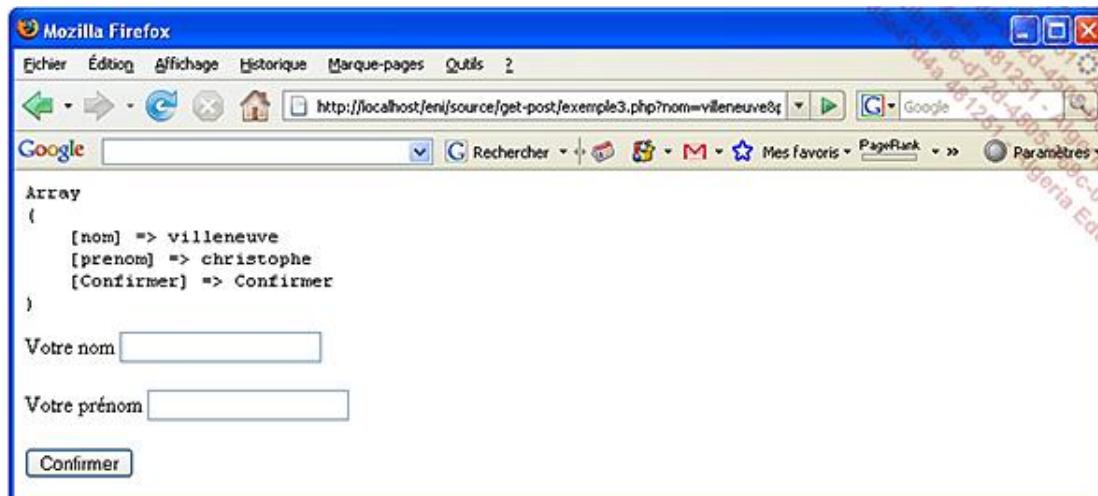
#### Nom de fichier

get-post/exemple3.php.

```
<?php
echo '<pre>';
print_r($_GET);
echo '</pre>';
?>
```

Cet exemple permet d'afficher tout le contenu de la variable `$_GET`.

En l'appliquant à l'exemple précédent, nous obtenons le résultat suivant :



## b. Envoi des données à partir d'un lien hypertexte

La méthode GET est aussi utilisée pour l'envoi de données à partir de liens hypertextes (URL).

Dans un petit exemple, qui nous sera utile pour notre application, nous allons étudier son principe :

get-post/exemple4.php

```
<?php
$message="Bonjour";
echo "<a href=exemple4-resultat.php?message=$message> Cliquer ici</a>";
?>
```

Cette page va envoyer le mot "Bonjour" dans une autre page Internet.

get-post/exemple4-resultat.php

```
<?php
if (isset($_GET['message']))
{
    echo "Message : ".$_GET['message']." <br />";
?>
```

```

    }
else
{
    echo "Pas de message <br />";
}
?>

```

Cet exemple vérifie si la valeur entrée a été détruite avec la fonction UNSET ou pas.

### isset

Détermine si une variable est affectée.

## c. Envoi des données à partir d'un fichier

Nous allons utiliser la variable `$_FILES` (qui nous sera utile pour notre application), par exemple pour envoyer une photo.

get-post/exemple5.php

```

<?php
if(!empty($_FILES))
{
    echo '<pre>';
    print_r($_FILES);
    echo '</pre>';
}
?>

```

Le script qui se trouve ci-dessus permet de vérifier si notre formulaire contient des données.



**empty** : détermine si une variable contient une valeur non nulle.

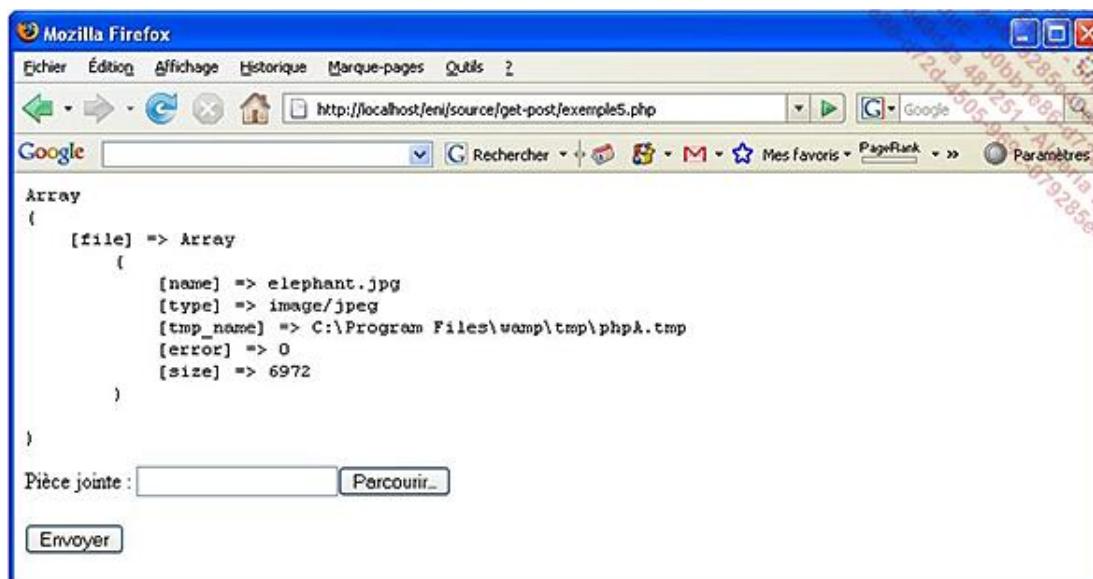
Nous afficherons le résultat des valeurs. Le tout sera envoyé à partir d'un formulaire HTML classique, comme ceci :

```

<form method="POST" action="exemple5.php" enctype="multipart/form-data">
    Pièce jointe : <input type="file" name="file" /><br /><br />
    <input type="submit" value="Envoyer" />
</form>

```

Voici le résultat si nous envoyons un fichier :



Il est impératif de mettre dans la ligne FORM le code suivant : `enctype="multipart/form-data"`. Si vous ne l'insérez pas dans votre formulaire le fichier ne pourra pas être envoyé.

## 2. Le contrôle des formulaires

Le contrôle des formulaires est un point important, c'est une étape qui se positionne entre la saisie et la mémorisation des données dans la base de données.

Nous allons utiliser comme base et comme description théorique le formulaire de saisie du nom et du prénom que nous avons créé dans la partie précédente.

Avant de valider notre formulaire, un certain nombre de points sont importants à vérifier car ce sont les bases de la sécurité et de la validité d'un formulaire.

Nous allons mettre en place un exemple pour chaque test, pour étudier comment se dérouleront les différents tests possibles, pour terminer avec la totalité des tests qui seront liés à notre application.

### a. Votre formulaire possède-t-il des valeurs ?

À la validation des données du formulaire, il faut vérifier que celui-ci contient bien des caractères dans les champs de saisie et que les champs de saisie n'ont pas été détruits entretemps. Nous disposons de deux fonctions.

#### EMPTY

Détermine si une variable contient une valeur nulle.

#### ISSET

Vérifie si la valeur entrée a été détruite avec la fonction UNSET ou pas.

Nous allons rendre obligatoire la saisie du champ prénom :

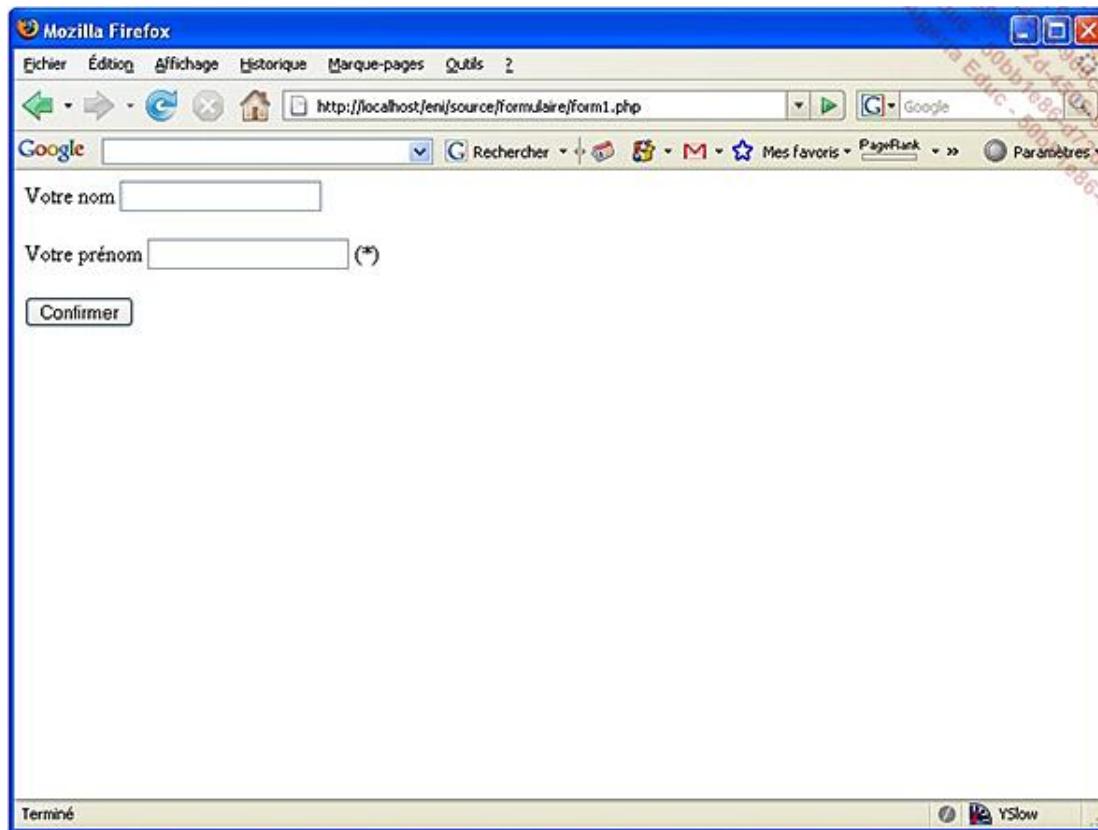
formulaire/form1.php

```
<?php
if (sizeof($_POST) > 0)
{
    if (empty($_POST['prenom']) && isset($_POST['prenom']))
    {
        $msg = "Champ obligatoire";
    }
    else
    {
        echo "Bonjour : ".$_POST['prenom']." ".$_POST['nom']."<br />";
    }
}
?>
```

Si le formulaire ne possède pas de valeur ou que la valeur a été détruite dans le champ prenom, nous allons arrêter le déroulement de l'application et signaler que ce champ est obligatoire.

```
<html>
<body>
<form name="saisie" method="POST" action="form1.php">
Votre nom <input name="nom" type="text"><br /><br />
Votre prénom <input name="prenom" type="text">
(*) <?php if (isset($msg)) echo $msg; ?> <br /><br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>
```

Lors du test du contenu du champ, nous mémoriserons le résultat dans \$msg pour afficher un message comme ci-après :



### b. Votre formulaire contient-il des caractères ?

Avant de valider le formulaire nous devons aussi penser à vérifier que notre formulaire ne possède pas de chaîne vide.

L'avantage d'effectuer ce test est de savoir s'il faut vraiment continuer à exécuter les différentes étapes de notre traitement. Ainsi, nous évitons de créer un enregistrement vide dans notre base de données.

formulaire/form2.php

```
<?php
if (sizeof($_POST) > 0)
{
    if (strlen($_POST['prenom'])<1)
    {
        $msg = "Le champ doit contenir au moins 1 caractère";
    }
    else
    {
        echo "Bonjour : ".$_POST['prenom']." ".$_POST['nom']."<br />";
    }
}
?>
<html>
<body>
<form name="saisie" method="POST" action="form2.php">
Votre nom <input name="nom" type="text"><br /><br />
Votre prénom <input name="prenom" type="text"> (*)
<?php if (isset($msg)) echo $msg; ?>
<br /><br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>
```

**strlen**

Calcule la taille d'une chaîne.



### c. Détection des caractères spéciaux

Nous allons garder notre formulaire de saisie avec le nom et le prénom et nous allons gérer la détection de caractères spéciaux permettant ainsi de mémoriser ces caractères dans notre base de données.

Ces caractères spéciaux sont les apostrophes ( ' ), les guillemets ( " ), les symboles suivants : < > &.

#### htmlspecialchars

Convertit les caractères spéciaux en entités HTML.

Ces caractères spéciaux sont utilisés par la fonction htmlspecialchars() pour obtenir les résultats suivants :

" & " (et commercial) devient " & ;

" " (guillemets doubles) devient " " lorsque ENT\_NOQUOTES n'est pas utilisé ;

" " (single quote) devient " ' " uniquement lorsque ENT\_QUOTES est utilisé ;

" < " (inférieur) devient " < ;

" > " (supérieur) devient " > ;

Pour afficher correctement le résultat de la saisie, nous allons utiliser la fonction stripslashes.

#### stripslashes

Supprime les anti-slash d'une chaîne.

Exemple :

```
<?php
$str = "Aujourd'hui il fait beau";
echo stripslashes($str);
?>
```

Voyons avec l'exemple ci-après comment utiliser ces deux fonctions :

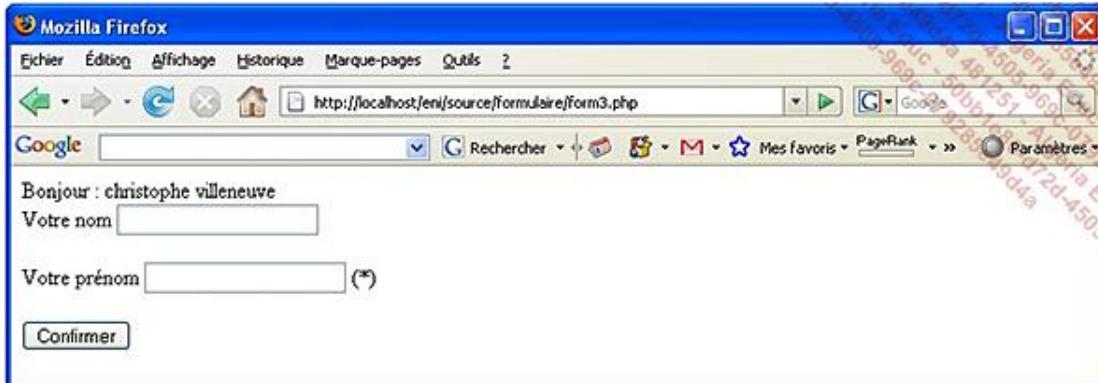
formulaire/form3.php

```
<?php
if (sizeof($_POST) > 0)
{
if (empty($_POST['prenom']) && isset($_POST['prenom']))
{
$msg = "Champ obligatoire";
if (strlen($_POST['prenom'])<1)
$msg="Votre champ doit comporter au moins 1 caractère";
}
else
{
```

```

$nom=htmlspecialchars($_POST['nom'], ENT_QUOTES);
$prenom=htmlspecialchars($_POST['prenom'], ENT_QUOTES);
echo "Bonjour : ";
echo stripslashes($prenom);
echo " ";
echo stripslashes($nom);
echo "<br />";
}
}
?>

```



➤ Si nous voulons gérer les accents et les caractères spéciaux en même temps, il est préférable d'utiliser la fonction **htmlentities()**. Nous verrons l'utilisation de cette fonction un peu plus loin.

#### d. Détection d'une chaîne de caractères

Dans le formulaire nous devons aussi vérifier si une chaîne de caractères que nous recevrons est un entier numérique.

Nous pouvons utiliser les deux fonctions suivantes : intval et is\_numeric :

##### intval

Retourne la valeur numérique entière équivalente d'une variable.

##### is\_numeric

Détermine si une variable est un type numérique.

formulaire/form4.php

```

<?php
if (sizeof($_POST) > 0)
{
    if ( is_numeric($_POST['codepostal']) && (intval(0 + $_POST['codepostal']) ==
$_POST['codepostal']))
    {
        echo 'Code postal : Saisie correcte <br>';
    } else {
        echo 'Code postal : Saisie incorrecte <br>';
    }
}
?>

```

Nous effectuons la saisie avec un formulaire HTML classique :

```

<html>
<body>
<form name="saisie" method="POST" action="form4.php">
Votre code postal <input name="codepostal" type="text"> <br />

```

```

<input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

## e. Vérification du format

Nous devons aussi penser à vérifier le format de certains champs saisis comme un e-mail, un lien hypertexte (URL), etc.

### htmlentities

Convertit tous les caractères éligibles en entités HTML.

Nous allons étudier comment faire pour vérifier une adresse e-mail. La vérification s'effectue en deux temps.

Nous allons convertir la chaîne de saisie au format de caractères HTML, ce qui nous permettra de traduire les caractères spéciaux, les lettres sous la forme d'accents, etc.

Ensuite nous allons effectuer un test sous la forme d'une expression régulière (cf. un peu plus loin dans ce chapitre - (section Expression régulière) c'est-à-dire en cherchant si certains caractères sont autorisés dans la chaîne de texte.

Ici, pour notre exemple, nous allons vérifier entre autres si @ est bien présent.

formulaire/form5.php

```

<?php
if (sizeof($_POST) > 0)
{
$email=htmlentities($_POST['email']);
if (!preg_match(''^[[:alnum:]][[-_.]?[[:alnum:]]]*@[[:alnum:]][[-_.]?[[:alnum:]]]*.
([a-z]{2,4})$',',$email))
{
    echo 'Votre email est incorrect !';
} else {
    echo 'Votre email est correct !';
}
?>
<html>
<body>
<form name="saisie" method="POST" action="form5.php">
Votre email <input name="email" type="text"> <br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

## f. Particularité avec les formulaires

Lorsque nous exécutons les différentes étapes de tests et que notre formulaire n'est pas validé à 100 %, nous devons réafficher les valeurs saisies dans le formulaire pour éviter que notre visiteur ressaisisse tous les champs avec le risque d'effectuer de nouvelles erreurs.

Reprendons notre exemple avec le nom et le prénom avec obligation que le champ prénom soit rempli.

### htmlentities

Cette fonction est identique à la fonction htmlspecialchars mais tous les caractères sont convertis en entités HTML.

formulaire/form6.php

```

<?php
if (sizeof($_POST) > 0)
{
    $nom=htmlentities($_POST['nom'], ENT_QUOTES, 'UTF-8');

```

```

$prenom=htmlentities($_POST['prenom'], ENT_QUOTES, 'UTF-8');

if (empty($_POST['prenom']) && isset($_POST['prenom']))
{
    $msg = "Champ obligatoire";
    if (strlen($_POST['prenom'])<1)
        $msg="Votre champ doit comporter au moins 1 caractère";
}
else
{
    echo "Bonjour : ";
    echo stripslashes($prenom);
    echo " ";
    echo stripslashes($nom);
    echo "<br />";
}
?>
<html>
<body>
<form name="saisie" method="POST" action="form6.php">
Votre nom <input name="nom" type="text"
value="<?php if (isset($nom)) echo addslashes($nom); ?>" /><br /><br />
Votre prénom <input name="prenom" type="text"> (*
<?php if (isset($msg)) echo $msg; ?>
<br /><br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

Nous attirons votre attention sur la syntaxe qui a été utilisée :

```
<input name="nom" type="text" value="<?php echo addslashes($nom); ?>" />
```

Les guillemets pour l'attribut VALUE signifient que si votre chaîne de caractères possède des espaces elle sera traitée entièrement. Sans cela toute la saisie ne serait pas prise en compte.

## g. Les listes dans les formulaires

Dans un formulaire de saisie, il est souvent nécessaire de vérifier le contenu des listes déroulantes pour être certain que la personne a bien sélectionné une des valeurs de la liste.

Nous allons maintenant voir comment contrôler ce champ. Nous prendrons par exemple une liste déroulante contenant Monsieur, Madame, Mademoiselle.

Pour effectuer le contrôle à l'aide d'un tableau, nous allons utiliser l'objet **array** qui permet la gestion de données sous la forme d'un tableau provisoire. Cet objet sera étudié plus en détail dans le chapitre suivant.

Nous allons considérer que les données sont dans le tableau et que nous avons besoin d'effectuer un test de vérification en effectuant une comparaison. Cette opération sera réalisée avec la fonction **in\_array** (et expliquée dans le chapitre suivant).

formulaire/form7.php

```

<?php
if (sizeof($_POST) > 0)
{
    $genre=array ('Monsieur','Madame','Mademoiselle');
    if (in_array ($_POST['genre'],$genre))
    {
        echo "Choix sélectionné ".$_POST['genre'];
    }
    else
    {
        echo "Champ non trouvé";
    }
}

```

```
?>
<html>
<body>
<form name="saisie" method="POST" action="form7.php">
    Faites votre choix
    <select name="genre">
        <option selected> </option>
        <option value="Monsieur">Monsieur</option>
        <option value="Madame">Madame</option>
        <option value="Mademoiselle">Mademoiselle</option>
    </select>
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>
```

Nous venons de découvrir de nombreuses techniques pour effectuer un contrôle de formulaire. Dans notre application, tous ces points seront réutilisés pour démontrer l'importance de ces tests.

 Ces tests de validation de formulaire touchent aussi à la sécurité des données. Il est important de lire la partie concernant la sécurité si vous ne voulez pas avoir de mauvaises surprises après la mise en ligne de votre application.

# Les tableaux

## 1. Les tableaux en général

Les tableaux dynamiques permettent, dans une même variable, de stocker plusieurs informations qui seront mémorisées et pourront être réutilisées ultérieurement.

Cette gestion dynamique est complètement différente des fonctions utilisées dans une table MySQL. Ce tableau (objet Array) correspond à une zone de stockage temporaire, qui est très utile pour l'utilisation et la validation des formulaires.

Ces tableaux sont souvent utilisés pour une gestion de caddie ou de panier mais aussi pour la mémorisation de messages d'alerte ou d'erreur dans certaines fonctions.

Les fonctions de gestion de tableau principales sont :

### array

Création d'un tableau.

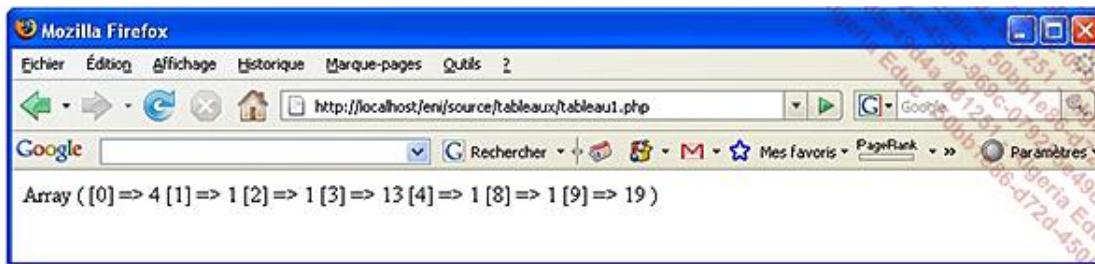
### in\_array

Retourne si une valeur appartient à un tableau.

tableaux/tableau1.php

```
<?php
$tableau = array( 4, 1, 1, 1, 1, 8=>1, 4=>1, 19, 3=>13);
print_r ($tableau);
?>
```

Voici le résultat :



Un tableau Array commence par la ligne 0 et ajoute 1 pour les lignes qui possèdent une valeur. Il est par ailleurs possible d'indiquer l'indice dans un tableau comme :

- l'indice 8 qui aura comme valeur 1 ;
- l'indice 4 qui aura comme valeur 1 ;
- l'indice 3 qui aura comme valeur 13.

Pour afficher l'indice numéro 4 de notre tableau :

```
<?php
$tableau = array( 4, 1, 1, 1, 1, 8=>1, 4=>1, 19, 3=>13);
echo $tableau[4];
?>
```

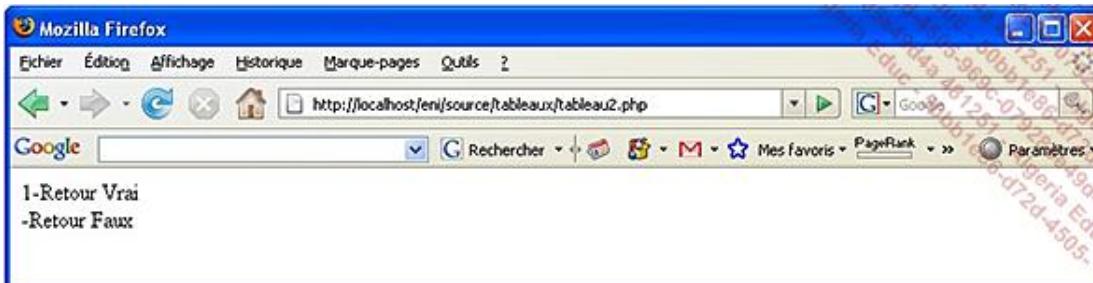
Le résultat qui sera retourné sera 1, puisque dans notre tableau la ligne 4 possède la valeur 1.

## 2. Les tableaux dans une fonction

Les tableaux offrent de nombreuses possibilités. Nous pouvons utiliser un tableau pour récupérer plusieurs valeurs venant d'une fonction.

Nous avons vu en début de ce chapitre (section Les bases du langage PHP - Les variables) que nous pouvions récupérer une valeur. Ici, nous allons récupérer plusieurs valeurs. Cette technique sera utilisée pour l'envoi (upload) de fichiers (cf. un peu plus loin dans ce chapitre - La gestion des fichiers).

Voici un exemple :



Pour obtenir ce résultat voici le code source :

tableaux/tableau2.php

```
<?php
function tableau($valeur)
{
    if ($valeur=="1")
        return array(TRUE,"Retour Vrai");
    else
        return array(FALSE,"Retour Faux");
}
$rep=tableau(1);
echo $rep[0]."-".$rep[1]."<br>";

$rep=tableau(2);
echo $rep[0]."-".$rep[1]."<br>";
?>
```

Nous avons effectué deux appels à la fonction **tableau**. Le premier envoie la valeur 1 et le deuxième la valeur 2.

La fonction va tester la valeur qu'elle reçoit et va stocker dans un tableau deux informations :

- Une valeur vrai ou faux qui peut servir de critère.
- Une valeur texte qui peut servir de message.

### 3. Les tableaux de messages dans un formulaire

Nous allons utiliser les tableaux pour gérer les messages dans un formulaire.

Gardons notre formulaire exemple comprenant un nom et un prénom et rendons obligatoires tous les champs de saisie.

Nous allons mémoriser les messages dans un tableau pour les afficher au bon endroit si notre visiteur ne remplit aucun des champs obligatoires. L'avantage de cette technique est de mémoriser tous les messages dans une seule variable.

tableaux/tableau3.php

```
<?php
if (sizeof($_POST) > 0)
{
    $erreurs = array();
    $msg = array();
    if (empty($_POST['nom']) && isset($_POST['nom']))
        $erreurs['nom'] = "Le champ Nom est obligatoire";
    if (empty($_POST['prenom']) && isset($_POST['prenom']))
        $erreurs['prenom'] = "Le champ Prenom est obligatoire";
}
```

```

{
    $erreurs['nom'] = true;
    $msg['nom'] = "Le Champ NOM est obligatoire";
}

if (empty($_POST['prenom']) && isset($_POST['prenom']))
{
    $erreurs['prenom'] = true;
    $msg['prenom'] = "Le Champ PRENOM est obligatoire";
}

if (empty($erreurs))
{
    $nom=htmlspecialchars($_POST['nom'], ENT_QUOTES);
    $prenom=htmlspecialchars($_POST['prenom'], ENT_QUOTES);

    echo "Bonjour : ";
    echo stripslashes($prenom);
    echo " ";
    echo stripslashes($nom);
    echo "<br />";
}
?>
<html>
<body>
<form name="saisie" method="POST" action="<?php echo $_SERVER['PHP_SELF']?>">
Votre nom <input name="nom" type="text" value="<?php if (isset($_POST['nom'])) echo
stripslashes($_POST['nom']); ?>" /> (*)
<?php if (isset($erreurs['nom'])) echo $msg['nom']; ?><br /><br />
Votre prénom <input name="prenom" type="text" value="<?php if (isset($_POST
['prenom']) echo
stripslashes($_POST['prenom']); ?>" /> (*)"
<?php if (isset($erreurs['prenom'])) echo $msg['prenom']; ?> <br /><br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

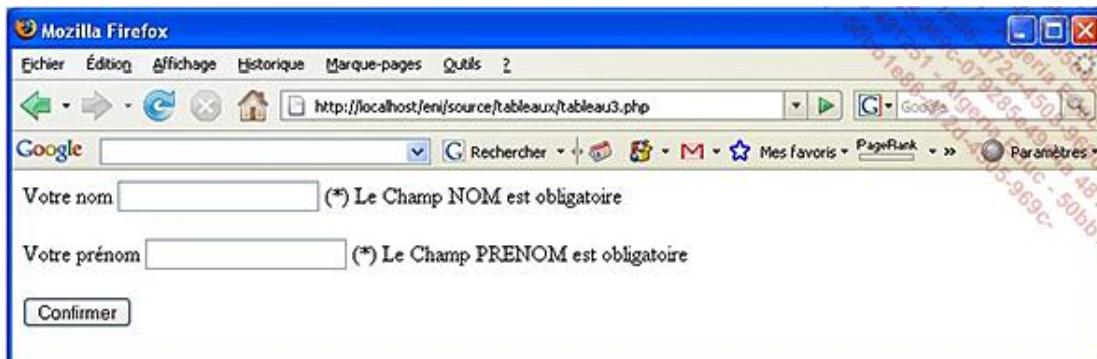
Pour la création d'un tableau à la volée nous devons déclarer un tableau vide pour permettre ainsi de stocker les informations les unes après les autres. Nous allons mémoriser dans notre tableau la variable nom et ensuite la variable prenom.

Bien sûr, si l'un des deux champs n'est pas rempli, l'application n'ira pas plus loin car notre tableau n'est pas NULL. Pour effectuer ce test, nous utilisons la fonction empty(\$erreurs)).

Si tout est correctement rempli, voici le résultat :



Si les champs ne sont pas remplis, voici le résultat :



Votre nom  (\*) Le Champ NOM est obligatoire

Votre prénom  (\*) Le Champ PRENOM est obligatoire

Confirmer

## 4. Rechercher dans un tableau

Quand les tableaux sont importants ou ne possèdent pas de points de repères, nous devons effectuer une recherche ligne par ligne pour vérifier si la valeur est présente ou pas.

Pour effectuer cette comparaison entre la valeur sélectionnée et le contenu du tableau nous allons utiliser `in_array`.

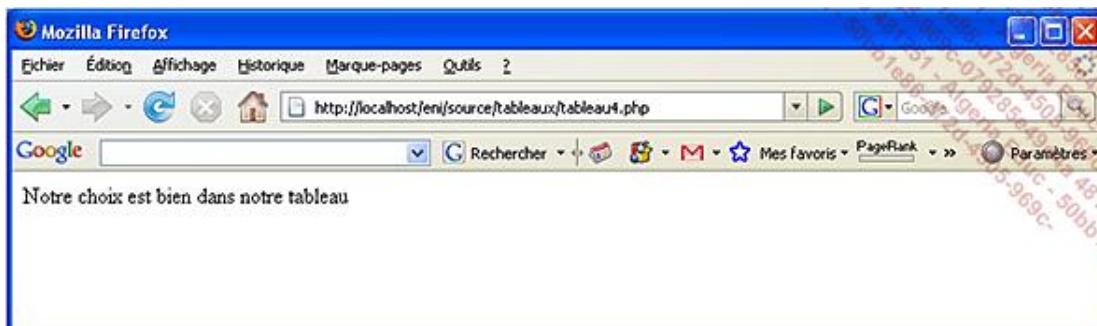
Reprenons l'exemple concernant l'utilisation d'une liste déroulante que nous avons créé dans la partie précédente.

tableaux/tableau4.php

```
<?php
$NotreChoix="Mademoiselle";

$genre=array ('Monsieur','Madame','Mademoiselle');
if (in_array ($NotreChoix,$genre))
{
    echo "Notre choix est bien dans notre tableau";
}
else
{
    echo "Champ non trouvé";
}
?>
```

Le résultat correspond :



Notre choix est bien dans notre tableau

Notre choix est bien dans notre tableau.

# Les bases de données

Après avoir validé et contrôlé le formulaire, nous devons maintenant enregistrer les données dans une table pour en garder une trace, mais aussi par la suite pouvoir les consulter et les modifier.

Trois possibilités s'offrent à nous :

- MySQLi : c'est la base de données évoluée de mysql et qui deviendra logiquement un standard car elle correspond à une évolution de mysql, avec de nombreuses autres possibilités.
- MySQL : c'est la base de données classique qui est actuellement très utilisée.
- PDO : il s'agit d'une interface permettant une connexion aux différentes bases de données du marché pouvant exister (MySQL, SQLite, PostgreSQL, Oracle...). Elle va permettre d'accéder à n'importe quelle base de données sans avoir à se préoccuper de savoir comment elle fonctionne réellement.

Nous étudierons chacune d'elles pour les adapter à notre application.

Dans une base de données, nous allons trouver une ou plusieurs tables décomposées en un ensemble de champs.

Ces champs déterminent les champs de saisie du formulaire, par exemple : le nom, le prénom, l'adresse, etc. Mais peuvent aussi mémoriser d'autres informations comme l'adresse de l'ordinateur (IP), le dernier passage de notre visiteur, etc.

## 1. MySQLi

MySQLi est une version évoluée de MySQL et pourrait à court terme remplacer MySQL avec des évolutions tant au niveau performances que nouvelles fonctionnalités (apparues depuis MySQL 4.1).

La base de donnée MySQLi est apparue avec PHP 5, mais va prendre plus d'importance avec l'arrivée de PHP 6 au niveau des gains.

Nous allons garder notre exemple avec les mêmes champs de saisies, c'est-à-dire le nom et le prénom.

Rappelons aussi qu'il est important d'effectuer un contrôle des données avant de les enregistrer dans la table (cf. dans ce chapitre - Le contrôle des formulaires).

### a. Ajouter des données dans une table

À partir du formulaire de saisie, nous allons remplir les champs nom et prénom. Après avoir appuyé sur le bouton **Confirmer**, nous allons nous connecter au serveur.

mysqli/exemple1.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

if (sizeof($_POST) > 0)
{
    $connex = mysqli_connect($serveur, $user, $passwd, $bdd);

    if (mysqli_connect_errno())
        die ("Echec de la connexion : ". mysqli_connect_error());
?>
```

Nous allons préparer notre requête SQL pour ensuite l'exécuter. Si tout s'est correctement déroulé, nous affichons un message de fin puis nous fermons la connexion.

```
<?php
$sql = "
INSERT INTO exemple (
    'nom'
```

```

        , 'prenom'
) VALUES (
    '".htmlspecialchars($_POST['nom'], ENT_QUOTES)."' 
    , '".htmlspecialchars($_POST['prenom'], ENT_QUOTES)."' 
) ";

$result = mysqli_query($connex, $sql);
if (!$result) die ("Probleme : " . mysqli_error($connex));
else
    echo "Ajout...oki";

mysqli_close($connex);
}
?>

```

### **mysqli\_error**

Retourne une chaîne décrivant la dernière erreur.

### **mysqli\_query**

Exécute une requête sur la base de données. Le résultat qui est renvoyé par la fonction permet de savoir si l'opération s'est correctement déroulée comme ceci on peut afficher le message correspondant au résultat.

### **mysqli\_close**

Ferme la connexion.

Le formulaire ci-après correspond au formulaire classique, il s'agit du formulaire HTML que nous avons déjà utilisé.

```

<html>
<body>
<form name="saisie" method="POST" action=<?php echo $_SERVER['PHP_SELF']?>>
Votre nom <input name="nom" type="text" /> <br /><br />
Votre prénom <input name="prenom" type="text" /> <br /><br />
    <input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

Cette fonction permet de recopier le chemin de provenance, ce qui permet lors de la validation du formulaire de recharger la page (cette fonction est détaillée dans le chapitre Fonctionnalités supplémentaires - Informations du serveur).

## **b. Visualiser les données d'une table**

Après avoir ajouté de nouvelles données, nous devons nous connecter au serveur de la base de données pour visualiser toutes ces données. Si ce n'est pas possible, nous le signalons comme ceci :

mysqli/exemple2.php

```

<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";
$connex = mysqli_connect($serveur, $user, $passwd, $bdd);
if (mysqli_connect_errno())
    die ("Échec de la connexion : ". mysqli_connect_error());
?>

```

Après avoir établi la connexion nous allons exécuter une requête SQL classique et afficher son résultat sous la forme d'un tableau.

```

< ?php
$sql="SELECT * FROM exemple";

```

```

$req=mysqli_query($connex,$sql);

echo "<table border=1>";
echo "<tr>";
echo "<td>N°</td>";
echo "<td>Nom</td>";
echo "<td>Prenom</td>";
echo "</tr>";

while ($row=mysqli_fetch_assoc($req))
{
    echo "<tr>";
    echo "<td>".$row['id']."</td>";
    echo "<td>".stripslashes($row['nom'])."</td>";
    echo "<td>".stripslashes($row['prenom'])."</td>";
    echo "</tr>";
}
echo "</table>";
?>

```

Nous libérons le résultat et fermons la connexion avec le serveur :

```

< ?php
mysqli_free_result($req);
mysqli_close($connex);
?>

```

### **mysqli\_fetch\_assoc**

Récupère une ligne de résultat sous la forme d'un tableau associatif.

### **mysqli\_free\_result**

Libère la mémoire associée à un résultat.

## **c. Mettre à jour des données dans une table**

Nous allons mettre à jour la ligne 2, comme ceci :

mysqli/exemple3.php

```

<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

$connex = mysqli_connect($serveur, $user, $passwd, $bdd);

if (mysqli_connect_errno())
    die ("Echec de la connexion : " . mysqli_connect_error());

$sql = "UPDATE exemple SET
        nom='Langage'
        ,prenom='PHP'
        WHERE id='2'
        ";

$result = mysqli_query($connex, $sql);
echo $result;
if (!$result) die ("Probleme : " . mysqli_error($connex));
else
    echo "Mise a jour...oki";
mysqli_close($connex);
?>

```

## d. Supprimer des données d'une table

Nous allons supprimer la ligne numéro 12 : après avoir établi la connexion avec notre serveur, nous allons préparer la requête en sélectionnant la ligne désirée et ensuite l'exécuter :

mysqli/exemple4.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

$connex = mysqli_connect($serveur, $user, $passwd, $bdd);
if (mysqli_connect_errno())
    die ("Echec de la connexion : ". mysqli_connect_error());
$sql = "DELETE FROM exemple WHERE id='12' ";
$result = mysqli_query($connex, $sql);
if (!$result) die ("Probleme : " . mysqli_error($connex));
else
    echo "Suppression...oki";
mysqli_close($connex);
?>
```

## e. Rechercher des données

Lorsque nous utilisons des données mémorisées dans une base de données, nous sommes souvent obligés d'effectuer une recherche pour les retrouver.

Pour effectuer une recherche dans une requête, nous utilisons les critères standard qui nous sont proposés, à l'aide de l'opérateur LIKE.

mysqli/exemple5.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

if (sizeof($_POST) > 0)
{
    $connex = mysqli_connect($serveur, $user, $passwd, $bdd);

    if (mysqli_connect_errno())
        die ("Echec de la connexion : ". mysqli_connect_error());

    $sql="SELECT * FROM exemple WHERE ('nom' LIKE '%".$_POST['motclef']."%' OR
'prenom' LIKE '%".$_POST['motclef']."%' ) ";

    $result = mysqli_query($connex, $sql);

    if (mysqli_num_rows($result) == 0)
    {
        echo "Nous n'avons pas trouvé de résultats";
    } else {
        while( $row=mysqli_fetch_object($result) )
        {
            echo $row['nom']." ".$row['prenom']."<br>";
        }
    }

    mysqli_free_result($result);
    mysqli_close($connex);
?>
```

```

}
?>
<html>
<body>

<form name="fsearch" method="POST" action="<?php echo $_SERVER['PHP_SELF']?>">
Rechercher : <input type="text" name="motclef">&nbsp;
              <input type="submit" name="action" value="Recherche">
</form>

</body>
</html>

```

### **mysqli\_fetch\_object**

Retourne la ligne courante d'un jeu de résultat sous forme d'objet.

### **mysqli\_free\_result**

Libère la mémoire associée à un résultat.

## **2. MySQL**

La base de données MySQL correspond actuellement au format le plus utilisé avec le langage PHP. Nous allons y consacrer quelques pages pour vous montrer comment réaliser les fonctions principales.

Bien qu'ici nous n'allons pas effectuer de tests, rappelons encore qu'il est important d'effectuer un contrôle des données avant leur insertion dans la table (cf. dans ce chapitre, Les formulaires - Le contrôle des formulaires).

### **a. Ajouter des données dans une table**

Utilisons de nouveau notre exemple de formulaire avec nom et prénom.

Nous devons dans un premier temps nous connecter au serveur. Nous allons déclarer le nom du serveur, l'identifiant de connexion, son mot de passe et le nom de la base de donnée avec la fonction mysql\_pconnect.

Si nous ne pouvons pas nous connecter au serveur, nous enverrons un message d'erreur.

mysql/exemple.php

```

<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

$connex = mysql_pconnect($serveur, $user, $passwd);
if (!$connex) die ("Impossible de se connecter : " . mysql_error());
?>

```

Après avoir établi la connexion entre notre navigateur et le serveur, nous allons insérer des données dans la table EXEMPLE, que nous avons créée précédemment et importée dans phpMyAdmin.

Nous allons préparer cette fonction en deux temps, d'abord en spécifiant la requête SQL et ensuite en l'exécutant.

```

< ?php
$sql = "INSERT INTO 'exemple' ('nom', 'prenom')
VALUES ('notre nom', 'notre prenom')";
$qid = mysql_query($sql);
if(! $qid )
  die ('Requête invalide : ' . mysql_error());
else
  echo "Insertion... oki";
?>

```

Lorsque nous avons exécuté la requête, nous devons fermer la connexion comme ci-après :

```
< ?php  
mysql_close($connex);  
?>
```

Nous affichons un petit message pour signaler que tout s'est correctement passé.

```
< ?php  
echo "Insertion... oki";  
?>
```

### **mysql\_query**

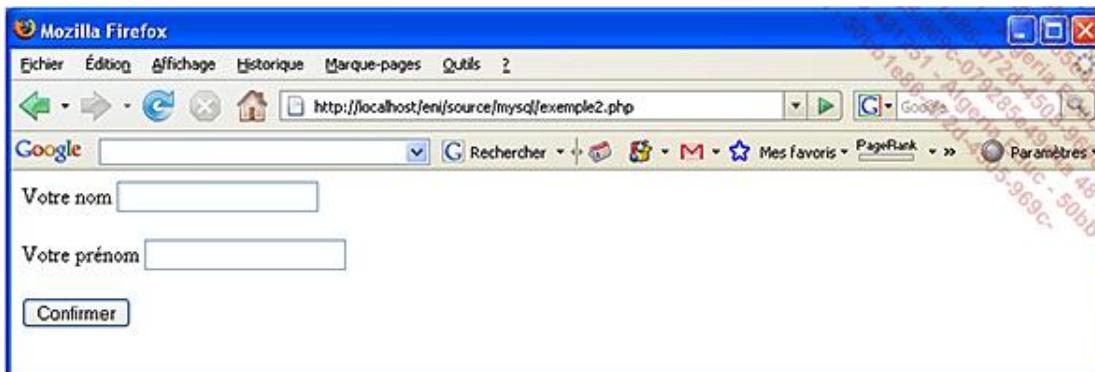
Envoie une requête à un serveur MySQL.

### **mysql\_close**

Ferme la connexion MySQL.

Lorsque le serveur a exécuté la fonction, il renvoie une information. Cette information est testée pour afficher un message de succès ou échec.

Maintenant nous allons voir comment insérer les données dans la table à partir de notre formulaire de saisie habituel :



Après validation du formulaire, voici les différentes étapes pour insérer les données dans la table :

mysql/exemple2.php

```
<?php  
$serveur = "localhost";  
$user= "root";  
$passwd = "";  
$bdd = "ouvrage";  
  
if (sizeof($_POST) > 0)  
{  
    $connex = mysql_pconnect($serveur, $user, $passwd);  
    if (!$connex) echo ("Impossible de se connecter : " . mysql_error());  
  
    $sql = "  
    INSERT INTO exemple (   
        'nom'  
        , 'prenom'  
    ) VALUES (   
        '" . htmlspecialchars($_POST['nom'], ENT_QUOTES) . "'  
        , '" . htmlspecialchars($_POST['prenom'], ENT_QUOTES) . "'  
    )";  
    $qid = mysql_query($sql);  
    if(! $qid ) die ('Requête invalide : ' . mysql_error());  
    mysql_close($connex);  
    echo "Insertion... oki";  
}
```

Notons que la requête SQL a été légèrement modifiée surtout dans la partie concernant les valeurs car nous avons incorporé la gestion de certains caractères (avec la fonction `htmlspecialchars`) comme nous l'avons vu précédemment.

## b. Visualiser les données d'une table

Après ajout des données, il est souvent nécessaire de vérifier leur présence dans la table. Nous allons procéder à l'affichage du contenu de notre table. Comme précédemment, nous allons nous connecter au serveur.

`mysql/exemple3.php`

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

$connex = mysql_pconnect($serveur, $user, $passwd);
if (!$connex) die ("Impossible de se connecter : " . mysql_error());
mysql_select_db($bdd,$connex);
?>
```

Puis, nous sélectionnons tous les champs de notre table pour en afficher le contenu complet dans un tableau HTML classique.

```
<?php
$sql="select * from exemple ";
$valeur=mysql_query($sql);
if( !$valeur ) echo "Problème dans la table exemple : " . mysql_error();

echo "<table border=1>";
echo "<tr>";
echo "<td>Nº</td>";
echo "<td>Nom</td>";
echo "<td>Prenom</td>";
echo "</tr>";

while( $list=mysql_fetch_object( $valeur ) )
{
    echo "<tr>";
    echo "<td>$list->id</td>";
    echo "<td>".stripslashes($list->nom). "</td>";
    echo "<td>".stripslashes($list->prenom). "</td>";
    echo " " "</tr>";
}
echo "</table>";
?>
```

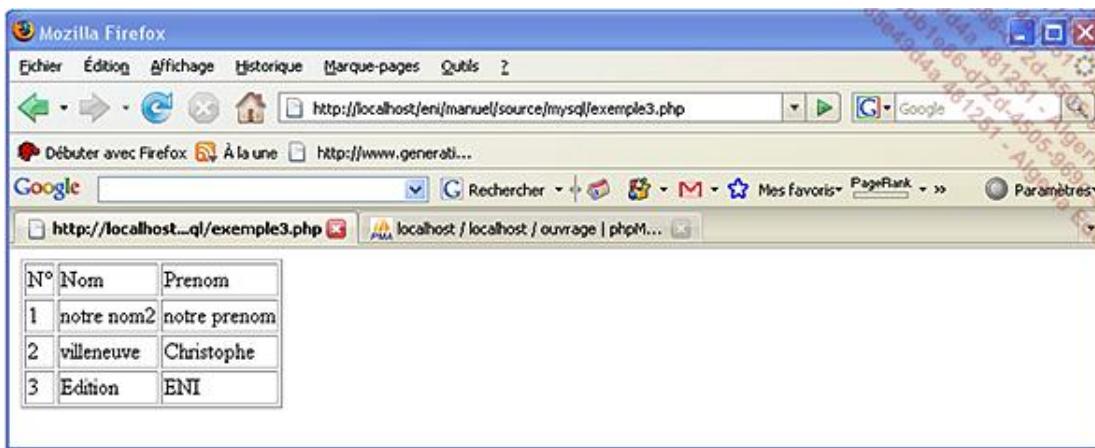
Puis, nous libérons et fermons la connexion entre le serveur http et celui de la base de données.

```
<?php
mysql_free_result($valeur);
mysql_close($connex);
?>
```



Notez l'utilisation de la fonction `stripslashes()`, qui permet de gérer les caractères spéciaux.

Voici le contenu de la table après exécution :



N°	Nom	Prenom
1	notre nom	notre prenom
2	villeneuve	Christophe
3	Edition	ENI

### c. Mettre à jour des données dans une table

Après avoir inséré et visualisé les données, nous ne devons pas oublier qu'il nous faudra les modifier et donc les mettre à jour.

Nous allons mettre à jour la ligne numéro 1, avec les informations :

- Nom : Langage
- Prenom : PHP

mysql/exemple4.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

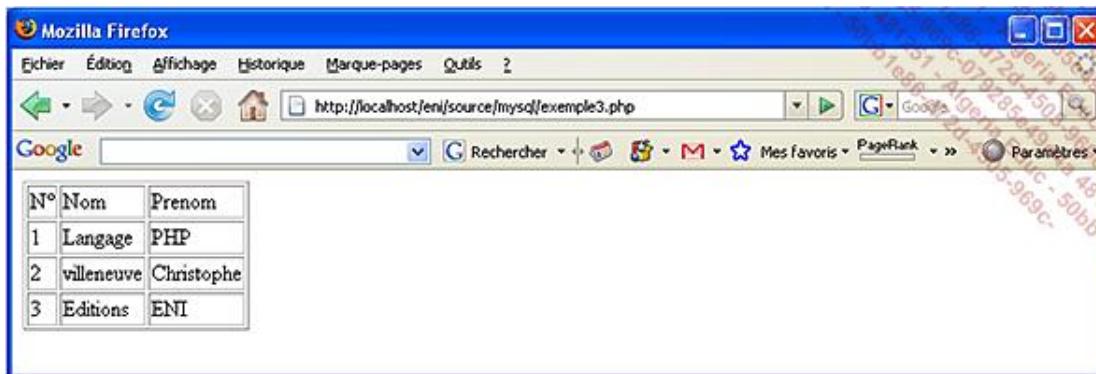
$connex = mysql_pconnect($serveur, $user, $passwd);
if (!$connex) echo ("Impossible de se connecter : " . mysql_error());
mysql_select_db($bdd,$connex);

$sql = "UPDATE exemple SET
    nom='Langage'
    ,prenom='PHP'
    WHERE id='1'
    ";

$qid = mysql_query($sql);
if(! $qid )
    die ('Requête invalide :'. mysql_error());
else
    echo "Mise à jour... oki";

mysql_close($connex);
?>
```

Voici le contenu de la table après la mise à jour de nos données :



#### d. Supprimer des données d'une table

Restons sur notre exemple pour supprimer la ligne id=1.

mysql/exemple5.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

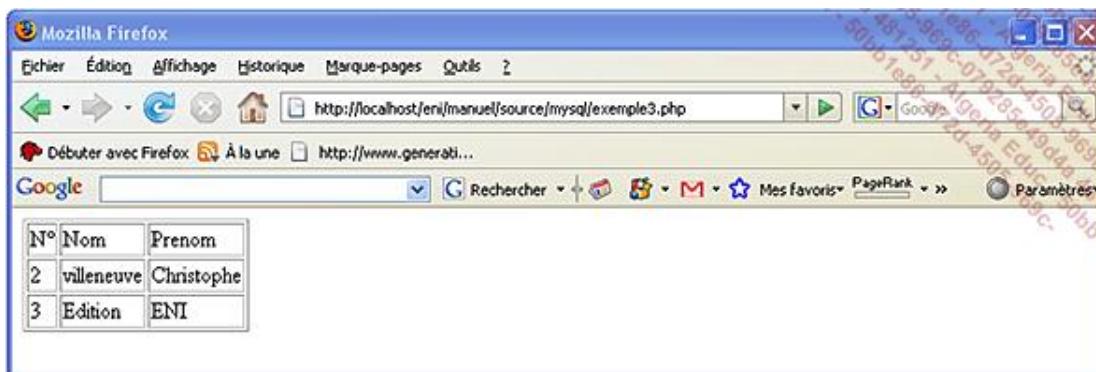
$connex = mysql_pconnect($serveur, $user, $passwd);
if (!$connex) echo ("Impossible de se connecter : " . mysql_error());

mysql_select_db($bdd,$connex);

$sql = "DELETE FROM exemple WHERE id='1' ";
$qid = mysql_query($sql);
if( ! $qid )
    die ('Requête invalide : ' . mysql_error());
else
    echo "Suppression... oki";

mysql_close($connex);
?>
```

Voici le nouveau contenu de notre table :



#### e. Rechercher des données

Pour effectuer une recherche de données dans une base de données, nous sommes obligés de passer par un formulaire. Lors de l'envoi des données à partir de ce formulaire, nous allons exécuter la recherche par l'intermédiaire d'une requête.

```
SELECT champ1, champ2, champ3, champ4, ...
FROM table
```

```
WHERE (champ1 LIKE '%".$_POST['motclef']."' AND champ2 LIKE '%".$_POST['motclef']."' ) ORDER BY champ1
LIMIT 1,30
```

**LIKE** : l'opérateur LIKE permet de faire une recherche dans un ou plusieurs champs en vérifiant si le mot est présent ou pas. Les symboles % encadrant le mot clé montrent que celui-ci peut être situé n'importe où dans le champ.

**LIMIT** : permet d'obtenir l'affichage d'un certain nombre de lignes et sur plusieurs pages. Nous verrons cela en détail dans le chapitre La saisie et l'affichage - Gérer plusieurs pages.

mysql/exemple6.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

if (sizeof($_POST) > 0)
{
    $connex = mysql_pconnect($serveur, $user, $passwd);
    if (!$connex) echo ("Impossible de se connecter : " . mysql_error());
    $sql="SELECT * FROM exemple WHERE ('nom' LIKE '%".$_POST['motclef']."' OR
'prenom' LIKE '%".$_POST['motclef']."' ) ";
    $resultat = mysql_query($sql);
    if (!$resultat) echo ('Requête invalide: ' . mysql_error());

    if (mysql_num_rows($resultat) == 0)
    {
        echo "Nous n'avons pas trouvé de résultats";
    } else {
        while($row = mysql_fetch_object($resultat) )
        {
            echo $row->nom." ".$row->prenom."<br>";
        }
    }
}

mysql_close($connex);
exit;
?>
<html>
<body>
<form name="fsearch" method="POST" action="<?php echo $_SERVER['PHP_SELF']?>">
    Rechercher : <input type="text" name="motclef">&nbsp;
    <input type="submit" name="action" value="Recherche">
</form>
</body>
</html>
```

Nous venons d'aborder les fonctions classiques que nous utilisons régulièrement et qui sont toujours d'actualité.

Concernant notre application, la connexion MySQL sera détaillée un peu plus loin dans l'ouvrage, mais nous resterons dans les mêmes approches et le même concept.

## 3. PDO

### a. Introduction

Le PDO (*PHP Data Objects*) permet de traiter la base de données comme un objet. Ces nouvelles fonctionnalités sont apparues avec PHP 5 et visent à pouvoir exploiter une base de données sans s'occuper de son format.

Le PDO permet, avec la même technique, d'utiliser les bases de données suivantes : DB2, MySQL, MySQLi, SQLite, ODBC, Oracle, PostgreSQL, etc.

En résumé, pour toutes les bases de données pouvant exister, il est possible de les insérer et de les utiliser en tant

qu'objet, donc avec PDO.

Avec certains systèmes d'exploitation (par exemple Windows), il est nécessaire d'effectuer une petite manipulation supplémentaire. Nous devons déclarer l'extension qui sera utilisée :

- Éditer le fichier php.ini.
- Effectuer une recherche de la chaîne « ; Windows Extensions ».
- Pour activer le mode PDO, il est nécessaire d'enlever le ; devant la ligne suivante : extension=php\_pdo.dll.
- Et aussi d'activer l'extension de la base de données, par exemple pour une base de données MySQL : extension=php\_pdo\_mysql.dll
- Si vous désirez utiliser une autre base de données par exemple MySQLi : extension=php\_pdo\_mysql.dll

Pensez à redémarrer votre environnement de développement pour que les modifications soient prises en compte. Si vous utilisez Wampserver dans l'environnement Windows, il faut le relancer.

Maintenant que le serveur peut utiliser le PDO, nous pouvons poursuivre.

## b. Connexion

Nous devons établir la connexion entre nos pages Internet et le serveur.

Ci-après quelques exemples de déclaration du serveur qui peuvent être utilisés avec des bases de données différentes :

### MySQL

```
$connex = new PDO ('mysql:host=localhost;dbname=baseDeDonnees', 'login',  
'MotDePasse');
```

### ODBC

```
$connex = new PDO ('odbc:baseDeDonnees', 'login', 'MotDePasse');
```

### ODBC (Access)

```
$connex = new PDO (odbc:Driver={Microsoft Access Driver (*.mdb)};  
Dbq=C:\\baseDeDonnees.mdb;  
Uid=login', 'login',  
'MotDePasse');
```

### PDO

Connexion à une base de données.

Voici un exemple d'utilisation de la connexion :

```
pdo/connection.php
```

```
<?php  
$serveur = "localhost";  
$user= "root";  
$passwd = " ";  
$bdd = "ouvrage";  
$port='3306';  
  
try
```

```

{
    $cnx = new PDO('mysql:host='.$serveur.';port='.$port.';dbname=
'. $bdd, $user, $passwd);
}

catch (PDOException $e)
{
echo 'N° : '.$e->getCode().'  
>';
die ('Erreur : '.$e->getMessage().'  
>');
}
?>

```

Nous avons intégré sur cet exemple la gestion d'erreurs sous forme d'exceptions que nous verrons un peu plus loin dans ce chapitre - section Le debugage - Erreur en Exception Etendue.

### c. Visualiser les données d'une table

Il existe différentes techniques pour exécuter une requête, nous allons privilégier la technique en deux temps :

- préparation de la requête,
- exécution de la requête.

Cette technique offre quelques avantages :

- optimisation des performances si la requête est appelée plusieurs fois,
- protection contre les injections SQL.

Nous allons être amenés à utiliser certaines fonctions pour affiche les données :

#### **setAttribute**

Définit un attribut pour une requête.

#### **prepare**

Prépare l'exécution d'une requête et retourne un objet qui la représente.

#### **execute**

Exécute une requête préparée.

#### **fetch**

Récupère la ligne suivante d'un jeu de résultat PDO.

#### **closeCursor**

Ferme le curseur, permettant à la requête d'être de nouveau exécutée.

Pour utiliser un exemple simple, nous devons commencer par effectuer une connexion avec le serveur de la base de données :

`pdo/visualiser.php`

```

<?php
$cnx = new PDO('mysql:host=localhost;port=3306;dbname=ouvrage', 'root', '');
$cnx->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>

```

Ensuite nous allons préparer la requête et l'exécuter :

```
<?php
$sql="SELECT nom,prenom from exemple ";
$qid=$cnx->prepare($sql);
$qid->execute();
?>
```

Lorsque la requête a été exécutée, nous pouvons afficher son résultat sous la forme d'un objet :

```
<?php
while($row=$qid->fetch(PDO::FETCH_OBJ))
{
    echo $row->nom.' '.$row->prenom.'<br />';
}
?>
```

Nous fermons le résultat puis la connexion :

```
<?php
$qid->closeCursor();
$cnx = null;
?>
```

#### **d. Ajouter des données dans une table**

Pour effectuer l'ajout de nouvelles données dans une table de notre base de données, nous devons respecter un certain ordre d'exécution.

Nous allons utiliser une autre fonction PHP :

##### **beginTransaction**

Démarre une transaction.

##### **commit**

Valide une transaction.

Cette fonction, nous permet de lancer une transaction entre le navigateur du visiteur et les données du serveur.

Dans notre exemple, nous effectuons une connexion avec un objet :

`pdo/insert.php`

```
<?php
$cnx = new PDO('mysql:host=localhost;port=3306;dbname=ouvrage', $user, $passwd);
$cnx->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

Avant de préparer notre requête, nous devons établir la connexion pour permettre de démarrer la transaction pour l'insertion des données. Ensuite nous préparons et exécutons la requête SQL :

```
<?php
$cnx->beginTransaction();
$sql="INSERT INTO exemple ('nom', 'prenom') VALUES ('Villeneuve', 'Christophe')";
$qid=$cnx->prepare($sql);
$qid->execute();
?>
```

Pour finir, la transaction est validée (commit) ce qui applique les modifications de la base de données :

```
<?php
$cnx->commit();
$cnx = null;
?>
```

## e. Mettre à jour des données dans une table

Après avoir ajouté des informations dans une table, nous devons effectuer des mises à jour des données.

Sur notre exemple habituel, modifions le contenu du champ nom de la ligne 3.

Pour effectuer cette manipulation, nous allons d'abord établir la connexion :

pdo/update.php

```
<?php
$cnx = new PDO('mysql:host=localhost;port=3306;dbname=ouvrage', $user, $passwd);
$cnx->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); // Obligatoire
pour la suite
?>
```

Nous démarrons la transaction pour préparer la requête puis l'exécuter :

```
<?php
$cnx->beginTransaction();
$sql="UPDATE exemple SET nom='ENI' WHERE id=3";
$qid=$cnx->prepare($sql);
$qid->execute();

?>
```

Lorsque la mise à jour est réalisée, nous validons la transaction et fermons la connexion :

```
<?php
$cnx->commit();
$cnx = null;
?>
```

## f. Nombre de lignes dans une table

Lorsque nous manipulons des informations provenant d'une table de la base de données, nous devons compter le nombre de lignes pour afficher le résultat ou pas. Le principe de compteur peut être utilisé en même temps que l'on prépare une requête, grâce au mot clef COUNT.

Après avoir établi la connexion avec le serveur (vu précédemment), nous préparons la requête avec un calcul :

### COUNT

Compte le nombre de lignes.

Pour ensuite exécuter la requête que nous avons préparée :

pdo/nombre.php

```
<?php
$sql="SELECT COUNT(*) from exemple ";
$compte = $cnx->prepare($sql);
$compte->execute();
?>
```

Nous pouvons utiliser la fonction fetchColumn pour connaître la valeur résultat de notre requête.

### fetchColumn

Retourne une colonne depuis la ligne suivante d'un jeu de résultats.

pdo/nombre.php

```
<?php
$resultat = $compte->fetchColumn();
echo $resultat."<br>";
```

```
?>
```

Nous devons fermer le curseur et fermer la connexion :

#### **closeCursor**

Ferme le curseur, permettant à la requête d'être de nouveau exécutée.

```
pdo/nombre.php
```

```
<?php
$compte->closeCursor();
$cnx = null;
?>
```

### **g. Afficher les données d'une table si elles sont présentes**

Nous allons effectuer avec une même requête SQL deux tâches différentes :

- Déterminer si des données sont présentes.
- Afficher le résultat si nous trouvons des données.

Nous devons nous connecter comme nous l'avons déjà vu dans les pages précédentes. Lorsque la connexion est établie, nous construisons notre requête SQL sous la forme ci-dessous :

```
pdo/update.php
```

```
<?php
$sql="SELECT nom,prenom from exemple ";
?>
```

Nous allons dans un premier temps déterminer si notre résultat possède des données à afficher avec une autre fonction en préparant et exécutant notre requête.

#### **fetchAll**

Retourne un tableau contenant toutes les lignes du jeu d'enregistrements.

```
<?php
$qid=$cnx->prepare($sql);
$qid->execute();
$nrow=$qid->fetchAll();
?>
```

La valeur que nous obtenons correspond au nombre de lignes possibles venant de notre requête.

Pour exécuter la requête, nous réutilisons la requête SQL que nous avons préparée précédemment et nous l'exécutons.

Pour afficher ce nouveau résultat, nous allons utiliser une boucle le traitant ligne par ligne.

#### **fetch**

Récupère une ligne suivante d'un jeu de résultats.

```
<?php
if ($nrow[0] !='')
{
    $qid=$cnx->prepare($sql);
    $qid->execute();
    while($row=$qid->fetch(PDO::FETCH_OBJ))
    {
        echo $row->nom.' '.$row->prenom.'<br />';
    }
}
```

```
    }
} else
{
    echo "Aucune rubrique disponible";
}
?>
```

Nous libérons le résultat et la connexion :

```
<?php
$qid->closeCursor();
$cnx = null;
?>
```

 Il existe d'autres fonctions et possibilités avec les bases de données en PDO, mais nous voyons ici les fonctions les plus couramment utilisées.

# Les sessions

## 1. Description

Les sessions permettent de mémoriser des informations sur plusieurs pages en PHP. Il est donc possible de suivre la navigation d'un internaute dans les pages du site Internet.

Chaque fois qu'un visiteur vient sur le site Internet et qu'il s'identifie, un identifiant de session lui est automatiquement attribué.

Les sessions sont aussi utilisées pour des espaces membres ou d'administration ; cela permet de vérifier si la personne qui veut accéder à un espace privé se trouve correctement identifiée.

Cependant les sessions sont aussi utilisées pour la gestion de caddies ou paniers pour les sites marchands.

Pour utiliser les sessions, il est nécessaire de recourir à certains paramètres. Au début de chaque page PHP, il faut appeler la fonction **session\_start()**. Elle permet de créer une session pour notre visiteur. Si la session existe déjà, elle ne changera pas.

### session\_start

Initialise une session.

Renseigner une session s'effectue de la façon suivante :

```
$_SESSION['votre variable'] = < votre texte > ;
```

Pour afficher le contenu d'une variable d'une session :

```
<?php  
echo $_SESSION['votre variable'] ;  
?>
```

Si l'utilisateur ne ferme pas son compte, ou bien son navigateur, la session sera automatiquement détruite par le serveur hébergeant le site Internet.

Si l'utilisateur ferme son compte quand il a terminé toutes les manipulations de son choix, on détruit et on libère la mémoire de la session comme ceci :

```
<?php  
session_destroy() ;  
unset($_SESSION) ;  
?>
```

### session\_destroy

Détruit une session.

### unset

Détruit une variable.

## 2. Utilisation de base des sessions

Nous allons placer dans une session un texte et la vérifier. Nous allons déclarer la session de départ.

```
session1/exemple1.php
```

```
<?php  
session_start();  
?>
```

Nous allons tester si notre session éditeur contient une valeur :

```

<?php
if (isset ($_SESSION['editeur'])
{
    echo ' Test 1 : Editeur est : ';
    echo $_SESSION['editeur']."<br>";
?>

```

Si la session est vide, nous allons signaler que la variable était vide. Nous allons ensuite remplir la session que nous avons déclarée et refaire un test pour vérifier :

```

<?php
} else { // si la session est vide
    echo 'Test 1 : Aucun editeur n\'est trouvé<br>';
    $_SESSION['editeur']="ENI";
    if (isset ($_SESSION['editeur']))
    {
        echo ' Test 2 : Editeur est : ';
        echo $_SESSION['editeur']."<br>";
    }
    else // sinon
    {
        echo 'Test 2 : Aucun editeur n\'est trouvé <br>';
    }
?>

```

Maintenant que nous avons terminé nos manipulations, nous pouvons détruire et libérer la session. Nous allons afficher la session pour être certain qu'elle soit vide :

```

<?php
session_destroy();
unset($_SESSION);
if (isset ($_SESSION['editeur']))
echo $_SESSION['editeur'] ;
?>

```

Nous obtenons le résultat suivant :



### 3. Un espace membre

Pour réaliser un espace membre (que nous utiliserons pour notre application) nous devons établir un espace sécurisé, permettant ainsi à l'administrateur et à toute personne possédant un compte d'accéder à la partie privée.

Il nous faut trois fichiers :

- un fichier **connex.inc.php** avec le code de connexion à la base de données,
- un fichier **index.php** comprenant le formulaire de saisie des codes d'accès,
- un fichier **resultat.php** qui montre un accès privé et par conséquent réservé aux membres.

Exceptionnellement, nous allons nous connecter au serveur de données "ENI" au lieu de "OUVRAGE" et nous allons utiliser la base de données au format MySQLi.

Par conséquent notre fichier de connexion va se présenter comme ceci :

```
<?php
$serveur = "localhost";
$user    = "root";
$passwd  = "";
$bdd     = "eni";

$connex = mysqli_connect($serveur, $user, $passwd, $bdd);
if (mysqli_connect_errno())
{
echo ("Echec de la connexion : ". mysqli_connect_error());
die ("Code d'erreur : ".mysqli_connect_errno());
}
?>
```

Nous allons déclarer dans le fichier index.php (ci-dessous) le début de session pour signaler que nous allons accéder à un espace permettant de filtrer les utilisateurs.

```
<?php
include "admin/connex.inc.php";
session_start();
?>
```

Ensuite, il faut gérer la partie du POST, c'est-à-dire l'envoi des données, qui permet de contrôler la saisie et d'autoriser ou pas l'accès à la partie membre :

```
<?php
if (sizeof($_POST) > 0)
{
    if (isset($_POST['login']))
    {
        $login=addslashes($_POST['login']);
        $password=MD5 ($_POST['password']);

        $sql="SELECT * FROM user WHERE login='".$login' AND password='".$password' ";
        $req=mysql_query($sql);
        if(! $req ) echo ('Requête invalide : ' . mysql_error());
        if (mysql_num_rows($req))
        {
            $row = mysql_fetch_object($req);
            $_SESSION['login'] = $row->login;
            $_SESSION['password'] = $row->password;
            $_SESSION['page'] = $row->page;

            header("Location:resultat.php");
        }
        else
            echo "Login ou mot de passe invalide";
    }
    else
    {
        header("Location:index.php?erreur=login");
        echo "erreur";
    }
}
?>
```

Dans notre fichier, plusieurs étapes s'enchaînent. Pour commencer nous contrôlons la validité de la saisie de notre formulaire.

Ensuite nous vérifions que les identifiants sont bien présents dans la base de données. Si tout est correct, nous mémorisons certaines informations dans notre session.

Nous allons ensuite rediriger le navigateur vers une autre page avec la fonction header.

## header

Permet d'aller sur une nouvelle page pour éviter que la page de provenance soit de nouveau modifiée.

Pour appeler ce code, nous avons besoin d'un formulaire de saisie comme celui-ci :

```
<html><body>
<center>

<form name="saisie" method="POST" action="index.php">
<table cellpadding="5" cellspacing="5" border=0 >
    <tr>
        <td align=center> Login<br>
        <input type="text" name="login">
    <br>
        <br>
        Mot de passe<br>
        <input type="password" name="password" size=20>
    <br>
        <div align="center">
            <input type="submit" name="action" value="Confirmer">
        </div>
    </td></tr>
    </table>
</form>
</center>
</body></html>
```

Après la saisie, les données sont envoyées par la méthode POST. Si tout est correct, nous pourrons accéder au fichier résultat.php, c'est-à-dire à l'espace membre ou à l'espace réservé correspondant aux codes d'accès.

session2/resultat.php

```
<?php
session_start();
?>
```

Nous allons mettre en place un petit filtre de session. Celui-ci va permettre de déterminer si l'identification a été effectuée.

Si la personne accède directement à cette page Internet, plusieurs cas sont possibles :

- La session est dépassée.
- Notre visiteur ne s'est pas identifié.

L'application renvoie directement au formulaire de saisie pour obliger le visiteur à s'identifier.

```
<?php
if (!isset($_SESSION['login']))
{
    header("Location:index.php");
    exit;
}
?>
```

Si tout s'est correctement passé, nous affichons quelques informations pour vérifier que tout est correct :

```
<?php
echo "Bienvenue dans votre espace membre : ".$_SESSION['login']."<br><br>";
echo "Votre login : ".$_SESSION['login']."<br>";
echo "Votre Mot de Passe : ".$_SESSION['password']."<br>";
echo "Direction vers la page : ".$_SESSION['page']."<br>";
?>
```

Nous détruisons puis libérons la session :

```
<?php
session_destroy();
unset($_SESSION);
?>
```

## 4. Limitation du temps des sessions

Lorsqu'une personne s'est connectée à notre site, surtout dans la partie sécurisée, elle peut laisser son navigateur ouvert et ne pas l'utiliser. Cette période de non-activité peut présenter un risque de vulnérabilité.

Toutefois, il est tout à fait possible de vérifier combien de temps cette page est restée non active en récupérant la durée de session fournie par le serveur.

Cette fonction retourne par défaut 1440 secondes, c'est-à-dire 24 minutes.

Pour récupérer ce type d'information, nous utilisons :

**session.gc\_maxlifetime**

Retourne en seconde la durée de vie des données sur le serveur.

**get\_cfg\_var**

Retourne la valeur d'une option de PHP.

```
session3/index.php
```

```
<?php
session_start();
$_SESSION['dernier_passage'] = time();
$_SESSION['duree'] = get_cfg_var("session.gc_maxlifetime");
echo "<a href='resultat.php'>Verifier la duree</a>"
?>
```

Nous venons de placer la durée de session et l'heure d'affichage de la page dans la variable de session.

Maintenant nous devons l'incorporer dans nos pages Internet pour permettre d'afficher ou pas la page demandée.

```
session/resultat.php
```

```
<?php
session_start();

if (time()-$_SESSION['dernier_passage']>$_SESSION['duree'])
{
echo "Session expirée";
exit();
}
else
{
    $_SESSION['dernier_passage'] = time();
    echo "Session... toujours valide";
}
?>
```

Dans notre exemple nous effectuons la différence entre l'heure d'affichage de la page et l'heure de l'affichage précédent de notre site et nous la comparons avec la durée en mémoire.

Si le temps est dépassé nous allons proposer à l'utilisateur de s'identifier une nouvelle fois.

S'il a changé de page, nous mémorisons l'heure du dernier passage pour permettre d'effectuer le même test sur une autre page.

# Les dates

Les dates sont utilisées pour obtenir de nombreuses informations :

- elles vont nous permettre de tracer un visiteur lors de ses différents passages,
- elles sont aussi utilisées par exemple pour les dates de naissance,
- elles nous permettent de gérer les différents formats pouvant exister, c'est-à-dire le format anglais, français, etc.

## 1. Affichage

Pour connaître le jour en utilisant une méthode classique, il existe la fonction DATE. Cette fonction permet d'afficher une date dans le format de notre choix.

Pour connaître la date d'aujourd'hui, voici un exemple :

date/date1.php

```
<?php
echo "Nous sommes le : ";
echo DATE("d-m-Y");
?>
```

Il est possible d'insérer quelques paramètres dans le champ date. Dans l'exemple :

- la lettre d affiche le jour,
- la lettre m affiche le mois,
- la lettre Y affiche l'année sur 4 caractères.

Il existe d'autres lettres possibles, par exemple pour afficher le jour de la semaine, le numéro de la semaine, l'heure et les minutes. Voici les symboles qui existent :

- s : secondes
- i : minutes
- H : heure 00 à 23 00
- I : indique si l'heure d'été est activée (1 = oui, 0 = non)
- O : différence d'heures avec l'heure GMT (Greenwich)
- d : jour du mois
- m : mois de l'année
- Y : année sur 4 chiffres
- y : année, sur 2 chiffres
- L : indique si l'année est bissextile (1 = oui, 0 = non)

- l : jour de la semaine (en anglais)
- F : mois écrit en clair
- t : nombre de jours dans le mois
- w : numéro du jour de la semaine 0 (dimanche) à 6 (samedi)
- W : numéro de la semaine dans l'année
- z : numéro du jour de l'année

Avec toutes ces possibilités, nous pouvons bâtir un petit exemple :

date/date2.php

```
<?php
echo "Nous sommes le : ".DATE("d-m-Y")."<br>";
echo "Il est ".DATE("l H \h i")." et ".DATE ("s")." secondes<br>";
?>
```

## 2. Format d'enregistrement

Concernant l'enregistrement des dates à partir d'un formulaire, nous pouvons utiliser la fonction `strftime()` de la façon suivante :

date/date3.php

```
<?php
echo "Nous sommes le : ";
setlocale(LC_TIME, "fr");
echo strftime("%d-%m-%Y");
?>
```

Dans notre application, nous allons utiliser cette fonction pour afficher la date au format français :

```
function datefr($ladate)
{
setlocale (LC_TIME, "fr_FR");
$date = strftime("%d-%m-%Y",strtotime($ladate));
return $date;
}
```

# Expression régulière

Une expression régulière permet de vérifier, rechercher, reconnaître ou remplacer une partie d'une chaîne de texte.

Elle permet avant tout de valider et filtrer les données que les utilisateurs envoient par les différents formulaires.

Cette partie est très importante car tous les programmes sérieux utilisent cette technique.

Nous allons utiliser un maximum d'exemples qui seront utiles pour notre application pour en faciliter la compréhension.

Avant de commencer, nous devons savoir qu'il existe différentes techniques pour vérifier une chaîne de caractère, soit par l'intermédiaire d'une expression régulière, soit avec les autres fonctions PHP qui existent comme les fonctions STRxxx. Ces fonctions sont plus simples à utiliser mais plus limitées dans nos applications surtout pour contrôler et valider une adresse de messagerie (e-mail).

Le principe d'utilisation de ces expressions est d'effectuer des tests avec des délimiteurs permettant ainsi de tester une ou plusieurs parties de la chaîne de caractères qui a été envoyée par un formulaire. Les délimiteurs sont variés mais nous utiliserons de préférence le symbole [ ] ou #.

Les expressions régulières qui sont à connaître sont :

- preg\_match,
- preg\_replace.

## 1. preg\_Match

Pour effectuer une recherche dans une chaîne de caractères en respectant exactement le texte, nous utiliserons preg\_match qui est une fonction très rapide en terme de temps de réponse par rapport aux autres fonctions qui réalisent le même travail.

expression/expl.php

```
<?php
$chaine = 'Un éditeur de qualité Edition ENI';
if (preg_match("Editions]", $chaine))
{
    echo "Vrai";
}
else
{
    echo "Faux";
}
?>
```

Nous obtenons le résultat Vrai.

Par contre, si le mot Edition était écrit en minuscule (edition), nous aurions obtenu Faux. Cela est dû au fait que le mot Edition possède un E majuscule. Pour éviter ce genre de retour et par conséquent ne pas se préoccuper des majuscules ou minuscules, il faut mettre la lettre i après l'expression.

expression/explb.php

```
<?php
$chaine = 'Un éditeur de qualité Editions ENI';

if(preg_match("[editions]i", $chaine))
{
    echo "Vrai<br>";
}
else
{
    echo "Faux<br>";
}
?>
```

L'exécution de notre exemple nous montre que le test nous répond Vrai.

Bien sûr, les expressions régulières possèdent un énorme intérêt. Elles permettent d'effectuer avec une même chaîne plusieurs tests en simultané.

Nous allons effectuer une recherche dans une phrase pour voir si nous trouvons les mots ENI et Hello.

Avec une méthode classique, voici ce qui peut être réalisé :

expression/exp2.php

```
<?php
    $chaine = 'Un éditeur de qualité Editions ENI';

    if (strpos($chaine, "ENI") !== FALSE && strpos($chaine, "Hello") !== FALSE)
    {
        echo "Vrai";
    }
    else
    {
        echo "Faux";
    }
?>
```

Avec une expression régulière :

```
< ?php
    if(preg_match("[ENI&Hello]", $chaine))
    {
        echo "Vrai";
    }
    else
    {
        echo "Faux";
    }
?>
```

Le résultat pour ces deux exemples est Faux, car le mot ENI est bien présent mais pas le mot Hello.

Maintenant nous allons évoquer les différentes possibilités de notre délimiteur :

& représente l'opérateur 'et' :

par exemple : preg-match("[Edition&ENI]","Un éditeur de qualité Editions ENI") ; -> Retourne Vrai

| représente l'opérateur 'ou' :

par exemple : preg-match("[Edition|ENI]","Un éditeur de qualité Editions ENI") ; -> Retourne Vrai

^ désigne le début d'une chaîne :

par exemple : preg-match("[^Edition]","Un éditeur de qualité Editions ENI") ; -> Retourne faux

\$ désigne la fin d'une chaîne :

par exemple : preg-match("[Edition\$]","Un éditeur de qualité Editions ENI") ; -> Retourne faux

[a-z] vérifie si c'est une chaîne de caractères :

par exemple : preg\_match("#[a-z]#","http://hello-design.fr/")-> Retourne vrai

preg\_match("#[a-z]#","01 23 45 67 89")->Retourne faux

[0-9] vérifie si c'est une chaîne numérique :

par exemple : preg\_match("#[0-9]#","01 23 45 67 89")->Retourne vrai

Voici le source de cet exemple :

expression/exp3.php

```
<?php
$chaine = 'Un éditeur de qualité Editions ENI';
echo fct_preg_match("#Edition#", $chaine). "<br>";
echo fct_preg_match("#edition#", $chaine). "<br>";
echo fct_preg_match("#edition#i", $chaine). "<br>";
echo fct_preg_match("#Edition|ENI#", $chaine). "<br>";
echo fct_preg_match("#^Un#", $chaine). "<br>";
echo fct_preg_match("#Edition$#", $chaine). "<br>";
echo fct_preg_match("#[a-z]#", $chaine). "<br>";

echo fct_preg_match("#[0-9]#", "01 23 45 67 89"). "<br>";
echo fct_preg_match("#[a-z]#", "01 23 45 67 89"). "<br>";
echo fct_preg_match("#[a-z]#", "http://hello-design.fr/"). "<br>";

$chaine = '1 éditeur de qualité Edition ENI';
echo fct_preg_match("#([a-z]{2,4})#", $chaine). "<br>";

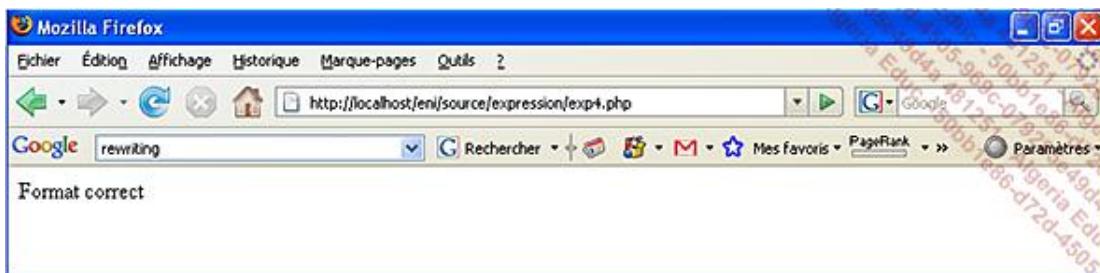
function fct_preg_match($critere, $chaine)
{
    if (preg_match($critere, $chaine))
    {
        echo $chaine. "...Vrai<br>";
    }
    else
    {
        echo $chaine. "...Faux<br>";
    }
}
?>
```

Pour notre application, nous allons utiliser cette fonction pour vérifier la validité des e-mails, c'est-à-dire pour vérifier si l'e-mail qui a été saisi correspond bien à un e-mail au format standard avec un @.

Nous allons utiliser l'exemple ci-dessous pour cela.

expression/exp4.php

```
<?php
$email="livre@hello-design.fr";
if (!preg_match('^[[:alnum:]]([-_.]?[[:alnum:]]*)@[[:alnum:]]([-_.]?[[:alnum:]]*)'.
([a-z]{2,4})$', $email))
{
    echo "Mauvais format d'Email";
}
else
{
    echo "Format correct";
}
?>
```



Dans cette expression régulière, de nouveaux symboles sont utilisés :

**[[alnum]]**

Accepte les caractères alphabétiques et numériques.

?

Caractère joker qui permet d'accepter toute sorte de chaîne.

## 2. preg\_replace

Le remplacement de certains caractères dans une chaîne est une fonctionnalité très utile. Le descriptif de la fonction utilisée est :

### **preg\_replace**

Rechercher et remplacer par une expression relationnelle standard.

Nous allons mettre le texte Edition ENI en gras à partir d'une expression régulière :

expression/exp5.php

```
<?php
$texte="[b]Editions ENI[/b]";
$cherche=array('/\[\//','\]\//');
$remplace=array('<','>');
echo preg_replace ($cherche,$remplace,$texte);
?>
```

Cette fonction permet de convertir en une seule fois plusieurs caractères qui sont présents dans une chaîne de caractères.

Dans notre exemple, nous avons remplacé les crochets par des signes supérieurs et inférieurs pour obtenir :  
<b>Editions ENI</b>

# La gestion des fichiers

## 1. Lecture/Ecriture

Actuellement il existe différentes manières pour lire et écrire dans un fichier informatique. Nous allons en voir deux : la première correspond à une utilisation standard que la majorité des développeurs utilisent, la deuxième méthode consiste surtout à exploiter les possibilités des versions PHP 4.3.x, PHP 5 et supérieur.

### a. Première méthode

Nous allons écrire dans un fichier texte le message suivant « Bonjour des Editions ENI ».

Les fonctions utilisées sont :

**fopen**

Ouverture d'un fichier.

**fputs**

Écrit un fichier en binaire (alias de fwrite).

**fclose**

Ferme un fichier.

fichiers/exemple1.php

```
<?php
$textel="Bonjour des Editions ENI";
$fp = fopen("exemple.txt", "a");

fputs($fp,$textel);
fclose ($fp);
?>
```

Nous allons lire maintenant le contenu du fichier que nous venons d'écrire avec les fonctions suivantes :

**fgets**

Renvoie la ligne courante du fichier.

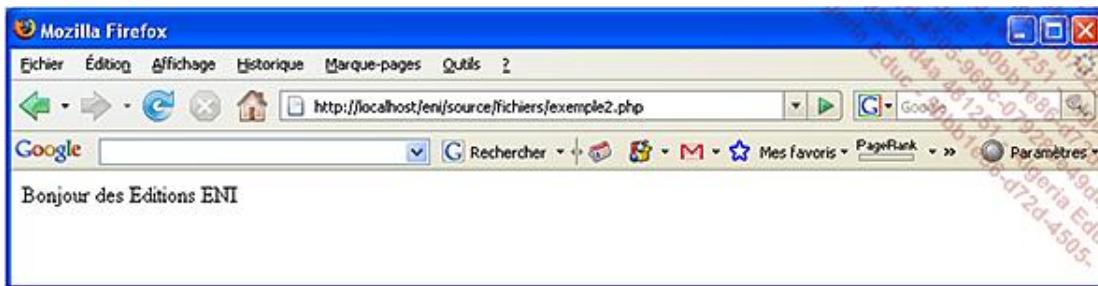
**feof**

Teste la fin du fichier.

fichiers/exemple2.php

```
<?php
$fp = fopen("exemple.txt", "r");
while(!feof($fp))
{
    $variable= fgets($fp,4096);
}
echo $variable;
fclose ($fp);
?>
```

Voici le résultat obtenu :



## b. Deuxième méthode

À partir de la version 5 de PHP, la fonction **file\_put\_contents** permet d'écrire en une seule fois le contenu d'une variable dans un fichier de son choix. Cette fonction revient en résumé à effectuer un fopen suivi d'un fwrite et d'un fclose.

### file\_put\_contents

Écrire une chaîne dans un fichier.

fichiers/exemple3.php

```
<?php
$textel="Bonjour des Editions ENI";
$fichier =  file_put_contents('exemple2.txt',$textel);
?>
```

À partir des versions 4.3 de PHP et aussi en PHP 5, la fonction **file\_get\_contents** permet d'effectuer en une seule étape la récupération du contenu du fichier dans une seule variable.

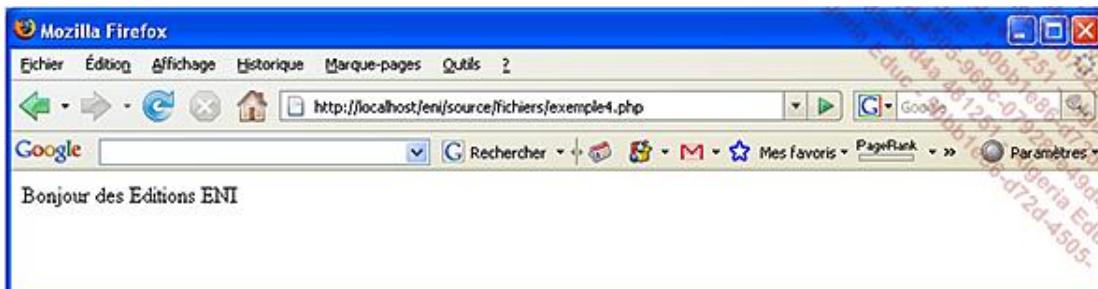
### file\_get\_contents

Lit tout un fichier dans une chaîne.

fichiers/exemple4.php

```
<?php
$fichier = file_get_contents('exemple2.txt');
echo $fichier;
?>
```

Voici le résultat obtenu :



 Nous allons privilégier cette deuxième méthode pour réaliser notre application.

## 2. Transfert de fichiers

Dans une application informatique, il n'est pas rare de mettre en place un système d'envoi de fichiers vers le serveur, par exemple une photo, un fichier, une documentation, etc.

Pour notre application. Nous allons envoyer dans un dossier "images" une photo de la personne membre qui se

trouve dans le carnet adresse sur notre serveur.

Pour commencer, il faut que nous préparions un formulaire pour permettre à nos visiteurs d'envoyer un fichier.

fichiers/upload.php

```
<html>
<body>
<form enctype="multipart/form-data" action="upload_execution.php" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="500000" >
Choisir un fichier : <input name="fichier" type="file" >
<input type="submit" name="action" value="Envoi" >
</form>
</body>
</html>
```

Notre formulaire possède deux petits détails qui sont importants :

- Une case cachée qui contient la taille maximum du fichier à envoyer. Ici nous allons la limiter à 500 Ko.
- Dans FORM, nous avons spécifié l'attribut `enctype="multipart/form-data"` qui permettra d'effectuer le transfert. Si vous ne l'insérez pas, notre formulaire ne pourra pas effectuer l'envoi du fichier sur notre serveur.

Il est possible d'insérer plusieurs champs de sélection pour envoyer des fichiers. Le langage PHP gère l'envoi multiple depuis la version PHP 3.0.10.

Nous allons utiliser comme fichier image l'ELEPHANT symbole du PHP qui sera disponible en téléchargement. Il est tout à fait possible de trouver cette illustration et le symbole du PHP, disponibles sur le site de l'auteur (voir à la fin de l'ouvrage pour connaître le site Internet).

Lors de la confirmation de notre formulaire, plusieurs étapes sont nécessaires pour le traitement de notre fichier.

- Nous devons vérifier que le formulaire a été correctement rempli.
- Si c'est le cas, nous allons vérifier si le dossier de réception (upload) existe bien.
- Si ce n'est pas le cas, il faut le créer et en autoriser l'accès en écriture.

Nous utiliserons les fonctions :

**is\_dir**

Indique si le dossier existe.

**mkdir**

Crée un dossier.

**chmod**

Change le mode du fichier.

fichiers/upload\_execution.php

```
< ?php
$destDir = "upload";
if (sizeof($_POST) > 0 && $_FILES['fichier'] && $_FILES['fichier'] != "none")
{
?>
```

Nous vérifions si des données ont été envoyées par un formulaire et qu'il existe un nom de fichier dans le champ de saisie.

Nous vérifions si le dossier de réception existe bien, sinon nous devons le créer et lui ouvrir les droits d'écriture.

```

<?php
if (!is_dir($destDir))
{
    if (!@mkdir($destDir))
    {
        die ("Erreur lors de la création du dossier $destDir");
    }
}
@chmod($destDir, 0777);
?>

```

@

Permet de bloquer l'affichage des messages d'erreurs lorsque la fonction est appelée.

Pour autoriser l'accès en écriture et en lecture sur un dossier, nous utiliserons la fonction CHMOD (« 777 »). Quand nous sommes sur un serveur local, cette fonction n'est pas utile. Par contre, elle devient nécessaire quand il s'agit d'un serveur de production. Cette action peut être réalisée "à la main" avec un logiciel de FTP.

Nous allons tester si le fichier à transférer existe et qu'il ne dépasse pas la taille permise lorsque nous l'envoyons à partir de notre formulaire. Notre serveur teste d'abord si le fichier existe :

### **file\_exists**

Vérifie si le fichier existe.

```

<?php
if (!file_exists($_FILES['fichier']['tmp_name']))
{
    die ("Le fichier n'est pas passé. Vérifier les critères");
}
?>

```

Nous allons effectuer ensuite un autre test sur la taille du fichier pour être sûrs que notre fichier ne dépasse pas la taille maximum.

### **filesize**

Renvoie la taille d'un fichier.

```

<?php
$taille_max=$_POST['MAX_FILE_SIZE'];
$taille_fichier = filesize($_FILES['fichier']['tmp_name']);
if  ($taille_max && ($taille_fichier > $taille_max))
{
    die ("La taille est trop importante");
}
?>

```

Nous allons également effectuer un test sur l'extension de notre fichier pour vérifier qu'il s'agit bien d'un fichier image. Nous en profitons pour imposer cette extension en minuscule.

 Une petite remarque importante : il est préférable d'avoir un nom de fichier en minuscule car de nombreux hébergeurs se trouvent sous des environnements et systèmes d'exploitation faisant la distinction entre les majuscules et minuscules, par exemple les environnements Linux ou Unix.

### **strrchr**

Trouve la dernière occurrence d'un caractère dans une chaîne.

```

<?php
$ext = strrchr($_FILES['fichier']['name'], '.');
$ext = substr($ext, 1);
$ext = strtolower($ext);
if ($ext!="jpg" && $ext!="jpeg" && $ext!="png" && $ext!="gif" )
{

```

```
    die("Le fichier n'est pas une image");
}
?>
```

Maintenant, nous allons effectuer une correction sur le nom du fichier.

Toujours dans l'exemple de notre chapitre actuel, nous utilisons le nom original de l'image qui correspond au nom de fichier que notre visiteur aura envoyé à partir d'un formulaire de saisie, c'est-à-dire si nous regardons dans le dossier de réception du serveur, nous allons trouver le même fichier. Par contre, cela peut poser quelques problèmes suivant la syntaxe de ce nom.

Avec notre exemple, nous pouvons rencontrer des noms de fichiers possédant des caractères sous la forme d'accents, des espaces, des caractères spéciaux, majuscules.

**strtr**

Remplace des caractères dans une chaîne

Pour valider le nom de notre fichier, nous allons effectuer trois opérations :

- Le remplacement des accents avec la fonction `strtr`.
  - Une vérification des autres caractères avec la fonction `preg_replace` (cf. section Expression régulière).
  - La conversion en minuscule du nom de fichier n'est pas une obligation, mais sur certains serveurs ou environnements, il est préférable de le faire pour un meilleur fonctionnement.

```
<?php
$fichier_destination = strtr($_FILES[fichier]['name'],
'ÀÁÃÃÄÃÇÈÉËÌÍÌÌÖÐÖÓÖÙÙÙÙÀââââçéâéâíííöððöööùùüýÿ',
'AAAAAAACEEEEIIIIOOOOOUUUUYaaaaaaceeeeiiiioooooouuuuyy');

$fichier_destination = preg_replace(
    '/[^a-zA-Z0-9\.\\$\\%\\'\\'\\-\\@\\{\\}\\~\\!\\#\\(\\)\\&\\_\\^]/'
    , '' , str_replace(array(' ','%20') , array('_','_') , $fichier_destination));

$fichier_destination=strtolower($fichier_destination);
?>
```

Ceci fait, il ne nous reste plus qu'à déplacer le fichier dans le bon dossier :

```
<?php
if (move_uploaded_file($_FILES['fichier']['tmp_name'], $destDir.$fichier_destination))
{
    die ("Le fichier est correctement passé");
}
else
{
    echo "Probleme de transfert";
}
?>
```

## move\_uploaded\_file

Déplace un fichier téléchargé.

Lorsque le fichier est correctement transféré, nous pouvons remettre le CHMOD comme avant : CHMOD (« 755 »).

# La sécurité

La sécurité d'un site Internet devient un des aspects principaux et sera toujours une des priorités pour un bon programmeur.

Le manque de sécurité peut être souvent lié à des erreurs de programmation. Nous allons voir quelques techniques qui nous seront utiles lors du développement de notre application.

 Pour information, nous ne verrons pas comment effectuer les différents types d'attaques pouvant exister, mais juste le nécessaire pour les éviter.

## 1. Les sessions

Pour sécuriser un peu plus l'utilisation des sessions, c'est-à-dire fixer une session pour définir un utilisateur légitime, nous allons attribuer une clef d'initialisation variable.

Utiliser cette clef permet, lorsque le langage PHP initialise les sessions, de vérifier si un identifiant a été transmis et s'il est connu. Si ce n'est pas le cas, un nouvel identifiant sera créé.

securite/session.php

```
<?php
session_start();
if (!isset($_SESSION['initialisation']))
{
    session_regenerate_id();
    $_SESSION['initialisation']=time();
}
echo $_SESSION['initialisation'];
?>
```

Nous pouvons aussi mémoriser l'adresse IP de l'ordinateur de notre visiteur pour sécuriser un peu plus les sessions et les provenances de nos visiteurs.

## 2. Droit d'accès à certaines pages

Le droit d'accès à certaines pages permet à certaines personnes reconnues d'accéder à des pages désignées. Souvent, ces pages sont destinées aux administrateurs ou aux personnes qui sont responsables du site pour le contrôler.

Si un visiteur veut accéder à une page pour laquelle il ne possède pas les droits accès, il faudrait le renvoyer vers une autre page. Cette éventualité existe si, par exemple, ce visiteur possède un compte sur le site et qu'il reçoit de notre part sous la forme d'un lien l'URL l'accès à une page qu'il ne devrait pas connaître. Si le visiteur clique sur ce lien et que nous ne mettons pas un filtre sur le niveau d'accès, il peut alors utiliser ces pages et provoquer des dommages.

```
<?php
if ($_SESSION['niveau']!="admin")
    header("Location:index.php");
?>
```

Ces quelques lignes sont à insérer en haut de votre page Internet. Si le visiteur ne possède pas les droits nécessaires, il sera automatiquement renvoyé à la page index du site.

## 3. Limitation de la durée des sessions

Comme nous l'avons vu précédemment, lorsqu'un visiteur se connecte sur une partie sécurisée du site (son compte par exemple) il peut laisser son navigateur ouvert sans l'utiliser. Cette période de non-activité peut présenter un risque de vulnérabilité.

Toutefois, il est tout à fait possible de modifier la durée des sessions. Ainsi, si le visiteur ne réalise aucune action et que ce délai est dépassé, nous pouvons le rediriger vers le formulaire d'identification.

Nous allons mémoriser dans notre session l'heure de passage de notre visiteur et la durée que nous lui autorisons pour changer de page.

session/dureesession.php

```
<?php
session_start();
$_SESSION['dernier_passage']=time();
$_SESSION['duree']=5 * 60;
?>
```

Dans l'exemple ci-dessus nous mémorisons la durée en seconde et nous lui accordons 5 minutes ce qui lui laisse 300 secondes pour changer de page. Cette durée peut être plus importante ou réduite.

Il est ensuite nécessaire pour chaque page d'effectuer un test supplémentaire pour vérifier si la session est toujours d'actualité et n'est pas expirée.

session/dureesession\_result.php

```
<?php
session_start();
if (time()-$_SESSION['dernier_passage']>$_SESSION['duree'])
{
echo "Session expirée";
exit();
}
else
    $_SESSION['dernier_passage'] = time();
?>
```

Dans notre exemple nous effectuons la différence entre l'heure d'affichage de la page et l'heure de l'affichage précédent de notre site et nous la comparons avec la durée en mémoire.

Si le temps est dépassé nous lui proposons de s'identifier une nouvelle fois.

Si notre visiteur a changé de page, nous mémorisons l'heure du dernier passage pour permettre d'effectuer le même test sur une autre page.

## 4. XSS

La faille XSS (*Cross Site Scripting*) connu aussi sous le nom CSS (*Cross-Site Scripting*) exploite les vulnérabilités d'une page Web dynamique. Le X représente Cross, car cela peut porter à confusion avec CSS (*Cascading Style Sheets*).

Cela est dû, entre autres, à l'absence de contrôles lors de la validation d'un formulaire ou d'un lien Internet. Pour sécuriser cette partie, nous vous conseillons d'utiliser les fonctions htmlspecialchars et htmlentities.

Ces deux fonctions sont presque identiques, mais nous utiliserons surtout la fonction **htmlentities** qui offre plus de possibilités de contrôle.

securite/xss.php

```
<?php
$titre="Un petit exemple <script>alert ('Editions ENI')</script>";
echo $titre."<br>";
?>
```

Nous obtenons l'affichage d'un message comme ceci :

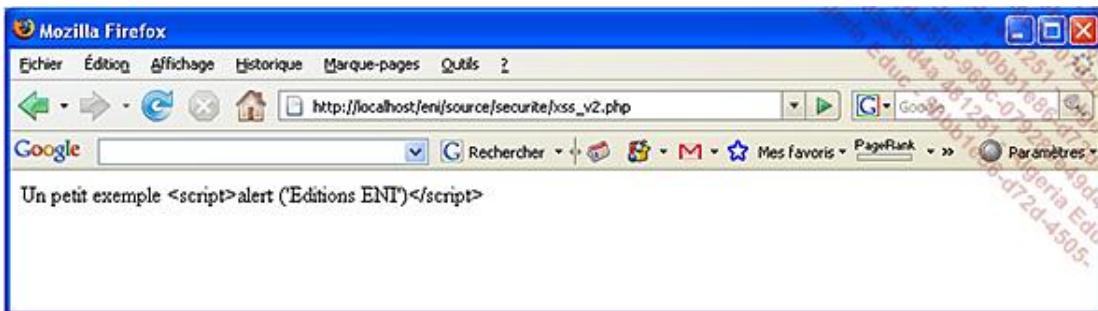


Pour contrecarrer ce genre d'attaque, voici la solution :

```
securite/xss-v2.php
```

```
<?php
$titre="Un petit exemple <script>alert ('Edition ENI')</script>";
echo htmlentities($titre). "<br>";
?>
```

Ce qui donne :



## 5. Injection SQL

Les injections sur Internet sont un aspect à ne pas négliger. Il est très dommage de constater un matin quelques dysfonctionnements sur votre site ou même qu'il a complètement changé de présentation.

```
securite/injectionsql.php
```

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";

$connex = mysql_pconnect($serveur, $user, $passwd);
$champ="Aujourd'hui, il fait beau";
echo mysql_real_escape_string($champ);
?>
```

Les injections SQL sont liées à la connexion vers la base de données. La fonction **mysql\_real\_escape\_string** permet d'ajouter une barre oblique devant les simples quotes.

Pour notre application, nous allons nous en servir pour la connexion à notre compte pour éviter d'avoir des perturbations dans notre gestion des utilisateurs.

```
<?php
$login=mysql_real_escape_string(htmlentities($_POST['login'], ENT_QUOTES));
?>
```

## 6. Supprimer avec confirmation

Pour en revenir toujours à notre application exemple, il sera possible d'autoriser la suppression des enregistrements de son compte. Il peut y avoir un risque, surtout si nous ne demandons pas une confirmation de la suppression. Plusieurs actions peuvent être entreprises :

- La confirmation de la suppression de la ligne, pour être sûr que la personne ne s'est pas trompée.
- Prévoir éventuellement de demander, à partir d'un formulaire, la confirmation de la suppression de la ligne en retapant son mot de passe.

 Nous ne mettrons pas d'exemple, cet aspect est développé dans le chapitre La saisie et l'affichage - Supprimer un contact.

## 7. CSRF (Cross Site Request Forgeries)

Cette technique est à l'opposé de la technique XSS (vue précédemment) car au lieu d'exploiter la confiance de notre visiteur, le CSRF (*Cross Site Request Forgeries*) exploite la confiance du site Web en ses utilisateurs.

La méthode joue surtout sur le chemin d'origine de l'url. En résumé, la provenance des fichiers qui seront affichés dans la barre d'adresse du navigateur. Pour éviter ce genre d'attaque :

- Utiliser le tableau `$_POST` plutôt que les variables créées grâce à `register_globals`.
- Ne pas simplifier les actions importantes surtout pour supprimer des données. Il est préférable de demander une confirmation supplémentaire pour être certain qu'il s'agit d'une personne physique.
- Forcer les utilisateurs à passer par un formulaire pour la validation des étapes importantes.

## 8. PHP\_SELF

La fonction `PHP_SELF` utilisée sous la forme `$_SERVER['PHP_SELF']` dans beaucoup de nos formulaires, permet de reprendre l'adresse URL (chemin complet) dans laquelle notre formulaire de saisie va recevoir la saisie des différents champs de nos visiteurs.

Cependant elle présente facilement un risque d'insérer des scripts malicieux pour prendre le contrôle de votre système. Pour sécuriser cette variable, voici la solution que nous vous conseillons :

`securite/php_self.php`

```
<form action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>">
</form>
```

## 9. Crypter le mot de passe

Quand nous regardons le contenu d'une base de données, nous pouvons lire les données en clair. Toutefois, certaines données plus sensibles (notamment un mot de passe) peuvent être protégées et vous pouvez donc ajouter une sécurité supplémentaire face à toutes les personnes qui peuvent consulter ses données.

La méthode consiste à utiliser la fonction **md5**.

Elle permet de convertir une chaîne de caractères en une chaîne de caractères hexadécimaux d'une longueur de 32.

`securite/motdepasse.php`

```
<?php
$exemple="Editions ENI";
echo $exemple."<br />";

echo md5($exemple). "<br />";
```

Voici le résultat obtenu :



## 10. Listing dossier

Cet aspect correspond à la possibilité de perdre des données basiques, c'est-à-dire des dossiers et sous-dossiers qui ne sont pas situés à la racine du disque dur.

Voyons cela avec un petit exemple :

Prenons le chemin **include** qui contient toutes les routines importantes : /notreSiteInternet/include

La seule technique de protection qui peut exister est d'ajouter un fichier index.php dans le dossier concerné qui va renvoyer directement à la racine du site Internet.

listing/index.php

```
<?php
header( "Location: ../index.php" );
?>
```

Il est nécessaire de l'insérer dans tous les dossiers et sous-dossiers de notre application comme dans le dossier image. Ainsi, les données ou accès qui sont sensibles ne pourront pas être récupérés par cette technique.

# Le déboggage

## 1. Les erreurs classiques

Lorsque nous développons un site Internet, nous pouvons rencontrer certains messages d'alerte lorsque nous désirons afficher une page Internet.

Ces messages d'erreurs signalent des petits soucis dans une ou plusieurs lignes de la page. Ces erreurs peuvent se présenter sous différentes formes.

### Le point-virgule

Le caractère point-virgule ";" est impératif à la fin de chaque ligne de programme, pour signaler au langage la fin de l'instruction.

Exemple :

```
Echo « bonjour »
```

Au lieu de :

```
Echo « bonjour » ;
```

### Les guillemets

Il est fréquent d'oublier de terminer un littéral avec le signe double quote ou simple quote.

Exemple :

```
Echo « bonjour ;
```

Au lieu de :

```
Echo « bonjour » ;
```

### Les variables

Il s'agit dans une même instruction de placer du texte, et l'affichage de la valeur d'une variable.

Exemple :

```
Echo « Bonjour ». $variable « des éditions ENI»
```

Au lieu de :

```
Echo « Bonjour ». $variable. « des éditions ENI »
```

### Les accolades

Il s'agit d'encadrer par des accolades différentes instructions, et si le bloc comporte trop de lignes, d'oublier l'accolade fermante. Ce qui provoque un déséquilibre dans notre page.

Exemple :

```
?php
for ($i;$i<=10;$i++)
{
    echo "$i<br />";
}
```

Au lieu de :

```
<?php
```

```
for ($i;$i<=10;$i++)
{
echo "$i<br />";
}
?>
```

## 2. Erreur en Exception Etendue

Depuis la version 5 de PHP, une gestion d'exception étendue est possible comme dans d'autres langages de programmation.

Cette fonctionnalité permet d'analyser les blocs de ligne de programme et d'en stopper à tout moment l'exécution si une instruction ne correspond pas aux critères de sélection prévus.

Nous utiliserons ce principe d'exception avec les bases de données PDO, mais il peut être appliqué partout en PHP.

Les fonctions dont nous avons besoin sont :

**try**

Envoie les données.

**catch**

Attrape si une erreur apparaît.

**getCode**

Retourne le code erreur.

**getMessage**

Retourne le message erreur.

Dans un script, voici ce que nous pouvons écrire :

```
Try
{
    ....
}
catch (Exception $e)
{
echo 'Nº : '.$e->getCode().'<br />';
die ('Erreur : '.$e->getMessage().'<br />');
}
?>
```

## 3. Les variables

Lors d'un développement, il est souvent nécessaire de connaître les valeurs qui se trouvent dans les variables lors de l'envoi de données à partir d'un formulaire. Nous allons utiliser la fonction `print_r` :

**print\_r**

Affiche les informations lisibles d'une variable.

**<pre>**

Rend la lecture plus lisible.

```
<?php
echo "<pre>";
print_r ($_SESSION);
echo "<br>";
```

```
print_r($_GET);
echo "<br>";
print_r($_POST);
echo "</pre>";
?>
```

Cet exemple montre le contenu de la session, des GET et des POST sur toutes les pages de notre application.

Dans celle-ci, nous placerons ce code dans le fichier footer.inc.php inclus dans toutes les pages pour fermer les différentes balises que nous avons ouvert dans l'en-tête (head.inc.php).

## 4. Les requêtes

Dans une page Internet, il est souvent nécessaire d'effectuer plusieurs appels vers la base de données, donc d'exécuter plusieurs fois mysql\_query.

Si une erreur apparaît, il est souvent difficile de retrouver la ligne où se pose le problème. Heureusement, le langage PHP nous permet d'utiliser une fonction qui, si nous l'activons, permet de connaître exactement où se trouve le problème.

### **assert\_options()**

Fixe et lit différentes options d'assertions.

Nous allons utiliser quatre options pour cette fonction :

#### **ASSERT\_ACTIVE**

Active l'assistant ASSERT.

#### **ASSERT\_WARNING**

Affiche ou désactive l'alerte PHP pour chaque ligne fausse.

#### **ASSERT QUIET\_EVAL**

Active ou désactive le rapport d'erreur.

#### **ASSERT\_CALLBACK**

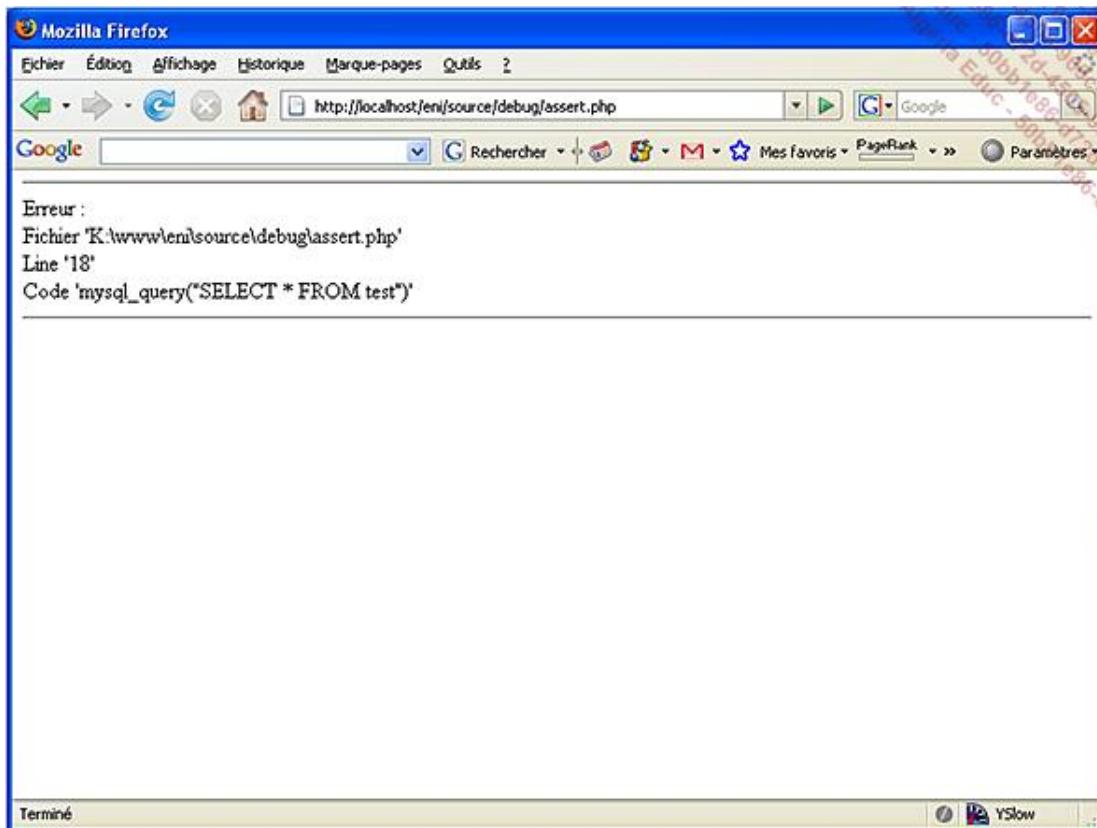
Retourne les informations utiles si une erreur apparaît.

```
debug/assert.php
```

```
<?php
assert_options(ASSERT_ACTIVE, 1);
assert_options(ASSERT_WARNING, 0);
assert_options(ASSERT QUIET_EVAL, 1);

function recuper_assert($file, $line, $code)
{
    echo "<hr />";
    echo "Erreur : <br />";
    echo "Fichier '$file'<br />";
    echo "Line '$line'<br />";
    echo "Code '$code'<br />";
    echo "<hr />";
}
assert_options (ASSERT_CALLBACK, 'recuper_assert');
assert ('mysql_query("SELECT * FROM test")');
?>
```

Nous obtenons le résultat suivant :



Quand nous aurons terminé le développement de notre application, pour mettre le site Internet en production nous devrons désactiver la fonction ce qui se traduira par la modification d'une seule instruction comme ceci :

```
<?php  
assert_options(ASSERT_ACTIVE, 0);  
?>
```

Le gros avantage est que nous n'aurons pas besoin d'éditer toutes les pages de notre site Internet pour trouver les lignes qui utilisent la fonction ASSERT.

Dans l'application, nous allons utiliser ce débogueur pendant la période de développement avec l'exemple qui a été utilisé précédemment.

Et dans toutes les instructions qui exécutent une requête, nous allons insérer la fonction de test qui va se présenter comme ceci :

```
<?php  
assert ('mysqli_query($sql)');  
?>
```

## 5. Le warning

Le warning est une option utile pour réaliser une bonne programmation PHP.

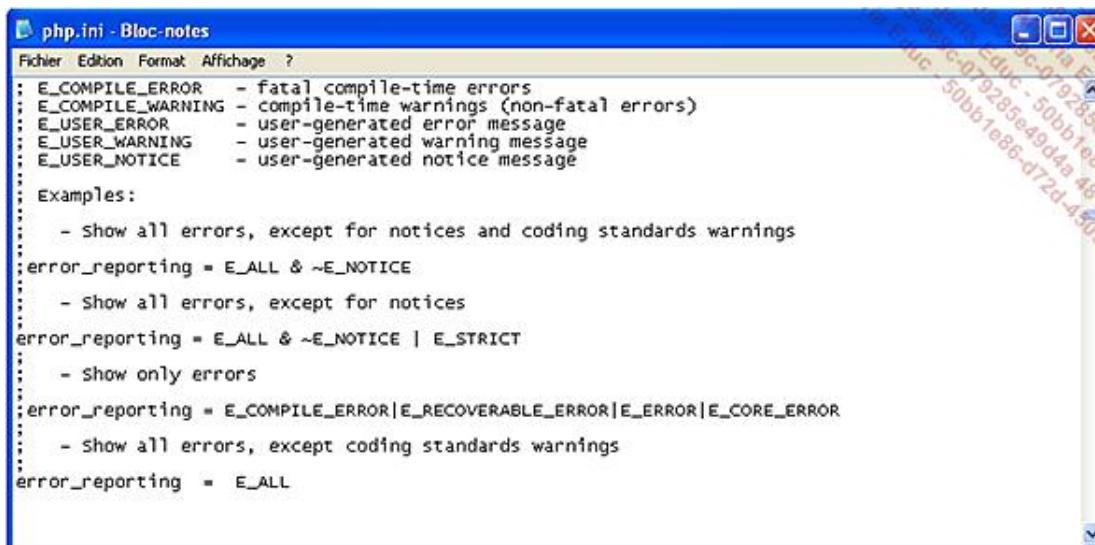
Cette option n'est jamais activée sur un serveur de production car il peut alors fournir des informations non souhaitées à l'internaute. Cependant, cette option est importante lors d'un développement car elle permet de réaliser une programmation qui respecte les normes en vigueur.

Pour l'activer, nous effectuons les opérations suivantes :

- rechercher l'emplacement du fichier php.ini ;
- éditer le fichier ;
- rechercher la ligne `error_reporting` concernée ;

- supprimer le point-virgule du début de ligne ;
- sauvegarder le fichier ;
- relancer le serveur.

Nous activons le mode ALL du fichier PHP.INI comme ceci :



```

php.ini - Bloc-notes
Fichier Edition Format Affichage ?
; E_COMPILE_ERROR    - fatal compile-time errors
; E_COMPILE_WARNING  - compile-time warnings (non-fatal errors)
; E_USER_ERROR       - user-generated error message
; E_USER_WARNING     - user-generated warning message
; E_USER_NOTICE      - user-generated notice message

Examples:
- show all errors, except for notices and coding standards warnings
error_reporting = E_ALL & ~E_NOTICE

- Show all errors, except for notices
error_reporting = E_ALL & ~E_NOTICE | E_STRICT

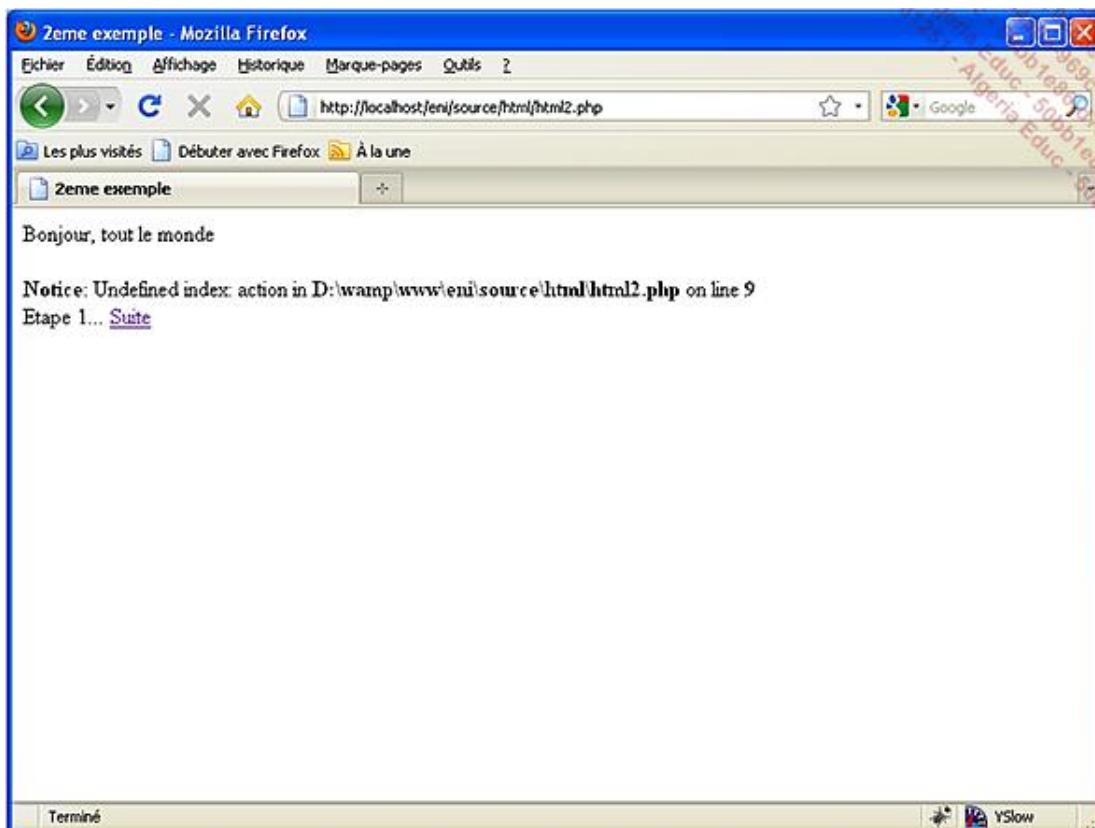
- Show only errors
error_reporting = E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR

- Show all errors, except coding standards warnings
error_reporting = E_ALL

```

Activer la totalité des options permet d'avoir en même temps le mode warning, ce qui active l'affichage des messages d'alertes sans bloquer le déroulement de l'application.

Reprenons un exemple du premier chapitre, L'environnement de développement :



Comme le montre la capture ci-dessus, nous désirons afficher quelque chose, cependant une action semble indéfinie.

Ce message peut apparaître sous différentes formes :

- l'utilisation d'une nouvelle variable,
- la récupération de données provenant d'un `$_GET[]` ou `$_POST[]`.

La solution consiste à rajouter un test sur la valeur interrogée en utilisant la fonction `ISSET`, pour obtenir le résultat suivant :

```
<?php
if (isset($_GET['action']) && $_GET['action']=="suite")
?>
```

# Image

Les images sont aussi des éléments importants à prendre en compte car nous pouvons les créer dans différents formats, avec différents contenus à la volée et les afficher à la demande.

## 1. Les fonctionnalités

Une image possède un en-tête qui définit un format d'affichage. Les formats disponibles en natifs sont : JPG/JPEG, GIF, PNG.

Dans une programmation HTML standard, nous utilisons la balise HTML `<img>` qui se présente de la façon suivante :

```
<img src= 'image1.png' >
```

Lorsque nous définissons le contenu de notre image, nous utilisons toujours la même balise HTML `<img>`, mais cette fois elle se présente de la façon suivante :

fichier/image1.html

```
<html>
<head></head>
<body>
<img src= 'image1.php'>
</body>
</html>
```

La balise `<img>` contient un nom de fichier, ce qui se traduit par l'appel d'un fichier externe qui sera affiché dans un format image.

Pour concevoir ce fichier, nous utilisons les fonctions suivantes :

**imagecreate** : crée une nouvelle image à palette.

**imagecreatetruecolor** : crée une nouvelle image en couleurs vraies.

**imagePNG** : envoie une image PNG vers un navigateur ou un fichier.

**imagedestroy** : détruit une image.

Après avoir créé un fichier « image1.php » qui va nous permettre de créer notre image.

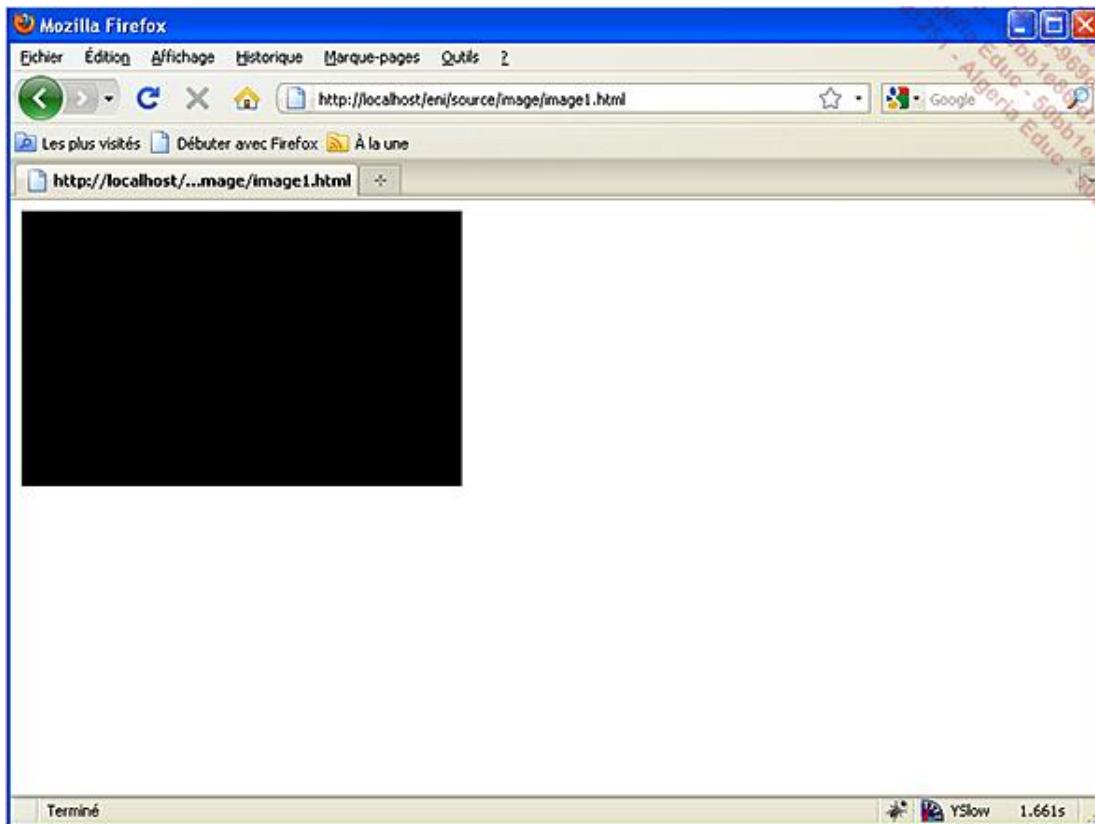
La création d'une image se déroule de la façon suivante :

- déclaration de l'en-tête,
- déclaration des dimensions,
- génération du contenu,
- affichage du contenu image,
- libération de l'image.

images/image1.php

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
imagepng($img);
imagedestroy($img);
?>
```

Nous obtenons le résultat suivant :



Il est possible de sauvegarder l'image au lieu de l'afficher à l'écran. L'opération se réalise de la façon suivante :

Image/image1.php

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
imagepng($img,"image1.png");
imagedestroy($img);
?>
```

Les autres formats comme le format JPG/JPEG ou GIF utilisent leurs propres fonctions :

**imagejpeg** : envoie une image JPEG vers un navigateur ou un fichier.

**imagegif** : envoie une image GIF vers un navigateur ou un fichier.

Elle se présente de la façon suivante pour le format JPG :

```
<?php
header('Content-type: image/jpg');
$img = imagecreatetruecolor(320,200);

imagejpeg($img);
imagedestroy($img);
?>
```

et pour le format GIF :

```
<?php
header('Content-type: image/gif');
$img = imagecreatetruecolor(320,200);

imagegif($img);
imagedestroy($img);
?>
```

Le résultat reste identique au format précédemment présenté, c'est pourquoi nous utiliserons le format PNG dans tout l'ouvrage.

## 2. Les objets graphiques

Les objets graphiques peuvent se présenter sous différentes formes grâce à des fonctions spécifiques qui offrent la possibilité de dessiner des lignes, des carrés, des cercles avec ou sans remplissage intérieur, mais aussi d'afficher un champ texte ou en police de caractères.

Suivant les dimensions des objets, ceux-ci peuvent représenter des formes différentes ; un carré peut devenir un rectangle ou un cercle peut se transformer en ellipse.

Les fonctionnalités disponibles :

**imageline** : dessine une ligne.

**imageellipse** : dessine une ellipse.

**Imagerectangle** : dessine un rectangle.

### a. Ligne

Nous allons tracer une ligne de couleur rouge dans l'image précédente. La déclaration, l'affichage et la libération ne changent pas. L'évolution apportée concerne l'utilisation d'une palette de couleurs (voir la section Les couleurs) et de la fonction **imageline** dont la syntaxe est la suivante :

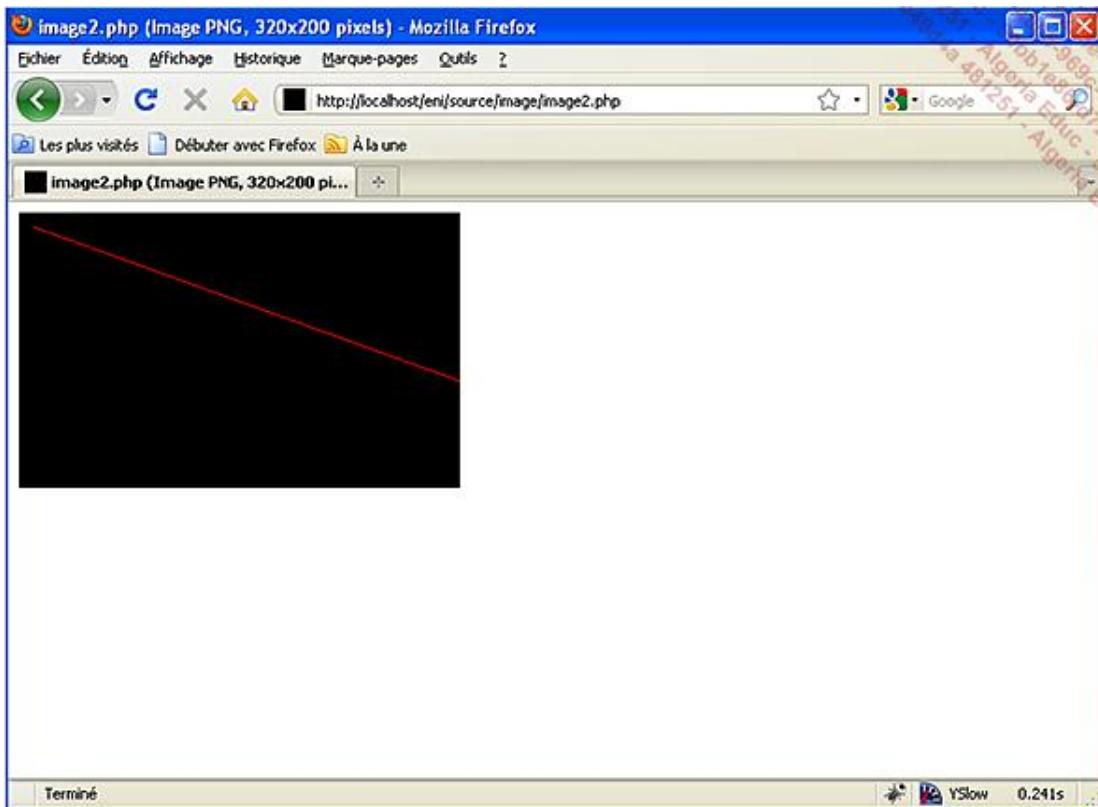
```
imageline(image , x1 , y1 , x2 , y2 , couleur )
```

qui se présente ainsi dans notre exemple :

```
image/image2.php
```

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
$col=imagecolorallocate($img, 255,0,0);
imageline($img,10,10,400,150, $col);
imagepng($img);
imagedestroy($img);
?>
```

pour obtenir le résultat suivant :



### b. Carré/rectangle

La représentation d'une forme géométrique comme un carré ou un rectangle s'effectue de la même façon.

Pour réaliser cette opération, nous utilisons la fonction suivante :

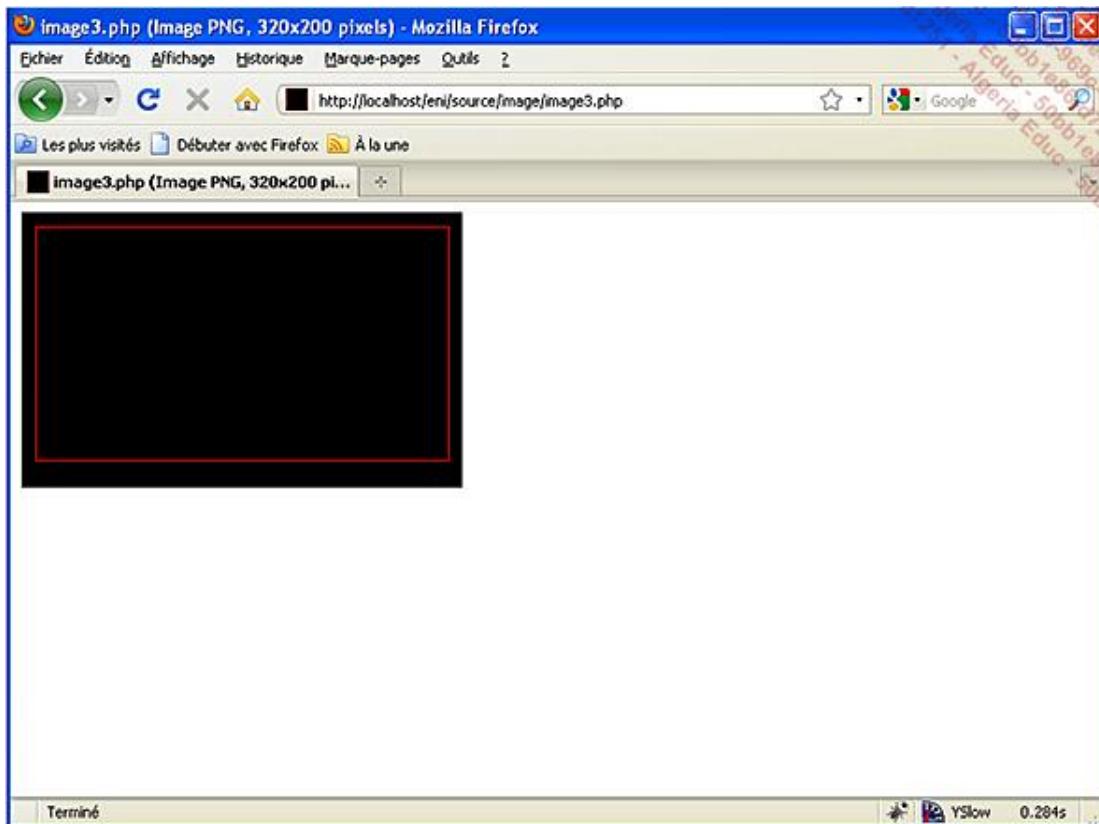
```
Imagerectangle ($img, x1 , y1 , x2 , y2 , couleur )
```

présentée de la façon suivante :

Image/image3.php

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
$col=imagecolorallocate($img, 255,0,0);
imagerectangle($img,10,10,310,180, $col);
imagepng($img);
imagedestroy($img);
?>
```

pour obtenir le résultat suivant :



### c. Cercle/ellipse

La représentation d'une forme géométrique comme un cercle ou une ellipse s'effectue de la même façon.

Pour réaliser cette opération, nous allons utiliser la fonction suivante :

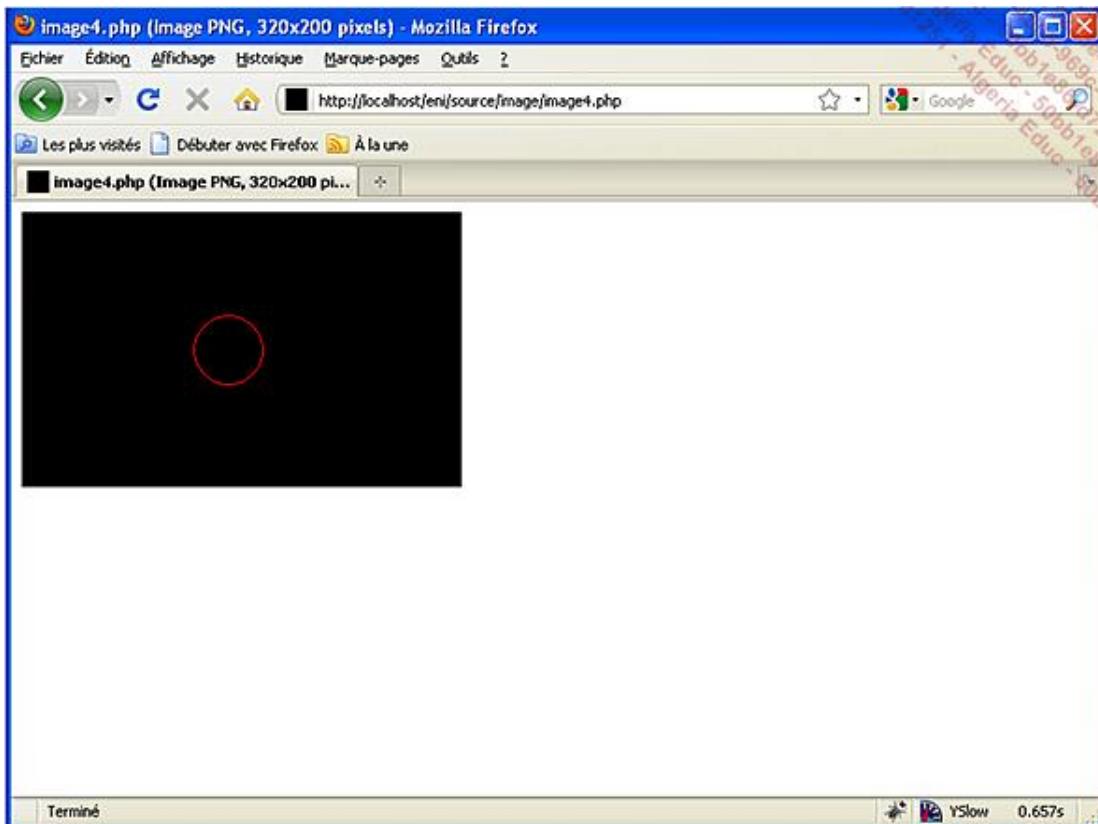
```
imageellipse (image , position X , position Y, largeur, hauteur , couleur )
```

présentée de la façon suivante :

Image/image4.php

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
$col=imagecolorallocate($img, 255,0,0);
imageellipse($img, 150, 100, 50,50, $col);
imagepng($img);
imagedestroy($img);
?>
```

pour obtenir le résultat suivant :



Suivant la largeur et la hauteur de la forme, nous obtenons des formes différentes.

### 3. Les couleurs

Les couleurs permettent de rendre un dessin attrayant et plus lisible.

Nous utilisons la fonction :

**Imagecolorallocate ()** : alloue une couleur pour une image.

Dont la syntaxe est la suivante :

```
Imagecolorallocate ($image, Rouge, Vert, Bleu)
```

Les valeurs utilisées pour cette fonction sont des valeurs décimales de 1 à 255 pour les couleurs rouge, vert, bleu. Ces 3 couleurs sont les couleurs primaires numériques. Grâce à ces 3 couleurs de base, nous obtenons des millions de couleurs.

Pour obtenir les couleurs principales, nous mettons à votre disposition quelques exemples utiles :

```
<?php
$blanc = ImageColorAllocate ($image, 255, 255, 255);
$noir = ImageColorAllocate ($image, 0, 0, 0);
$bleu = ImageColorAllocate ($image, 0, 0, 255);
$violet= ImageColorAllocate ($image,255,255, 0);
$jaune= ImageColorAllocate ($image, 0, 255,255);
$vert = ImageColorAllocate ($image, 0, 255, 0);
$rouge= ImageColorAllocate ($image, 255, 0, 0);
?>
```

Reprendons les différents objets dessinés précédemment pour les remplir d'une couleur. Pour cela, utilisons les fonctions suivantes :

**Imagefilledrectangle ()** : dessine un rectangle rempli.

**Imagefilledellipse()** : dessine une ellipse remplie.

Nous allons créer un cercle bleu et un carré vert dans le même dessin, ce qui se traduit comme ceci :

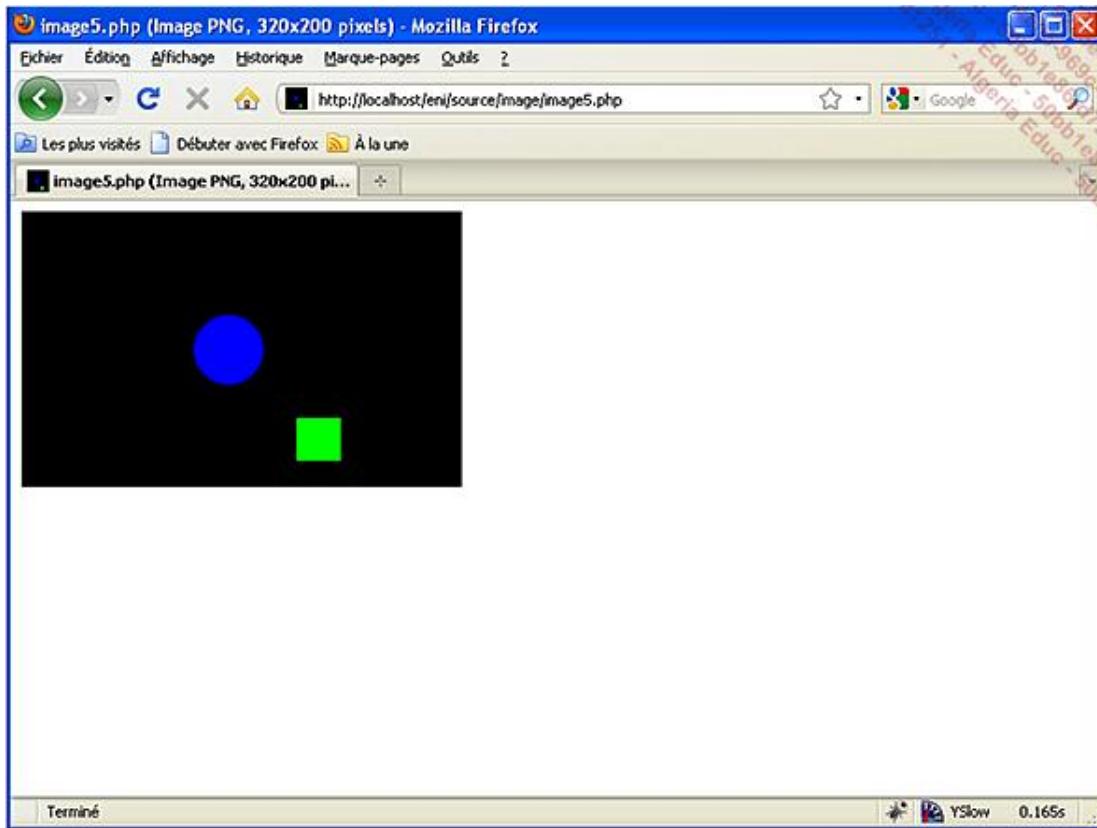
```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);

$vert=imagecolorallocate($img, 0,255,0);
$bleu=imagecolorallocate($img, 0,0,255);

imagefilledrectangle($img,200,150,231,180, $vert);
imagefilledellipse($img, 150, 100, 50,50, $bleu);

imagepng($img);
imagedestroy($img);
?>
```

pour obtenir le résultat suivant :



## 4. Le texte

Une image ne contient pas que des objets graphiques, elle peut aussi contenir un texte. Il existe deux méthodes pour afficher un texte :

**Imagestring ()** : dessine une chaîne horizontale.

**Imagefttext ()** : dessine un texte avec une police True Type.

### a. Chaîne horizontale

L'affichage d'une chaîne de texte de différentes tailles se programme de la façon suivante :

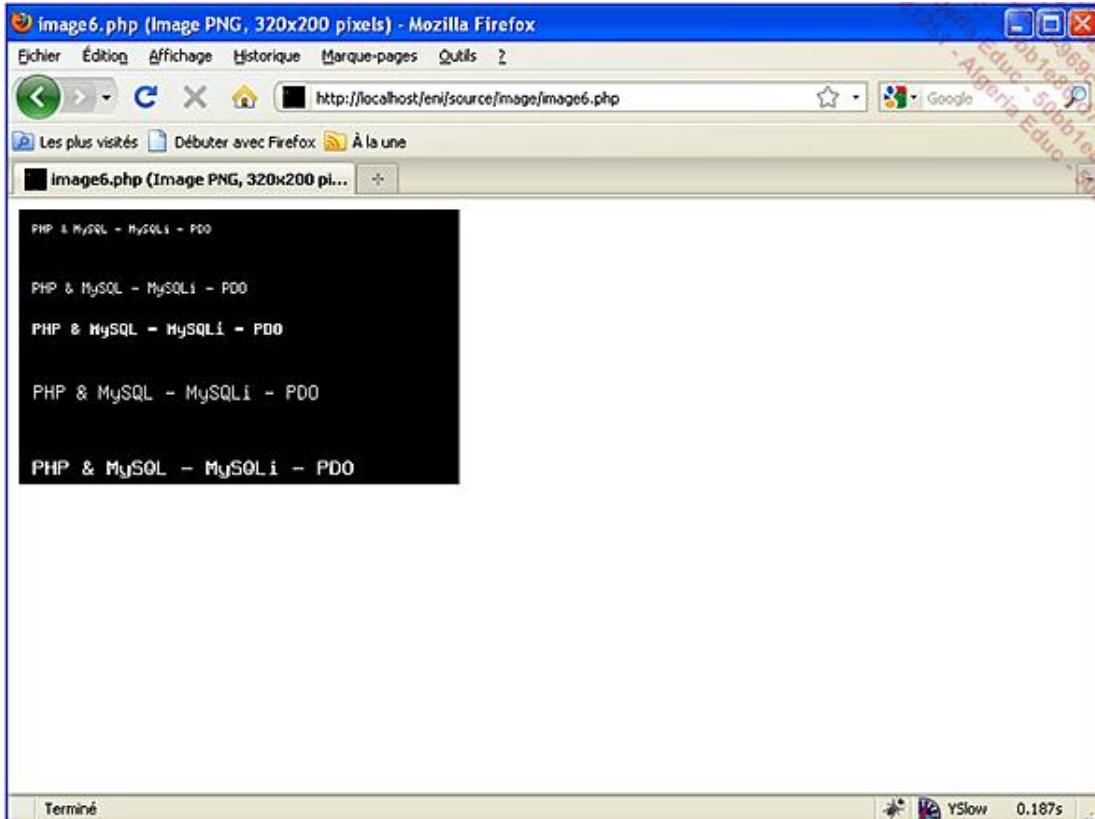
```
<?php
header('Content-type: image/png');
```

```

$img = imagecreatetruecolor(320,200);
$blanc=imagecolorallocate($img, 255,255,255);
imagestring($img, 1, 10, 10, "PHP & MySQL - MySQLi - PDO", $blanc);
imagestring($img, 2, 10, 50, "PHP & MySQL - MySQLi - PDO", $blanc);
imagestring($img, 3, 10, 80, "PHP & MySQL - MySQLi - PDO", $blanc);
imagestring($img, 4, 10, 125, "PHP & MySQL - MySQLi - PDO", $blanc);
imagepng($img);
imagedestroy($img);
?>

```

Pour obtenir le résultat suivant :



Cette fonction est très utilisée pour ajouter un petit texte sur une image.

## b. Police TrueType

L'utilisation des polices de caractères TrueType est très répandue car le format est associé aux fichiers \*.TTF disponibles dans tous les systèmes d'exploitation. Les polices de caractères les plus répandues sont Arial, Times, Verbatim, etc. Cependant il existe des milliers de polices libres de droits disponibles sur le Web.

Cette fonction se présente de la façon suivante :

```
Imagettftext (image , taille,angle, x, Y, couleur, police TTF, texte)
```

Affichons un texte dans notre image :

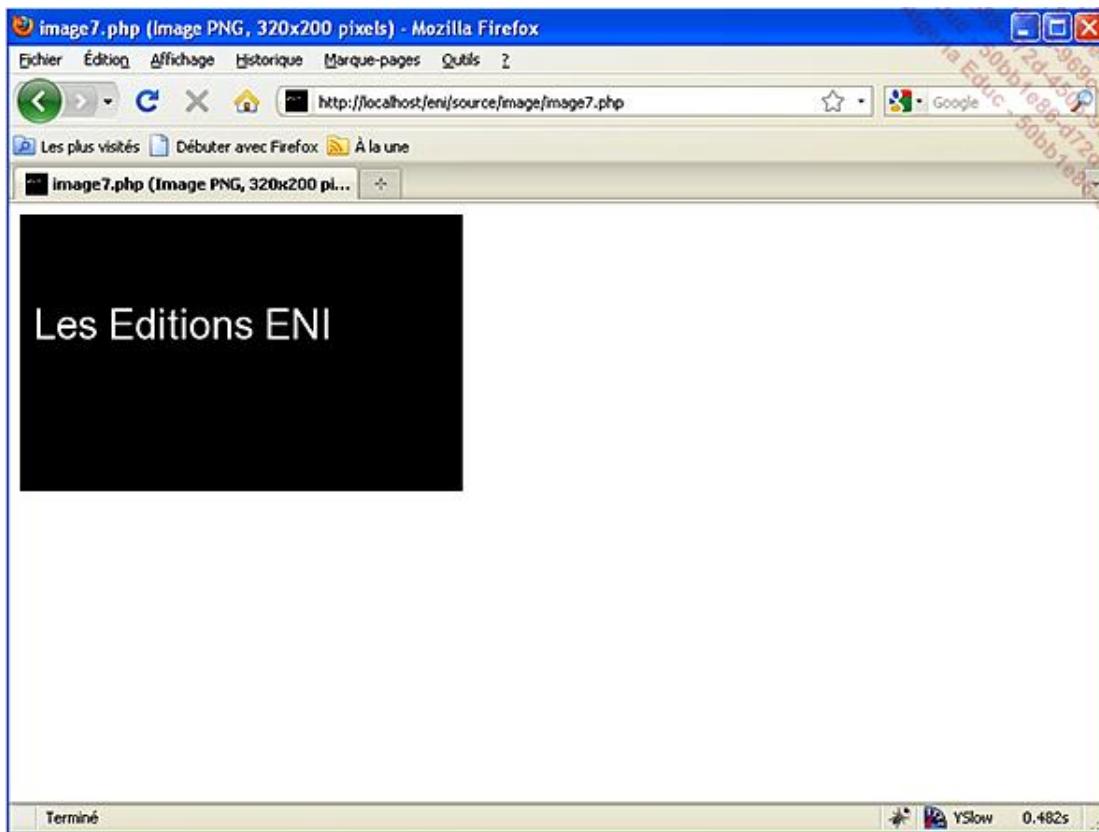
Image/image7.php

```

<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);
$blanc=imagecolorallocate($img, 255,255,255);
imagettftext($img, 22, 0, 10, 90, $blanc, "Swiss721 BT.TTF", "Les Editions ENI");
imagepng($img);
imagedestroy($img);
?>

```

Nous obtenons le résultat suivant :



## 5. Fichier externe

Lorsque nous créons une image, nous avons la possibilité d'inclure une autre image à l'intérieur de celle-ci.

Pour réaliser cette opération, il existe la fonction suivante :

```
Imagecreatefromjpeg(nom du fichier)
```

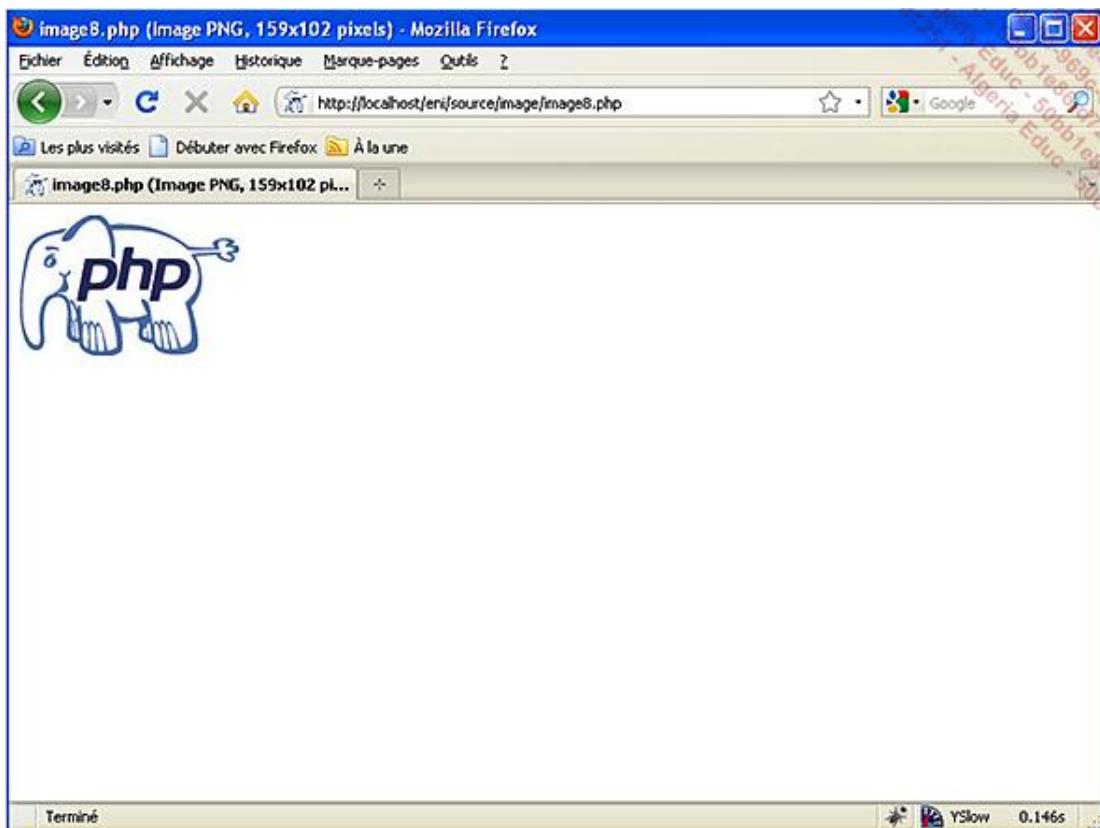
Nous utiliserons une image JPEG pour obtenir le code suivant :

```
Image/image8.php
```

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor(320,200);

$img = imagecreatefromjpeg("elephant.jpg");
imagepng($img);
imagedestroy($img);
?>
```

Et obtenons le résultat suivant :



## Introduction

Dans ce chapitre nous allons aborder plus précisément le développement de notre application. Les fonctions et concepts de base détaillés dans le chapitre précédent seront exploités dans les pages suivantes pour aboutir à la réalisation de notre interface avec ses différentes fonctionnalités :

- les accès sécurisés pour définir les droits des utilisateurs,
- la gestion du carnet d'adresses,
- les paramétrages du carnet d'adresses,
- la gestion des mots de passe,
- la gestion des administrateurs.

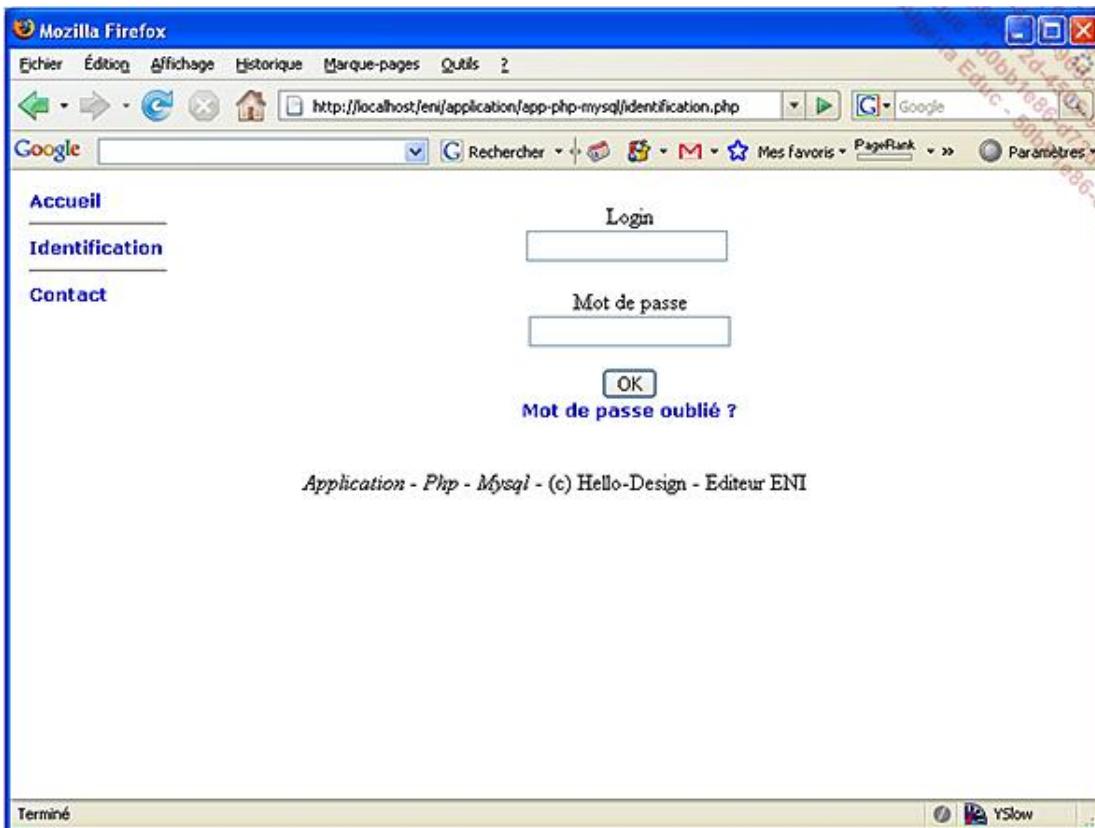
# Connexion à un compte

## 1. L'identifiant

Comme nous l'avons vu dans le chapitre La préparation du développement, nous pouvons accéder à certaines parties du site par l'intermédiaire d'une interface. L'identifiant permet d'avoir un filtrage simple mais utile pour accéder à certaines pages du site internet.

Nous allons réutiliser ce principe pour permettre à un utilisateur d'accéder à la partie administration du site (admin) tout en gardant la possibilité pour ce même utilisateur de bénéficier comme tous les autres utilisateurs de la gestion de son propre carnet d'adresses.

Dans l'exemple, nous allons accéder à notre espace privé à partir d'un formulaire, comme ci-après :



Nous proposons dans ce formulaire deux champs à remplir : un champ Login (identifiant) et un champ Mot de passe. Ces deux champs sont obligatoires et permettront au visiteur d'accéder à son compte (si les champs ont été remplis correctement).

Si les valeurs ne sont pas bonnes, nous afficherons un message signalant le problème qui se présentera comme ceci :

Fichier identification.php

```
<?php
if(isset($_GET['erreur']) && ($_GET['erreur'] == "login"))
{
echo "login ou mot de passe incorrect";
}
if(isset($_GET['erreur']) && ($_GET['erreur'] == "intru"))
{
echo "Echec d'identification !!!";
}
if(isset($_GET['erreur']) && ($_GET['erreur'] == "session"))
{
echo "Session expirée";
}
```

Un peu plus loin dans ce chapitre, section Connexion à un compte - L'e-mail, le mot de passe, nous ajouterons un lien pour le cas où le visiteur aurait oublié ses identifiants afin qu'il puisse les récupérer (nous lui ferons parvenir ces éléments sur son adresse mail pour être sûrs qu'il s'agit de la bonne personne qui demande ces identifiants).

## 2. La connexion

Avant d'utiliser notre application, rappelons certains points concernant le contrôle des champs :

**Login** : l'identifiant saisi va être gardé en l'état, sans aucune conversion à part la conversion des caractères au format HTML. Nous réaliserons cette manipulation avec la fonction **HTMLENTITIES()** que nous avons étudiée précédemment.

**Mot de passe** : le mot de passe va être protégé par la fonction **md5**. Ainsi nous allons envoyer une valeur cryptée ce qui permettra d'assurer la confidentialité auprès des personnes qui possèdent un compte.

La fonction md5 est une chaîne de caractères (**string**) qui utilise un algorithme et retourne un résultat sur 32 caractères hexadécimaux.

En voici un petit exemple.

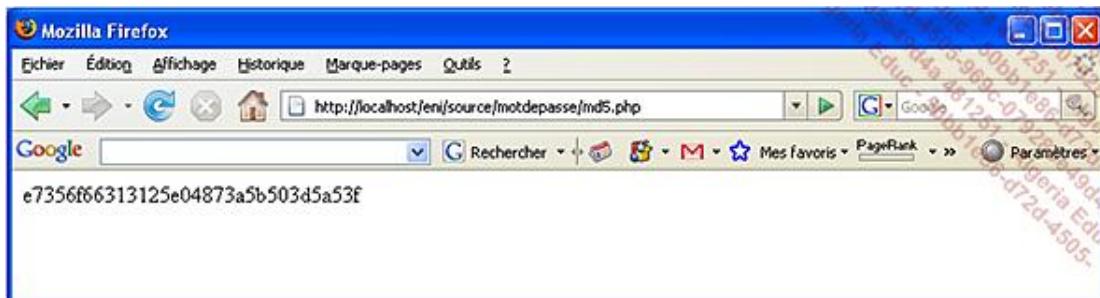
`motdepasse/md5.php`

```
<?php
$exemple="Editions ENI2007";
echo md5($exemple). "<br />";
?>
```

### md5

Calcule le MD5 d'une chaîne.

Voici le résultat que nous obtenons :



Lorsque le visiteur va se connecter, nous devons effectuer un certain nombre de vérifications avant de pouvoir lui donner accès à son compte.

Nous allons d'abord vérifier s'il possède un compte, avec tous les contrôles des caractères spéciaux comme ci-après.

Nous allons avoir besoin du fichier `fct.inc.php` qui contient une fonction qui va nous permettre de générer une clef de repérage.

`login.inc.php`

```
<?php
include "include/fct.inc.php";
?>
```

Nous testons si le champ n'est pas vide et contient bien une valeur avant de continuer. Si ce n'est pas le cas, nous revenons à l'écran précédent et nous affichons un message d'erreur.

Nous utilisons la fonction **mysql\_real\_escape\_string** :

### **mysql\_real\_escape\_string**

Protège les caractères spéciaux d'une chaîne pour l'utiliser dans une requête SQL.

```
<?php
if (isset($_POST['login']) && !empty($_POST['login'])) )
{
$login=htmlentities($_POST['login'], ENT_QUOTES, 'UTF-8');
$login=mysql_real_escape_string($login);
$password=htmlentities($_POST['password'], ENT_QUOTES, 'UTF-8');
$password=md5($password);
$password=mysql_real_escape_string($password);
}
else
{
  header("Location: identification.php?erreur=login" );
}
?>
```

Maintenant, nous allons vérifier si les caractères saisis sont bien présents dans la base de données. Nous devons avant tout convertir le login pour être sûr qu'il est composé de caractères HTML pour éviter toutes les attaques possibles.

Nous allons aussi chiffrer le champ mot de passe car les mots de passe de la base de données sont cryptés pour éviter que les comptes soient utilisés par des personnes auxquelles ils n'appartiennent pas.

```
<?php
$sql="SELECT * FROM user WHERE login='$login' AND password='$password' ";
$req=mysql_query($sql);
if(! $req ) echo ('Requête invalide : ' . mysql_error());
if (mysql_num_rows($req)==0)  header("Location:identification.php?erreur=login");
?>
```

Si nous ne trouvons pas le compte, nous revenons sur le formulaire d'authentification en signalant le problème.

Si nous avons trouvé le compte dans la base de données, nous modifions la clef de repérage qui nous permet d'être certains d'identifier la bonne personne lors des différentes manipulations et navigations dans le site Internet. Nous enregistrerons en même temps la date de passage qui peut nous être utile pour plus tard.

```
<?php
$idclef=recup_clef();
$date=DATE("Y-m-d");
$sql="UPDATE user SET idclef='$idclef',date_lastpass='$date' WHERE login='$login'
AND password='$password' ";
$query = mysql_query($sql) OR die("Mise à jour de la clé impossible : <br />.
mysql_error());
?>
```

La clef est déterminée par un appel à la fonction `recup_clef()` que nous avons vue dans la partie précédente. Cette clef change tout le temps permettant ainsi de ne pas avoir de doublon dans notre site Internet.

Après avoir effectué les vérifications d'usage, nous chargeons quelques informations que nous plaçons dans une variable SESSION.

```
<?php
$sql="SELECT * FROM user WHERE login='$login' AND password='$password'
AND idclef='$idclef' ";
$req=mysql_query($sql);
if(! $req ) echo ('Requête invalide : ' . mysql_error());
$row = mysql_fetch_object($req);

if (mysql_num_rows($req)!=0)
{
  session_start();
  $_SESSION['login'] = $row->login;
  $_SESSION['idclef'] = $row->idclef;
  $_SESSION['niveau'] = $row->niveau;
  $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
  $_SESSION['iduser'] = $row->id;
  $destination=$row->page;
  header("Location:$destination");
}
```

```

        }
    else
    {
        header( "Location:identification.php?erreur=intru" );
    }
?>

```

Si la personne est correctement identifiée, nous allons lui permettre de continuer en accédant à la partie complète du menu.

Nous en profitons pour enregistrer la clef, l'adresse de l'ordinateur (IP de la machine), l'identifiant, et son niveau d'accès. Ainsi, nous éviterons de faire appel à la base de données à chaque changement de page. Pour plus de détails concernant les choix des éléments mémorisés :

- Clef : la clef sera changée à chaque connexion.
- Adresse de la machine : nous sommes sûrs qu'il s'agit du même ordinateur connecté lors des changements de pages et de navigation.
- Niveau d'Accès : nous avons prévu deux niveaux : un niveau pour les utilisateurs (user) et un autre pour le ou les administrateurs (admin). Les possesseurs du niveau admin auront accès aux mêmes menus que les utilisateurs et posséderont un menu supplémentaire pour gérer les accès des comptes utilisateurs.

Pour accéder au menu complet d'un compte, nous allons accéder à la page par l'intermédiaire de la fonction header. Cette fonction permet de spécifier un nouvel en-tête http lors de l'envoi des fichiers HTML.

Grâce à cette fonction, nous évitons que la personne possédant un compte valide plusieurs fois son choix.

### 3. L'e-mail, le mot de passe

La plupart des sites Internet qui possèdent des identifiants et des mots de passe pour accéder à un compte privé proposent de recevoir ces identifiants par e-mail en cas de perte.

En PHP, la fonction mail() permet d'envoyer un e-mail. Elle prend en argument un destinataire, un objet et un corps de message.

Exemple :

mail/ex1.php

```

<?php
$destinataire = "email@votresite.com";
$objet = "1er exemple" ;
$message = "Un petit message pour notre 1er exemple \n" ;
$message .= "avec fonction mail() \n";
if ( mail($destinataire, $objet, $message) )
    echo "Envoi du mail réussi.";
else
    echo "Echec de l'envoi du mail." ;
?>

```

Bien sûr la fonctionnalité de base est assez limitée. Nous allons la faire évoluer en lui ajoutant un e-mail d'expéditeur pour permettre de spécifier une réponse et nous obtenons ceci :

mail/ex2.php

```

<?php
$emailexp="mail@votreemail.com";
$emaildest="email@destinataire.com";
$mailmessage = "Un petit message pour notre 2eme exemple \n" ;
$mailmessage .= "avec fonction mail() \n";

$titre="test email simple";
if (mail ("$emaildest","$titre","$mailmessage","From: $emailexp"))
    echo "Envoi du mail réussi.";
else

```

```
echo "Echec de l'envoi du mail.";  
?>
```

Nous avons réalisé l'envoi d'un e-mail au format texte. Bien sûr, nous pouvons incorporer dans le corps de l'e-mail un message au format HTML comme ceci :

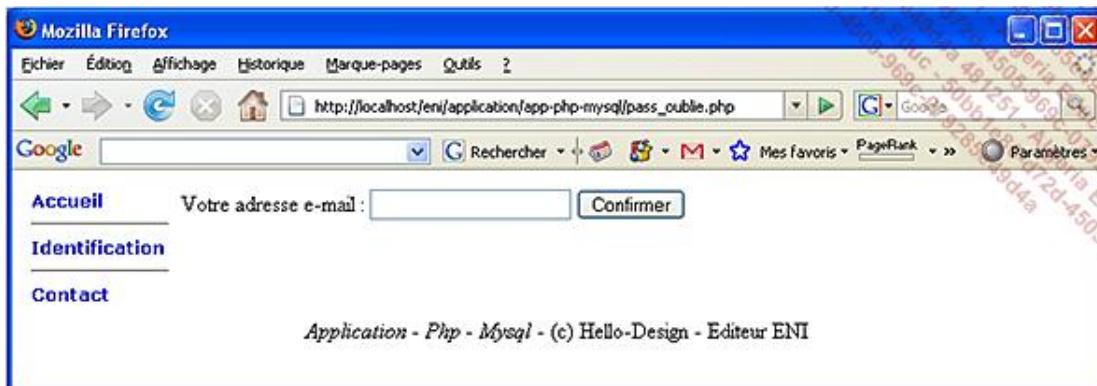
mail/ex3.php

```
<?php  
$emailexp="mail@votreemail.com";  
$emaildest="email@destinataire.com";  
$mailmessage = "<html><body>";  
$mailmessage .= "Les <h1>Editions ENI</h1>" ;  
$mailmessage .= "vous remercie d'avoir utilisé ce 3eme exemple <br>" ;  
$mailmessage .= "avec la fonction <b>mail()</b> au format HTML <br>" ;  
$mailmessage .= "</body></html>";  
$titre="test email au format HTML";  
  
if (mail ("$emaildest","$titre","$mailmessage","From: $emailexp"))  
    echo "Envoi du mail réussi.";  
else  
    echo "Echec de l'envoi du mail.";  
?>
```

Par contre, si vous évoluez dans un environnement Windows et que vous utilisez cette fonction sur votre serveur localhost, il est nécessaire d'effectuer une petite manipulation dans le fichier php.ini.

Il faut activer les lignes SMTP et sendmail\_form. Pour cela, il faut juste enlever le « ; » au début de la ligne. Depuis la version 1.7 de Wampserver, tout s'effectuera automatiquement.

Concernant notre application, pour envoyer un e-mail avec les codes d'accès du visiteur, nous allons tout d'abord passer par l'intermédiaire d'un formulaire de saisie comme ceci :



Après la saisie de l'e-mail par le visiteur, nous devons effectuer quelques tests pour pouvoir envoyer les codes d'accès vers cette adresse.

Nous allons tout d'abord vérifier si le champ e-mail est bien rempli et n'est pas vide pour continuer la préparation de l'envoi :

pass\_oubolie.php

```
<?php  
require ("head.inc.php");  
if (isset($_POST['action']) || !empty($_POST['action'])) $action = $_POST['action'];  
?>
```

Nous allons ensuite vérifier que c'est bien le bouton **Confirmer** de notre formulaire qui a été utilisé. Grâce à cela, nous allons convertir la saisie en caractères HTML, ce qui va nous permettre de rechercher si l'e-mail est présent dans la base de données.

Avec cette étape intermédiaire nous évitons les risques de blocages et de prises de contrôles malveillantes.

```
<?php  
switch($action)  
{
```

```

case "Confirmer":
    $emaildest=htmlentities($_POST["email"]);
    $sql="select * from user where email='".$emaildest' ";
    $valeur=mysql_query($sql);
    if( ! $valeur ) die ("Probleme de requete : " . mysql_error());

    if (!mysql_num_rows($valeur))
    {
        require_once "head.inc.php";
        echo "Cet Email n'existe pas dans nos comptes,
Merci de recommencer<br>";
        echo "<a href= ".$_SERVER['PHP_SELF']."' class=links>
Cliquer ici</a>";
        include "footer.inc.php";
        exit;
    }
    ...
}
?>

```

Après avoir vérifié la présence de l'e-mail dans la base de données, nous allons régénérer le mot de passe. En effet, le mot de passe actuel étant crypté avec la fonction MD5 personne ne peut en connaître la valeur, le seul moyen consiste à recréer le mot de passe. Nous allons donc faire appel à une fonction qui va générer un mot de passe. Dans le choix des caractères nous allons éviter ceux qui peuvent porter à confusion (entre les chiffres et les lettres par exemple).

Nous allons attribuer un mot de passe comprenant 8 caractères, mais il est toujours possible de spécifier une chaîne de caractères plus longue. Nous allons utiliser le principe d'une recherche de chaîne de caractères par rapport à un nombre aléatoire.

```

<?php
function creation_password()
{
    $alphabet = "abcdefghijklmnopqrstuvwxyz";
    $alphabet .= "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    $alphabet .= "23456789";

    $nbcara = 8;
    $i = 0;
    $pass = "";

    srand((double)microtime()*1000000);
    while ($i<$nbcara)
    {
        $valcara = rand(0, strlen($alphabet));
        $pass .= substr("$alphabet", $valcara, 1);
        $i++;
    }
    return $pass;
}
?>

```

Lorsque le nouveau mot de passe est créé, nous allons l'enregistrer en le protégeant de nouveau :

```

<?php
$row=mysql_fetch_object( $valeur );
$login=$row->login;
$mot_de_passe=creation_password();

$sql = "UPDATE user SET password='".MD5($mot_de_passe)."'"
WHERE email='".$emaildest."'";
$qid = mysql_query($sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
?>

```

La dernière étape va être la confection et l'envoi d'e-mail :

```

<?php
$emailexp="votresite@hello-design.fr";
$titre="Votre mot de passe";

```

```

$mailmessage="Bonjour,\n Vous avez demandé vos codes d'accès
pour le site .. : \n\n";
$mailmessage.=" Identifiant : ".$login."\n";
$mailmessage.=" Mot de passe : $mot_de_passe\n\n";
$mailmessage.="Pour information, il vous a été attribué un nouveau mot
de passe, le précédent est perdu\n";

if (mail ("$emaildest","$titre","$mailmessage","From: $emailexp"))
{
    echo "<br>Un email vous a été envoyé<br>";
    echo "<a href=identification.php class=links>Suite</a>";
    exit;
} else {
    echo "Echec de l'envoi du mail <br>";
    echo "<a href= ".$_SERVER['PHP_SELF']."' class=links>Cliquer ici</a>";
    exit;
}
}
break;
default:
break;
}
?>

```

L'e-mail est envoyé automatiquement et le visiteur reçoit les nouveaux codes d'accès dans sa messagerie. Il peut maintenant se connecter à son compte.

## 4. La session

Les sessions mémorisent certains éléments d'identification lors de la saisie des codes d'identification. Une fois dans la partie privée du site, nous allons effectuer la vérification et le contrôle de ces valeurs de sessions pour être certain de retrouver la bonne personne page après page.

Ces tests doivent être effectués pour chaque page Internet du site. Pour cela, nous allons utiliser un fichier commun **config.inc.php**. Ce fichier va se décomposer en plusieurs parties.

Nous créons une session :

```

<?php
session_start();
?>

```

Nous vérifions si la session possédant l'horaire du dernier passage n'a pas été détruite. Si c'est le cas, nous recréons un numéro de session, et réenregistrons l'heure de passage, la durée de session et l'adresse de l'ordinateur. Si ce n'est pas le cas, nous mémorisons la nouvelle heure de son passage.

```

<?php
if (!isset($_SESSION['dernier_passage']) )
{
    session_regenerate_id();
    $_SESSION['dernier_passage']=time();
    $_SESSION['duree']=$duree_session;
    $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
}
else
{
    $_SESSION['dernier_passage']=time();
}
?>

```

Ensuite nous vérifions si la clef de repérage du compte existe toujours. Si ce n'est pas le cas, nous ne sommes plus capables de savoir qui est connecté ; nous redirigeons donc le visiteur vers la page index du site.

```

if (!isset($_SESSION['idclef']) )
{
    header("Location:index.php");
}

```

Nous devons effectuer un nouveau test concernant la durée de présence sur la page Internet. Nous effectuons un test sur la durée de non utilisation du navigateur. Si le visiteur n'a généré aucune activité sur le site pendant la période définie, nous allons le renvoyer vers la page de saisie des paramètres d'identification. Si un changement de page a été effectué pendant la durée prévue, nous allons enregistrer l'heure d'affichage de cette nouvelle page.

```
if (time()-$_SESSION['dernier_passage']>$_SESSION['duree'])
{
    header("Location:identification.php?erreur=session");
}
else
{
    $_SESSION['dernier_passage'] = time();
}
?>
```

Nous avons effectué les principaux tests sur l'utilisation des sessions pour notre application. Mais ils ne s'arrêtent pas là, nous allons aussi effectuer un test de l'adresse machine.

## 5. L'adresse machine

Nous effectuerons une petite validation supplémentaire avant d'autoriser l'affichage de la page demandée. Il s'agit de repérer l'adresse IP de la machine, ce qui va nous permettre de suivre la navigation du visiteur.

Nous mémoriserons l'adresse de la machine dans une variable de session, ainsi lorsque le visiteur va changer de page, nous pourrons vérifier qu'il s'agit toujours la même personne.

```
<?php
if($_SERVER['REMOTE_ADDR'] != $_SESSION['ip'])
{
    header("Location:identification.php?erreur=session");
}
?>
```

Si nous remarquons que l'adresse a changé, nous renvoyons le visiteur sur la page d'identification pour qu'il retape ses codes d'accès.

# Menu

## 1. Affichage

Afficher le menu est nécessaire pour permettre la navigation sur le site internet. Par contre nous devons être certains d'autoriser cette navigation seulement si le visiteur s'est bien identifié.

Le test de vérification va s'effectuer de la façon suivante :

```
<?php if (!isset($_SESSION['dernier_passage']) || empty($_SESSION['idclef'])  
|| (time()-$_SESSION['dernier_passage']>$_SESSION['duree']) ) { ?>  
<a href=identification.php class=links>Identification</a><br>  
<?php } else { ?>  
<a href=compte.php class=links>Votre compte</a><br>
```

Notre test de contrôle consiste à vérifier que le dernier passage existe, que nous possédons déjà une clef identification et que la session a été ouverte dans le temps qui lui a été impartie. Si l'une des valeurs n'est pas correcte, le visiteur sera obligé de s'identifier. Si tout s'est correctement déroulé, nous pouvons afficher le menu général avec toutes ses options.

## 2. Contenu

Nous allons proposer au visiteur les choix suivants :

- création de rubriques,
- saisie de personnes dans son carnet d'adresses,
- et, suivant son niveau d'accès, administration des comptes des visiteurs.

Pour rendre l'application plus intéressante, nous allons proposer d'autres choix au visiteur qui peuvent être considérés comme des sous-menus. Les sous-menus seront gérés par des pages Internet qui seront affichées.

Avant d'afficher les options du menu auxquelles le visiteur a droit, il faut vérifier que la clef d'identification (générée comme nous l'avons vu auparavant) est toujours bien présente.

```
<?php if (empty($_SESSION['idclef'])) { ?>  
<a href=identification.php>Identification</a><br>  
<?php } else { ?>  
<a href=compte.php>Votre compte</a><br>
```

L'exemple montre que si la clef est absente de la session, nous proposons une nouvelle identification. Si elle est présente, nous affichons quelques informations en souhaitant la bienvenue au visiteur.

Pour afficher le bon menu en fonction de la navigation du visiteur, nous allons mettre en place plusieurs filtres. Dans chaque page nous allons spécifier le menu concerné.

Par exemple, pour aller dans le menu carnet, nous allons placer ce repère dans une variable.

```
<?php  
$menu="carnet";  
?>
```

Comme ceci quand nous affichons le menu et qu'il détecte que nous sommes dans la partie carnet, l'application affiche le carnet et les sous-menus liés à celui-ci.

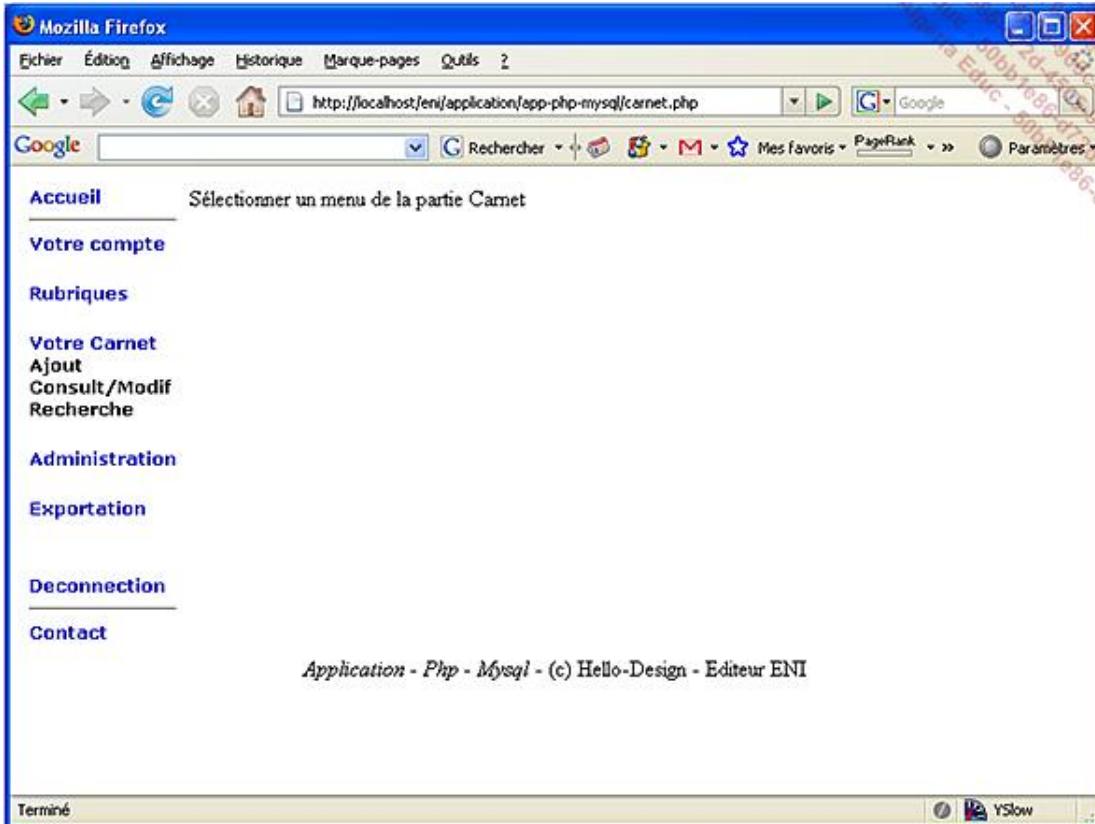
```
<a href=rubrique.php>Les rubriques</a><br>  
<?php if ($menu=="carnet") { ?>  
<a href=rubrique-add.php>Ajout</a><br>  
<a href=rubrique-view.php>Consult/Modif</a><br>  
<?php } ?>
```

Le titre Rubriques sera toujours affiché. Par contre le sous-menu ne sera affiché que si le visiteur sélectionne cette partie, grâce au critère inséré au début de chaque page Internet.

Concernant les accès aux parties uniquement destinées aux personnes dotées du niveau administrateur, nous allons comparer avec le niveau de la session pour les afficher ou pas.

```
<?php if ($_SESSION['niveau']=="admin") { ?>
<a href=admin.php>Administration</a><br>
<?php if ($menu=="admin") { ?>
<a href=admin-add.php>Ajout</a><br>
<a href=admin-view.php>Consult/Modif</a><br>
<?php } ?>
```

Concernant la gestion des sous-menus voici le résultat que nous obtenons si nous avons sélectionné le carnet :



### 3. Description

Le menu présente de nombreuses lignes, ce qui correspond à un sommaire. Chaque élément du menu propose des pages différentes.

#### \* Accueil

Cette page correspond à la première page que nous voyons lorsque nous arrivons sur le site Internet.

#### \* Identification / Votre compte

Cet élément propose deux choix :

- Si le visiteur n'est pas identifié, il aura la possibilité de saisir ses identifiants.
- Si le visiteur est identifié, nous affichons les informations que nous avons en notre possession :
  - son login
  - le mot de passe masqué (personne à part lui ne le connaît). Il pourra modifier son mot de passe s'il le souhaite
  - son e-mail

- la date de création de son compte

#### **\* Rubriques**

L'utilisateur a la possibilité de choisir des rubriques de classement pour ses adresses et aussi de les personnaliser.

#### **\* Votre carnet**

Le carnet correspond à un carnet d'adresses classique mais informatisé. Pour chaque contact enregistré dans le carnet, il sera possible d'inscrire des informations supplémentaires le concernant.

*Par exemple : son nom, prénom, adresse, e-mail, téléphone, etc.*

#### **\* Administration**

Ce menu est réservé aux personnes qui possèdent le niveau admin. Il permet de gérer les comptes des utilisateurs.

#### **\* Exportation**

L'exportation propose différents choix pour extraire les données à destination d'autres logiciels ou applications.

#### **\* Déconnexion**

La déconnexion correspond à la fin des manipulations dans le carnet d'adresses. Elle ferme la session et renvoie directement à l'accueil. L'utilisateur redevient un visiteur anonyme.

#### **\* Contact**

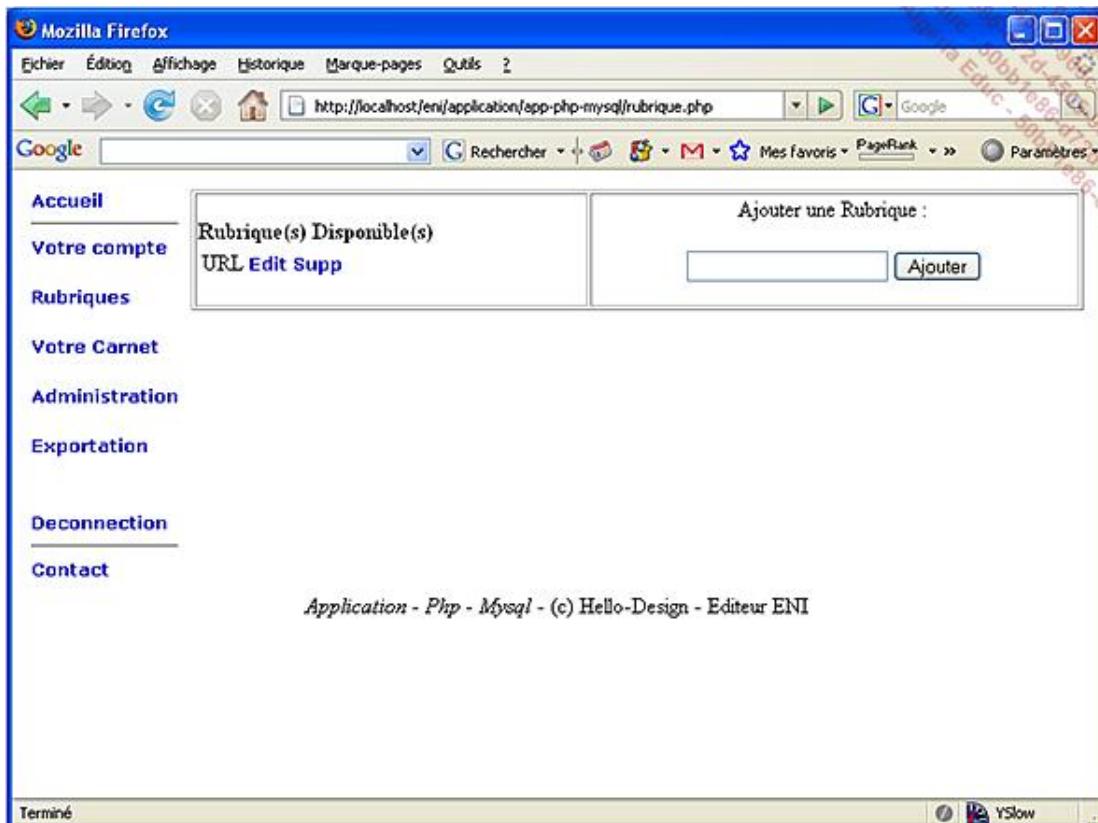
Cette page permet d'obtenir quelques informations sur la ou les personnes qui ont réalisé le site Internet.

# La gestion des rubriques

Nous allons dans cette section permettre à chaque titulaire d'un compte de pouvoir personnaliser ses rubriques, pour les utiliser par la suite et les incorporer automatiquement pour un usage personnel sans que les autres utilisateurs puissent les connaître.

Cette section sera construite et détaillée avec une base de données MySQL.

 Pour information, cette section est également disponible avec les formats MySQLi et PDO, en téléchargement sur le site de l'éditeur.



Quelques explications sont nécessaires à propos de cet écran : comme la plupart des applications bureautiques disponibles sur le marché, la partie configuration du compte propose la totalité de la gestion sur un seul écran.

Nous allons utiliser cette approche en donnant la possibilité d'ajouter, de modifier, d'éditer ou de supprimer une rubrique, en utilisant \$\_POST pour envoyer les données.

## 1. Ajouter une rubrique

Les rubriques qui vont être ajoutées dans un compte vont permettre par exemple de classer certaines informations par thème.

La partie **Ajouter** se trouve sur la moitié droite de l'écran. Nous avons une zone de texte qui va permettre à l'utilisateur de saisir ses rubriques favorites et de les ajouter.

Après confirmation de la saisie, nous devons effectuer certains tests pour en vérifier la validité :

`rubrique.php`

```
<?php
if ( empty($frm['nom']) )
{
    echo "Le champ est vide";
} elseif ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR']
    || $_POST['idclef'] != $_SESSION['idclef'])
```

```

{
    echo "Tentative de pénétration";
} else {
    $sql = "INSERT INTO rubrique ('iduser', 'nom')
VALUES (
    '".$_SESSION['iduser']."' ,
    '".htmlentities($frm[nom], ENT_QUOTES).'
    )";
$qid = mysql_query($sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
}
?>

```

Pour incorporer cette nouvelle rubrique dans la base de données, nous testons déjà si le champ est bien rempli. Sans ce test, nous risquons de nous retrouver avec un champ vide dans la table rubrique ce qui peut être très gênant.

Nous effectuons ensuite un petit test supplémentaire qui n'est pas vraiment obligatoire puisqu'il est déjà présent à l'ouverture de la page avec le test des sessions. Mais étant donné qu'il s'agit des Rubriques d'un carnet d'adresses, il vaut mieux effectuer un test supplémentaire pour être tranquille. Nous allons vérifier que c'est toujours le même ordinateur qui effectue la manipulation et que la clef de session est toujours identique.

Après ces tests, nous pouvons insérer le contenu de la saisie dans la table rubrique.

La table rubrique possède deux champs :

- Le champ iduser : permet de mémoriser l'identifiant de l'utilisateur. Cette valeur vient de la session.
- Le champ nom : nous allons convertir le contenu en caractères HTML (avec la conversion des caractères spéciaux) pour être certain que nous n'aurons pas de problèmes lors de l'affichage des listes dans les autres écrans de l'application.

## 2. Visualiser les rubriques

Quelle que soit la manipulation que nous allons effectuer dans cette section, nous devons pouvoir consulter la liste de nos rubriques. Cette liste est personnelle et ne sera pas consultable par les autres titulaires d'un compte sur le site.

Pour construire cette liste nous allons faire appel à l'exécution d'une requête comme celle-ci :

`rubrique.php`

```

<?php
$sql="SELECT rubrique.id as idselect,rubrique.iduser,rubrique.nom,user.id
    FROM rubrique, user
    WHERE rubrique.iduser=user.id
        AND rubrique.iduser='_'.$SESSION['iduser'].''
        AND user.idclef='_'.$SESSION['idclef'].' ';
$qid=mysql_query($sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
?>

```

Dans un souci de sécurité, nous effectuons une connexion à la table rubrique avec la table des utilisateurs pour être certains qu'il n'y a pas eu de pertes ou de modifications de l'utilisateur.

Nous allons aussi détecter si nous avons bien des données avec la fonction mysql\_num\_rows qui nous permet de savoir si nous avons au moins une information à afficher.

```

< ?php
if (mysql_num_rows($qid)!=0)
{
?>

```

Si c'est le cas, nous construisons un tableau HTML classique sans bordure pour rendre les alignements verticaux plus facile. Nous affichons trois colonnes :

- une colonne Rubrique,

- une colonne édition,
- une colonne suppression.

La colonne Rubrique contient le résultat de la requête. Les deux autres colonnes vont être utiles pour modifier et supprimer chacune des lignes de la liste qui sera affichée, elles vont être étudiées dans les pages qui suivent.

```

echo "<td><b>Rubrique(s) Disponible(s)</b><br>" ;
echo "<table>" ;
while( $row=mysql_fetch_object( $qid ) )
{
    echo "<tr>" ;
    echo "<td>".stripslashes($row->nom). "</td>" ;
    echo "<td><a href=". $_SERVER[ 'PHP_SELF' ]. "?id=$row->idselect
class=links>Edit</a></td>" ;
    echo "<td>" ;
    echo "<a " ;
    echo "onClick=\\\"Javascript:return confirm('Êtes-vous sûr de vouloir
enlever cette ligne ?');\\\" " ;
    echo "href=". $_SERVER[ 'PHP_SELF' ]. "?id=". $row->idselect . "&choix=trash" ;
    echo " class=links>Supp</a>" ;
    echo "</td>" ;
    echo "</tr>" ;
}
echo "</table>" ;
echo "</td>" ;
}
else
{
    echo "Aucune rubrique disponible" ;
}
?>

```

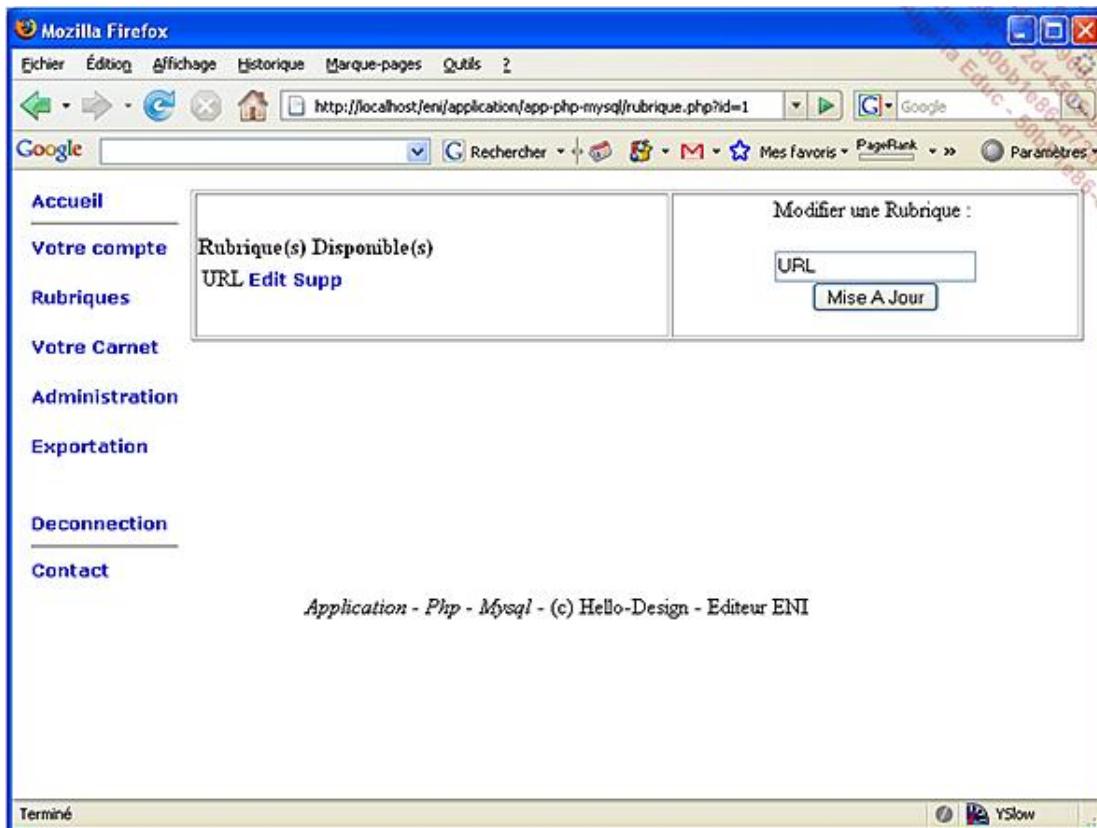
### 3. Éditer une rubrique

Le lien **Edit** va nous offrir plusieurs possibilités :

- corriger le nom de la rubrique,
- changer le nom de la rubrique.

Pour pouvoir effectuer ces manipulations, nous devons procéder en plusieurs étapes : une pour la partie visible et une pour le traitement et la mise à jour de la base de données.

Voici la présentation de l'écran :



Pour modifier une ligne, nous créons un lien dynamique et nous sélectionnons la ligne désirée grâce à ce lien qui se présentera comme ceci :

```
< ?php
echo "<a href=\".$_SERVER['PHP_SELF']."?id=$row->idselect class=links>Edit</a>" ;
?>
```

L'instruction se décompose ainsi :

**\$\_SERVER['PHP\_SELF']**

Recopie le nom de la page pour le lien dynamique.

**id**

Mémorise le numéro de la ligne affichée, c'est-à-dire le numéro de la ligne.

Lorsque l'utilisateur active le lien, la page Internet se recharge avec de nouveaux paramètres, permettant ainsi de modifier sa présentation.

Nous gardons la moitié gauche de l'écran (l'affichage de la liste des rubriques) sans modifications. Ce qui va changer concerne la moitié droite car nous nous trouvons sur une page de mise à jour.

Avant d'afficher la valeur dans le formulaire, nous devons vérifier que le paramètre **id** possède bien une valeur et qu'elle n'est pas vide.

```
<?php
$id=verif_GetPost($_GET['id']);
?>
```

Cette instruction appelle une fonction pour vérifier certains critères. Dans un souci d'efficacité, nous avons choisi de créer une fonction, ainsi nous pourrons y faire appel pour tester et contrôler la provenance de certaines valeurs. La fonction se présente comme ceci :

```
<?php
function verif_GetPost($clef=' ')
{
    if ( empty($clef) && isset($clef) )
        return false;
```

```

elseif (strlen($clef)<1)
    return false;

$clef=htmlentities($clef, ENT_QUOTES,'UTF-8');
return $clef;
}
?>

```

La fonction va récupérer la valeur du champ que nous envoyons par GET ou par POST. Nous testons si le champ n'est pas vide ou nul et vérifions si nous avons au moins un caractère. Nous en profitons pour le convertir au format de caractères HTML et nous renvoyons le résultat pour permettre de continuer le bon déroulement de l'application.

Après cette petite vérification, nous pouvons exécuter la même requête que pour afficher la liste des rubriques disponibles mais nous insérons un critère en plus, la valeur du paramètre id que nous venons de tester, pour pouvoir afficher la ligne choisie par le visiteur.

Bien sûr, nous avons converti cette valeur en caractère html pour être certain de ne pas avoir subi une attaque malveillante et surtout non souhaitée.

```

<?php
$id=htmlentities($id);
$sql="SELECT rubrique.id as idselect,rubrique.iduser,rubrique.nom,user.id
    FROM rubrique, user
    WHERE rubrique.iduser=user.id
        AND rubrique.iduser='". $_SESSION['iduser']."'."
        AND user.idclef='". $_SESSION['idclef']."'."
        AND rubrique.id='".$id'
    ORDER BY rubrique.nom
";
$req=mysql_query($sql);
if (!$req) die('Requête invalide : ' . mysql_error());
$row=mysql_fetch_object( $req);
?>
<input name="id" type="hidden" value=<?php echo $id;?> ><br>
<input type="text" name="nom"  value="<?php echo stripslashes($row->nom); ?>"><br>
<input type="submit" name="action" value="Mise A Jour">

```

Maintenant que nous avons affiché la valeur de la rubrique sélectionnée, l'utilisateur pourra effectuer les corrections désirées. Nous en profitons pour enregistrer la valeur id dans un champ caché qui nous sera utile pour la mise à jour de la table.

Pour sauvegarder les modifications, l'utilisateur clique sur le bouton **Mise A jour** ce qui déclenche un processus intermédiaire avant la mise à jour de la table.

```

<?php
$frm = $_POST;
if ( empty($frm['nom']) )
{
    echo "Le champ est vide";
} elseif ( $_SESSION['ip'] != $_SERVER['REMOTE_ADDR'] 
    || $_POST['idclef'] != $_SESSION['idclef'] )
{
    echo "Tentative de pénétration";
} else {
$sql="UPDATE rubrique SET
    'nom' = '".htmlentities($frm['nom'], ENT_QUOTES).'
    where id='".htmlentities($frm['id'])."' AND iduser='". $_SESSION['iduser']."' ";
    $qid = mysql_query($sql);
    if (!$qid) die('Requête invalide : ' . mysql_error());
}
?>

```

Nous vérifions que le champ n'a pas été effacé et que c'est toujours la même personne qui effectue la modification avec le même compte d'accès.

Nous appliquons lors de la mise à jour de la table, la conversion des caractères en caractères HTML.

Étant donné que toutes ces opérations sont effectuées sur la même page Internet, nous n'avons pas besoin d'être redirigé vers la page pour visualiser le contenu de la liste des rubriques. Cela s'effectue automatiquement.

## 4. Supprimer une rubrique

Pour supprimer une rubrique, nous utilisons le même principe que pour le lien **Edit** : à partir de la liste des rubriques que nous affichons nous pouvons sélectionner une des lignes et cliquer sur un lien qui va permettre de la supprimer.

Pour supprimer cette ligne, nous créons un lien dynamique intitulé **Supp** sur lequel l'utilisateur cliquera :

```
< ?php
echo "<a onClick=\"Javascript:return confirm('Êtes-vous sûr de vouloir enlever
cette ligne ?');\" href=\"$_SERVER['PHP_SELF']."?id=".$row->idselect."&choix=
trash class=links>Supp</a>";
?>
```

L'instruction se décompose comme ceci :

**\$\_SERVER['PHP\_SELF']**

Recopie le nom de la page pour le lien dynamique.

**id**

Mémorise le numéro de la ligne affichée.

**choix**

Un critère avec la valeur trash qui correspond à la corbeille (suppression).

Avant de lancer la suppression de la ligne choisie, nous allons en demander la confirmation en utilisant JavaScript. L'avantage de cette approche est de pouvoir annuler le choix.

Si le visiteur confirme la suppression, nous vérifions que la rubrique sélectionnée n'est pas utilisée dans son carnet d'adresses. Dans ce cas en effet, ce serait dommage de la supprimer.

```
<?php
$sql="SELECT carnet_details.idcarnet, carnet_details.idcarnet,carnet.iduser,
carnet.id
FROM carnet_détails ,carnet
WHERE carnet_details.idcarnet=carnet.id
    AND carnet.iduser='". $_SESSION['iduser']."' "
    AND carnet_details.idrubrique=' $id'
";
$qid=mysql_query($sql);
if (mysql_num_rows($qid))
    echo "Impossible effacer la ligne";
?>
```

Dans cette requête, nous faisons une jointure entre deux tables puisque l'une contient le numéro du visiteur (iduser), pour éviter de supprimer quelque chose qui ne lui appartient pas.

Si la rubrique (insérer :) à supprimer n'est pas utilisée, nous allons passer par une autre étape intermédiaire qui consiste à demander au visiteur de retaper son mot de passe pour être certain qu'il s'agit du bon utilisateur. Cette étape n'est pas obligatoire pour notre application mais elle fortement conseillée pour une application qui se trouve sur Internet avec beaucoup de personnes qui peuvent se connecter.

Nous affichons un écran de saisie :



Nous pouvons effectuer une dernière vérification pour être sûrs que le mot de passe est correct. Nous recherchons la présence du mot de passe avec la clef de session et la clef insérée dans le champ caché du formulaire.

```
<?php
$sql="SELECT * FROM user
WHERE password = '".MD5($frm['pass'])."'
AND idclef='".$_SESSION['idclef']."'"
AND idclef='".$frm['idclef']."'"
";
$qid=mysql_query($sql);
if (!mysql_num_rows($qid))
die ("Erreur dans votre mot de passe");
?>
```

Comme nous avons tout vérifié et que les tests ont tous été concluants, nous pouvons effacer la rubrique qui a été sélectionnée.

```
<?php
$sql="DELETE FROM rubrique WHERE
rubrique.id = '".$frm['id']."'"
AND rubrique.iduser = "'.$_SESSION['iduser']."'"
;
$qid = mysql_query($sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
?>
```

Nous affichons de nouveau la liste des rubriques qui restent dans la base de données, permettant ainsi de continuer les manipulations.

# La gestion du carnet d'adresses

## 1. Présentation

Nous allons dans cette partie voir comment remplir une fiche (ou un contact) dans le carnet d'adresses. Nous devons procéder en différentes étapes :

- effectuer les contrôles nécessaires,
- rendre obligatoires certains champs,
- exécuter une requête évoluée en MySQLi.

Après avoir rempli la fiche principale, nous allons mettre en place de nouvelles fonctionnalités comme un bloc note où nous pourrons insérer différentes informations concernant la personne, par exemple : site Internet, blog, tchat, e-mail...

---

 Pour information, cette partie est également disponible pour les formats MySQL et PDO en téléchargement sur le site de l'éditeur.

---

Nous allons inscrire le champ Nom en majuscules et le champ Prénom avec juste la première lettre en majuscule et le reste en minuscules.

## 2. Ajouter un contact

Pour commencer, nous réalisons un formulaire de saisie, comme ci-après :

Accueil      Ajouter un contact dans le carnet :

**Votre compte**      Genre :

**Rubriques**      Nom :

**Votre Carnet**      Prenom :  \*

**Ajout**      Adresse :

**Consult/Modif**      Code postal :

**Recherche**      Ville :

**Administration**      Email :  \*

**Exportation**      Téléphone :

**Déconnection**      Portable :

**Contact**      Photo :

---

Rubrique :

Observation :

---

Application - Php - Mysqli - (c) Hello-Design - Éditeur ENI

Terminé

Le formulaire se compose de deux parties :

- une partie concernant les coordonnées,
- une partie informations supplémentaires.

Concernant la première partie, nous y trouvons différentes zones à saisir :

- Genre : Madame, Monsieur, Mademoiselle ;
- Nom : le nom du contact ;
- Prénom : le prénom du contact ;
- Adresse : son adresse avec le code postal et la ville ;
- Email : son e-mail ;
- Téléphone : son téléphone ;
- Photo : la possibilité de mettre une photo.

Nous allons rendre obligatoires la saisie du prénom et de l'e-mail, c'est-à-dire que si ces deux champs ne sont pas

remplis lors de la validation du formulaire, nous allons réafficher les informations déjà inscrites dans les zones correctes en signalant l'obligation de remplir ces champs.

Concernant la deuxième partie, nous n'avons que deux zones :

- la rubrique ;
- observations.

La liste des rubriques sera remplie automatiquement à partir des informations que nous avons enregistrées à l'aide du menu Rubriques.

La zone **Observations** permettra de saisir une information ou un commentaire sur la rubrique.

### a. Première partie de la saisie

Cette partie du formulaire correspond à un formulaire HTML classique. Il est important de signaler quelques points : nous allons faire évoluer la balise `<select></select>` (format HTML) qui correspond à une liste déroulante, en recherchant automatiquement la bonne ligne qui sera affichée, c'est-à-dire marquée `SELECTED`.

La balise HTML se décompose comme ceci :

```
<select>
<option></option>
</select>
```

L'utilisation de cette balise avec le langage PHP est très simple, et nous allons utiliser une petite fonction pour éviter de taper toutes les lignes :

```
<?php
echo "<select name=\"genre\">";
echo ligne_selected(" ", "-1", $frm['genre']);
echo ligne_selected("Monsieur", "Monsieur", $frm['genre']);
echo ligne_selected("Madame", "Madame", $frm['genre']);
echo ligne_selected("Mademoiselle", "Mademoiselle", $frm['genre']);
echo "</select>";
?>
```

Lors de l'affichage d'une ligne, nous envoyons trois paramètres à la fonction :

- Valeur = valeur que nous allons retourner (`$value`) ;
- Nom = Le mot à afficher, ici il s'agit du genre (`$nom`) ;
- Valeur extérieure = détermine si l'option est `SELECTED` (`$entree`).

Grâce à cela, nous allons pouvoir construire les lignes options dont nous avons besoin :

```
function ligne_selected($nom, $value, $entree)
{
    $resultat = "<option value='".$value."' ";
    if ($value == $entree) $resultat .= " SELECTED ";
    $resultat .= ">".$nom."</option>";
    return $resultat;
}
```

L'avantage de cette technique est d'alléger le nombre de lignes mais aussi de permettre la programmation semi-automatique, donc d'obtenir un gain de temps et moins de risques d'erreurs.

L'autre particularité du formulaire correspond aux champs obligatoires. La gestion des champs obligatoires va s'établir de la façon suivante : nous allons tester les lignes signalées par une étoile car ce sont des champs obligatoires. Si un problème apparaît, nous stockerons dans un tableau le ou les messages d'erreur pour les afficher en une seule fois.



Pour mémoire : cette technique a été étudiée dans le chapitre La préparation du développement - Le contrôle des formulaires.

## b. Deuxième partie de la saisie

Cette partie du formulaire ne comporte pas de champs obligatoires, car ces informations peuvent être ajoutées plus tard en éditant la fiche. La zone Rubrique se présente sous la forme d'une liste déroulante. Cette liste est une énumération de données, mais ici c'est l'utilisateur du compte qui va la remplir sans que nous lui imposions des choix. Cette liste est alimentée à partir des rubriques saisies dans la section Rubriques de l'application qui a été étudiée dans ce chapitre, section La gestion des rubriques.

Pour construire la liste déroulante, nous allons utiliser le résultat d'une requête, mais au lieu de l'afficher au format HTML, nous allons afficher le résultat dynamiquement à partir du résultat de la requête que nous venons d'exécuter.

```
<?php
echo "<select name=\"idrubrique\">";
echo ligne_selected(" ","-1","");
$sql="SELECT id,iduser,nom FROM rubrique WHERE iduser='".$_SESSION['iduser']."'"
ORDER BY nom";
$rubrique=mysqli_query($connex,$sql);
if (!$rubrique) die('Requête invalide : ' . mysqli_error($connex));
if (mysqli_num_rows($rubrique)!=0)
{
    while( $list=mysqli_fetch_object( $rubrique) )
    {
        echo ligne_selected($list->nom,$list->id,'');
    }
}
echo "</select>";
?>
```

Pour afficher le choix de la ligne qui a été retenu, nous allons appeler la fonction `ligne_selected` dont nous avons parlé précédemment pour permettre d'afficher la bonne ligne. Mais ici, elle sera toujours vide car il s'agit d'une nouvelle saisie.

La zone Observations est un champ des plus classiques, donc aucune manipulation ne sera effectuée actuellement dessus.

## c. Contrôle du formulaire

Le contrôle du formulaire consiste à effectuer quelques tests par l'intermédiaire d'une fonction. Si les tests passent tous correctement, nous enregistrerons dans la base de données les données qui ont été saisies.

Dans notre fonction, nous allons être amenés à nous connecter à la base de données. Une fonction étant un programme séparé du reste de l'application, pour effectuer le test nous devons signaler que nous avons besoin d'utiliser la connexion avec le serveur à l'aide de cette fonction :

### Global

Permet d'utiliser des variables générales provenant du site proprement dit.

```
<?php
function valide_form(&$frm, &$erreurs)
{
Global $connex;
?>
```

Les erreurs seront stockées dans un tableau : nous devons effectuer plusieurs tests et pour retourner en une seule fois l'ensemble des messages, nous les mémoriserons dans un tableau que nous allons déclarer.

```
<?php
$erreurs = array();
$msg = array();
?>
```

Nous vérifions si le champ Prénom est vide ou pas. Si celui-ci est vide, nous mémorisons le message d'erreur et la présence d'une erreur ce qui nous permettra un peu plus loin de contrôler si le formulaire possède des erreurs ou pas.

```
<?php
if (empty($frm['prenom'])) 
{
    $erreurs['prenom'] = true;
    $msg['prenom'] = "Il manque le prénom";
}
?>
```

Maintenant, nous testons le champ Email un peu différemment du champ Prénom. Nous allons convertir son contenu en caractères HTML et au format UTF-8 :

```
<?php
$email=htmlentities($_POST['email'], ENT_QUOTES,'UTF-8');
?>
```

Nous testons si le champ n'est pas vide :

```
<?php
if (empty($email))
{
    $erreurs['email'] = true;
    $msg['email'] = "Absence d'email";
}
?>
```

Nous effectuons maintenant un test utilisant les expressions régulières pour vérifier si l'e-mail qui a été saisi correspond au format standard des e-mails.

```
<?php
elseif (!preg_match('/^[:alnum:][[-_.]?[:alnum:]]*@[[:alnum:]][[-_.]?[:alnum:]]*.(?:[a-z]{2,4})$/',$email))
{
    $erreurs['email'] = true;
    $msg['email'] = " Mauvais format d'Email";
}
?>
```

Nous vérifions maintenant si l'e-mail n'existe pas déjà dans le carnet d'adresses de l'utilisateur.

```
<?php
elseif ((mysqli_num_rows(mysqli_query($connex, "SELECT 1 FROM carnet WHERE email='".$frm['email']."' AND iduser='".$_SESSION['iduser']."' )) > 0))
{
    $erreurs['email'] = true;
    $msg['email'] = "L'email existe déjà";
}
?>
```

Après avoir effectué l'ensemble des tests, nous renvoyons le tableau, poursuivant ainsi le déroulement de notre application.

```
<?php
    return $msg;
}
```

Si un problème ou une erreur apparaît lors de chacun des tests, nous mémorisons où se situe le problème pour le signaler dans le formulaire.

Si tous les tests se sont correctement déroulés, nous allons enregistrer les données saisies dans la base de données MySQLi. Cependant, nous allons effectuer cette insertion en plusieurs étapes :

- insertion de la fiche proprement dit ;
- insertion des éléments d'information concernant la personne ;
- ajout de la photo en effectuant une mise à jour de la fiche.

L'insertion de la fiche va s'effectuer de la façon suivante :

```
<?php
$sql = "
INSERT INTO carnet (
    'iduser'
    , 'nom'
    , 'prenom'
    , 'adresse1'
    , 'adresse2'
    , 'codepostal'
    , 'ville'
    , 'tel'
    , 'portable'
    , 'email'
) VALUES (
    '".$_SESSION['iduser']."'"
    ,'"stroupper(htmlentities($frm['nom']))"'
    ,'"ucfirst(strtolower(htmlentities($frm['prenom'])))"'
    ,'"htmlentities($frm['adresse1'])"'
    ,'"htmlentities($frm['adresse2'])"'
    ,'"$frm['codepostal']"'
    ,'"htmlentities($frm['ville'])"'
    ,'"htmlentities($frm['tel'])"'
    ,'"htmlentities($frm['portable'])"'
    ,'"$frm['email']"'"
) ";

$qid = mysqli_query($connex,$sql);
if(! $qid ) die ('Requête invalide : ' . mysqli_error($connex));
?>
```

Rappelons que le nom de famille doit être inscrit en majuscule et que pour le prénom c'est juste la première lettre et le reste en minuscule. Pour simplifier la modification du prénom, nous mettons le prénom complet en minuscule et ensuite nous utilisons sur le premier caractère la fonction de mise en majuscule. Les fonctions utilisées sont :

### **strtoupper**

Renvoie une chaîne de caractère en majuscule.

### **strtolower**

Renvoie une chaîne de caractère en minuscule.

### **ucfirst**

Met le premier caractère en majuscule.

Quand nous insérons l'enregistrement, le numéro de la ligne est automatiquement créé car nous avons défini lors de la création de la table un compteur automatique (auto-incrémentation).

Pour récupérer ce numéro, nous utilisons la fonction **mysqli\_insert\_id** de la connexion.

```
<?php
$id_carnet = mysqli_insert_id($connex);
?>
```

### **mysqli\_insert\_id**

Retourne l'identifiant automatiquement généré par la dernière requête.

Pour pouvoir ensuite enregistrer la deuxième partie du formulaire, c'est-à-dire une rubrique et une observation :

```
<?php
$sql = "
INSERT INTO carnet_details (
    'idcarnet'
    , 'idrubrique'
```

```

        , 'observation'
) VALUES (
    ".$id_carnet."
    ,".$frm['idrubrique']."
    ,".$htmlentities($frm['observation'])."
);
$qid = mysqli_query($connex,$sql);
if( ! $qid ) die ('Requête invalide : ' . mysqli_error($connex));
?>

```

Ici, nous enregistrons quelques informations : le numéro de la ligne du carnet (idcarnet), ensuite la rubrique (idrubrique) et pour finir les observations.

Il est préférable d'enregistrer la clé que la valeur du champ car si nous effectuons une modification de la valeur du champ (ex : correction orthographique) nous avons juste le contenu du texte à modifiée car la clé, elle ne bouge pas.

Il ne nous reste plus qu'à insérer la photo dans l'enregistrement.

#### d. Manipulations du fichier photo

Une fois insérées toutes les données, occupons nous de la photo. L'avantage de procéder dans cet ordre est que si l'utilisateur n'a pas rempli le champ photo, l'application continuera son déroulement.

Pour l'ajout de la photo, nous allons reprendre la méthode étudiée dans la section concernant le transfert de fichier (cf. chapitre La préparation du développement - Transfert de fichiers). Mais, pour plus de facilité, nous allons renommer le fichier en utilisant le prénom de la fiche du carnet que nous venons de remplir au lieu de garder le nom original du fichier.

Nous changeons ce nom pour obtenir quelque chose de plus rationnel, sinon, lorsque vous regarderez le dossier où se trouve toutes les photos, vous risquez d'y trouver toutes sortes de noms et de chiffres et d'avoir du mal à vous y retrouver. Renommer les fichiers avec les prénoms des contacts permet d'obtenir une structure plus facile à contrôler.

Nous allons créer un fichier dédié à la gestion des fichiers : **fct\_upload.inc.php**. Ainsi, il sera beaucoup plus facile de retrouver les fonctions.

```

<?php
function rename_fichier($newName,$fichier)
{
?>

```

Nous récupérons le nom du dossier de destination des photos qui a été défini au début de notre application :

include/config.inc.php

```

<?php
Global $destDir;
?>

```

Nous utilisons le nom du dossier de sauvegarde des fichiers qui seront utilisés :

```

<?php
$ext = strrchr($fichier, '.');
$newName = strtolower($newName);
?>

```

Nous mettons à part l'extension du fichier et mémorisons le nom du fichier qui sera le prénom de la personne au format chaîne de caractères en minuscule.

Testons maintenant si le fichier existe déjà :

```

<?php
if ( !file_exists($destDir.$newName) )
{
    $i=0;
?>

```

Avant d'effectuer l'opération de transfert, nous devons vérifier si le prénom que nous désirons utiliser se trouve déjà dans le dossier. Le nom du fichier peut déjà exister si un autre utilisateur de votre site Internet, possède dans ses contacts le même prénom. Pour éviter d'écarter le fichier existant, nous ajoutons au nom une valeur numérique

incrémentée jusqu'à ce que nous ne trouvions plus de nom de fichier existant.

```
<?php
  while ( file_exists($destDir.$newName.$ext) )
  {
    $i++;
    if (file_exists($destDir.$newName.$ext))
    {
      $decoupe = explode('_', $newName);
      $newName = $decoupe[0]."_" . $i;
    }
    else
      break;
  }
?>
```

Lorsque nous avons une combinaison prénom avec son numéro qui n'existe pas, nous renommons le fichier avec celle-ci et nous renvoyons à notre application le nouveau nom du fichier que nous avons inséré dans notre espace.

#### **rename**

Renomme un fichier.

```
<?php
rename ($destDir.$fichier,$destDir.$newName.$ext);
return $newName.$ext;
?
?>
```

Ce changement de nom effectué, nous l'enregistrons sur la fiche.

Pour exécuter toute cette procédure, voici la routine dont nous avons besoin :

```
<?php
if (isset($_FILES['fichier']) || !empty($_FILES['fichier']))
  $fichier = $_FILES['fichier'];
if ( $fichier && $fichier != "none" )
{
  $rep=upload($destDir,$fichier);
  if ($rep[0]== TRUE)
  {
    $fichier=rename_fichier($frm['prenom'],$rep[1]);

    $sql = "UPDATE carnet SET photo='".$fichier."' WHERE id='".$id_site."'";
    $qid = mysqli_query($connex,$sql);
    if (!$qid) die('Requête invalide : ' . mysqli_error($connex));
  }
}
?>
```

Lorsque nous avons terminé toutes les manipulations, nous affichons un message de succès.

## **3. Afficher les contacts**

Lorsque l'utilisateur aura enregistré un certain nombre de contacts dans le carnet d'adresses, il lui sera nécessaire de pouvoir visualiser le contenu complet du carnet.

### **a. Visualiser**

Pour effectuer la consultation du carnet, nous allons réaliser plusieurs étapes :

- préparer la requête
- exécuter la requête

- afficher le résultat
- libérer le résultat
- fermer la connexion avec la base de données

carnet-view.php

```
<?php
$sql="select carnet.*,carnet.id as idtmp,user.id,user.idclef
FROM carnet,user
WHERE user.id=carnet.iduser
AND carnet.iduser='".$_SESSION['iduser']."'"
AND user.idclef='".$_SESSION['idclef']."' ";
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
?>
```

Pour sélectionner le carnet d'adresses en fonction de l'utilisateur, nous mettons en place un filtre avec les critères que nous avons mémorisés dans la session, c'est-à-dire le numéro de clef variable et le numéro de ligne du possesseur du compte.

Nous effectuons l'affichage des informations principales sous la forme d'un tableau HTML. Les informations qui sont affichées sont considérées comme principales, permettant ainsi d'obtenir plus rapidement les informations dont nous pouvons avoir besoin.

```
<?php
echo "<table border=1 width=100%>";
echo "<tr>";
echo "<td>Nom</td>";
echo "<td>Prenom</td>";
echo "<td>Email</td>";
echo "<td>Tel</td>";
echo "<td>Portable</td>";
echo "<td>Edit</td>";
echo "</tr>";

while( $row=mysqli_fetch_object( $qid ) )
{
  echo "<tr>";
  echo "<td>".$row->nom."</td>";
  echo "<td>".$row->prenom."</td>";
  echo "<td>".$row->email."</td>";
  echo "<td>".$row->tel."</td>";
  echo "<td>".$row->portable."</td>";
  echo "<td><a href=carnet-view-edit.php?id=$row->idtmp class=links>Cliquez
ici</a></td>";
  echo "</tr>";
}
echo "</table>";
?>
```

Nous libérons le résultat et fermons la connexion.

```
<?php
mysqli_free_result($row);
mysqli_close($connex);
?>
```

 Bien sûr pour chaque ligne affichée, nous allons pouvoir éditer, consulter et modifier le contenu de la fiche correspondante du carnet d'adresses. Nous le verrons un peu plus loin dans l'ouvrage.

## b. Filtrer

Quand nous parlons d'un carnet d'adresses, nous pensons au petit calepin que tout le monde a eu au moins une fois dans ses mains. Ce petit calepin possède un système de classement alphabétique, permettant ainsi de retrouver plus facilement le nom ou prénom de la personne recherchée.

Nous allons donc proposer un filtre alphabétique pour permettre de sélectionner une lettre et d'afficher tous les noms qui commencent par cette lettre.

Pour réaliser cette fonctionnalité, nous allons insérer une petite boucle pour afficher les lettres de l'alphabet et ainsi éviter de voir 26 lignes correspondant aux 26 lettres de l'alphabet. Nous utiliserons un petit tableau pour gagner du temps, et surtout économiser sur les lignes de programmation, comme ceci :

carnet-view.php

```
< ?php
$lettre=array('A','B','C','D','E','F','G','H','I','J','K',
              'L','M','N','O','P','Q','R','S','T','U','V',
              'W','X','Y','Z','Tous');
for ($i=0;$i<26;$i++)
{
    echo "<a href=".htmlentities($_SERVER['PHP_SELF'])."?filtre=
".$lettre[$i]." class=links>".$lettre[$i]."</a>";
    echo "-";
}
?>
```

Lors de la sélection d'une des lettres proposées, nous rechargeons la page en cours, ce qui nous permet d'afficher le résultat correspondant à la lettre concernée comme ceci :

Nous affichons le mot **Tous** à la fin de la liste alphabétique, car le visiteur doit pouvoir revenir à l'affichage total de la liste sans passer par le menu.

```
<?php
if (isset ($GET['filtre'])) $filtre=verif_GetPost($_GET['filtre']);

if (!empty($filtre) && $filtre!="Tous")
{
    $filtre= "AND ( nom LIKE '" . $filtre . "%' )   ";
}
else
    $filtre= " ";
?>
```

Sur sélection d'une des lettres et avant d'accéder à la préparation de la requête, nous devons effectuer une vérification des données provenant du lien Internet.

Nous effectuons un test pour vérifier si le mot **Tous** n'est pas présent ou que le champ possède bien une valeur.

Si ces critères ont été renseignés, nous allons ajouter dans la requête une restriction sélectionnant tous les noms qui commencent par la lettre qui a été choisie.

Si ce n'est pas le cas, nous laissons un caractère vide, permettant ainsi à la requête SQL de s'exécuter normalement sans restriction supplémentaire.

```
<?php
$sql="select carnet.* ,user.id,user.idclef
      FROM carnet,user
      WHERE user.id=carnet.iduser
      AND carnet.iduser='". $_SESSION['iduser']."'"
      AND user.idclef='". $_SESSION['idclef']."'"
      $filtre
      ";
?>
```

Le reste de l'affichage ne change pas donc nous n'en parlerons pas de nouveau.

## c. Trier

Lors de l'affichage d'une liste, il est souvent nécessaire de pouvoir effectuer un petit tri pour une visualisation plus rapide ou pour obtenir une autre présentation.

Pour réaliser le tri, nous allons utiliser une fonction standard du SQL, il s'agit de :

### **ORDER BY**

Trier le champ X dans un ordre croissant.

### **DESC**

S'insère après le champ pour effectuer un tri décroissant, par exemple ORDER BY ... DESC.

Dans notre application, nous allons autoriser le tri sur le nom ou le prénom :

```
<?php
if (isset ($_GET['tri'])) $tri=verif_GetPost($_GET['tri']);
if (isset ($tri) && $tri=="nom") $order="ORDER BY nom ";
else $order="";
if (isset ($tri) && $tri=="prenom") $order="ORDER BY prenom ";

if (isset ($_GET['ordre'])) $ordre=verif_GetPost($_GET['ordre']);

if ($tri && $ordre!="DESC") $ordre = "DESC"; else $ordre = "";
?>
```

Lorsque l'utilisateur clique pour effectuer un tri, la page enregistre la colonne qui a été sélectionnée et se réaffiche automatiquement triée. Si le visiteur sélectionne de nouveau la même colonne, le tri s'effectuera dans le sens contraire car nous mémorisons le sens du tri.

Pour cette manipulation le lien HTML s'occupe de tout, il se compose ainsi :

```
< ?php
echo "<a href=".htmlentities($_SERVER['PHP_SELF'])."?tri=
nom&ordre=$ordre class=links>Tri</a>";
?>
```

Ce lien indique quelle colonne a été sélectionnée, et avec une autre valeur l'ordre actuel.

## **d. Gérer plusieurs pages**

Lorsque le carnet est bien rempli, nous pouvons obtenir un résultat très volumineux pour certaines lettres de l'alphabet.

Par conséquent, il est toujours intéressant de pouvoir limiter l'affichage du nombre de lignes sur un écran.

Nous allons gérer des numéros de pages pour disposer de numéros en bas du tableau pour naviguer entre elles plus facilement.

Pour optimiser cette partie, nous utilisons certains noms de variables :

### **\$nb\_pages**

Nombre de lignes par pages.

### **\$start**

Position de départ.

```
<?php
$nb_pages=5;

if (isset ($_GET['start'])) $start=verif_GetPost($_GET['start']); else $start=0;
if(isset($start) && !$start)
    $start=0;
else
    $start = $start;
?>
```

Ici, le nombre de lignes par page retenu est 5, mais il est tout à fait possible de modifier cette valeur pour permettre

l'affichage de 10, 20 ou 50 lignes par page.

Nous récupérons le numéro de la page permettant de se positionner dans ce mini navigateur. Bien sûr, nous utilisons tous les filtres de contrôle et de sécurité dont nous avons maintenant l'habitude.

Lorsque nous avons déterminé la valeur des variables, nous insérons ces valeurs dans la requête pour lui permettre d'afficher les bonnes lignes.

Pour effectuer l'affichage de ces lignes, nous utilisons la fonction standard SQL **LIMIT**. Cette clause se construit de la façon suivante : on indique le numéro de la ligne suivi du nombre de lignes à retenir.

Dans notre application, nous avons retenu 5 lignes par page, donc si nous désirons afficher la page 4, nous effectuons le calcul comme ceci :

Numero de page \* 5 = \$start

Ce que nous obtenons :  $4*5 = 20$ .

La valeur de \$start est 20.

Et la variable \$limit vaut 5.

```
<?php
$sql="select carnet.*,user.id,user.idclef
      FROM carnet,user
      WHERE user.id=carnet.iduser
            AND carnet.iduser='".$_SESSION['iduser']."'"
            AND user.idclef='".$_SESSION['idclef']."'"
            $filtre
            $order
            LIMIT $start,$nb_pages
            ";
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysql_error($connex));
?>
```

Le résultat que nous obtenons ira des lignes 20 à 25.

Sous le tableau, nous affichons sur une ligne les numéros des pages qui correspondent.

De plus, nous en profitons pour afficher à chaque extrémité le mot **début** et le mot **fin**, nous permettant ainsi de naviguer sur les extrémités plus facilement.

Pour commencer, nous devons afficher le mot **début** si nous ne sommes pas au début du compteur, c'est-à-dire si la variable \$start possède une valeur numérique.

```
< ?php
if($start)
{
echo "<a href=".$_SERVER['PHP_SELF']."?start=".($start-$nb_pages)." class=links>";
echo "Début ";
echo "</a>";
echo "&nbsp;";
}
?>
```

Pour déterminer le nombre de pages à afficher, nous allons reprendre la requête utilisée pour afficher le tableau. Nous allons lui demander de compter le nombre de lignes totales avec la fonction SQL standard : **COUNT**.

```
<?php
$sql="select COUNT(*)
      FROM carnet,user
      WHERE user.id=carnet.iduser
            AND carnet.iduser='".$_SESSION['iduser']."'"
            AND user.idclef='".$_SESSION['idclef']."'"
            $filtre
            $order
            ";
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));

$row=mysqli_fetch_row($qid);
?>
```

Nous récupérons la valeur qui nous est communiquée et, dans un premier temps, vérifions si le chiffre que nous avons obtenu est supérieur au nombre de lignes que nous devons afficher par page. Si c'est le cas, nous commençons à afficher les numéros de pages possibles.

Nous allons comparer chaque numéro et tester si le numéro de la page qui est affichée correspond au numéro de page du compteur. Si c'est le cas, nous affichons le numéro de la page dans un style d'écriture différent pour montrer où nous sommes situés dans l'ensemble du catalogue.

```
<?php
if($row[0]>$nb_pages)
{
    $start++;
    for($index=0;($index*$nb_pages)<$row[0];$index++)
    {

        $npage=$index+1;

        if ($start!=$npage)
        {
            echo "<b>";
            echo "<a href=". $_SERVER[ 'PHP_SELF' ]."?start=". $index*$nb_pages . "
class=links>$npage</a>";
            echo "</b>";
            echo "&nbsp;";
        }
        else
        {
            echo $npage;
            echo "&nbsp;";
        }
    }
}
?>
```

Il ne nous reste plus qu'à afficher le mot **fin** qui correspond à la dernière page du carnet d'adresses que nous avons sélectionné. Si nous nous trouvons sur la dernière page, nous ne l'affichons pas.

```
<?php
if($row[0]>($start+$npage))
{
echo "<a href=". $_SERVER[ 'PHP_SELF' ]."?start=". ($start+$npage) . " class=links>";
echo " Fin";
echo "</a>";
}
?>
```

Chaque numéro correspond à un lien. Si ce lien est choisi, nous rechargeons la page avec les nouveaux critères de sélection.

## 4. Éditer un contact

Dans notre application l'utilisateur pourra afficher le contenu du carnet d'adresses, comme nous venons de le voir dans les pages précédentes. Il pourra choisir une des entrées proposées, pour en consulter le détail et aussi effectuer les modifications si cela est nécessaire.

Nous voyons ci-dessous l'écran complet :

Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils 2

http://localhost/eni/application/app-php-mysql/carnet-view-edit.php? Google

Google Rechercher Mes favoris PageRank Paramètres

**Accueil**      **Fiche :**

<b>Votre compte</b>	Genre : Monsieur	Rubrique	Observations
<b>Rubriques</b>	Nom : VILLENEUVE	site internet	www.hello-design.fr
<b>Votre Carnet</b>	Prenom : Christophe *	Editeur	www.editions-eni.fr
<b>Ajout</b>	Adresse :		
<b>Consult/Modif</b>	Code postal :		
<b>Recherche</b>	Ville :		
<b>Administration</b>	Email : livre@hello-design.fr *		
<b>Exportation</b>	Téléphone :		
<b>Déconnection</b>	Portable :		
<b>Contact</b>	Photo :	 <a href="#">Voir plan avec GoogleMaps</a>	
<input type="button" value="Mise A Jour"/> <input type="button" value="Supprimer"/>			

Application - Php - Mysqli - (c) Hello-Design - Editeur ENI

Terminé

La fiche comporte deux parties :

- une partie sur la gauche contenant les informations sur le contact ;
- une partie à droite avec les informations utiles qui concernent ce contact.

Pour afficher et gérer la partie de droite qui doit contenir les rubriques qui ont été remplies à partir du menu Rubrique, nous allons traiter le résultat sous la forme d'un tableau.

```
<?php
$sql="SELECT carnet_details.idcarnet,carnet_details.id as idtmp,carnet_details.
idrubrique,carnet_details.observation,carnet.id,carnet.carnetclef
FROM carnet,carnet_details
WHERE carnet.id=carnet_details.idcarnet AND carnet.carnetclef='".$carnetclef" ';
assert ('mysqli_query($connex, $sql)');
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
?>
```

Nous déterminons le nombre de lignes renvoyées pour réaliser une boucle d'affichage.

```
<?php
$nligne= mysqli_num_rows($qid);
$nligne=0;
while ($row=mysqli_fetch_array( $qid))
{
  echo "<tr>";
  echo "<td valign=top>";
?>
```

Pour chaque ligne, nous allons afficher sous la forme d'une liste déroulante les rubriques personnelles que nous avons déterminées.

```

<?php
echo "<select name=\"idrubrique[]\">";
echo ligne_selected(" ","-1",$frm['idrubrique'][$i]);

$sql="SELECT id,iduser,nom FROM rubrique WHERE iduser='". $_SESSION['iduser']."' "
ORDER BY nom";
assert ('mysqli_query($connex, $sql)');
$rubrique=mysqli_query($connex, $sql);
if (!$rubrique) die('Requête invalide : ' . mysqli_error($connex));
if (mysqli_num_rows($rubrique)!=0)
{
    while( $row2=mysqli_fetch_array( $rubrique) )
    {
        echo ligne_selected($row2['nom'],$row2['id'],$row['idrubrique']);
    }
}
echo "</select>";
echo "</td>";
?>

```

Nous affichons pour la même ligne les observations et commentaires importants. Pour respecter la présentation de la saisie du visiteur dans ce champ, nous utilisons la fonction **nl2br** :

#### **nl2br**

Insère un retour chariot à la ligne HTML à chaque nouvelle ligne.

```

<?php
echo "<td>";
echo "<textarea name=\"observation[]\" cols=\"30\" rows=\"3\">".stripslashes
(nl2br($row[observation])).</textarea>";
?>

```

Nous mémorisons l'identifiant (idtmp) de chaque ligne pour permettre d'effectuer la mise à jour du formulaire.

```

<?php
echo "<input name=\"iddetails[]\" type=\"hidden\" value=\"$row['idtmp']\">";
echo "</td>";
echo "</tr>";
$nligne++;
}
?>

```

Nous libérons le résultat.

```

<?php
    mysqli_free_result($qid);
?>

```

Nous avons deux possibilités pour insérer une nouvelle ligne (nouveau couple rubrique/observation= pour la fiche contact en cours :

- Un bouton **Ajouter** : plus rapide à réaliser. Le principe consiste à afficher un autre formulaire si on clique sur ce bouton.
- Une ligne vide : consiste à afficher obligatoirement une nouvelle ligne, qui serait utilisable de suite.

Nous allons privilégier le deuxième choix.

Lorsque l'utilisateur aura effectué les modifications désirées, il sélectionnera le bouton **Mise A jour**.

Cette mise à jour va être réalisée en plusieurs étapes :

- La mise à jour de la fiche.
- La mise à jour des détails.

## a. Mise à jour de la fiche

Avant d'effectuer la mise à jour de la fiche, nous allons effectuer les vérifications classiques de sécurité et si les champs obligatoires ont été correctement renseignés.

Rappelons que ces champs obligatoires sont le prénom et l'e-mail.

Les différents tests nécessaires effectués, nous pouvons procéder à la mise à jour comme ceci :

carnet-vieux-edit.php

```
<?php
function update_fiche(&$frm)
{
global $connex;

$sql = "UPDATE carnet SET
genre='".$frm[genre]."'
,nom='".$strtoupper(htmlentities($frm['nom']))."'
,prenom='".$ucfirst( strtolower(htmlentities($frm['prenom'])))."'
,adresse1='".$htmlentities($frm['adresse1'])."'
,adresse2='".$htmlentities($frm['adresse2'])."'
,codepostal='".$frm['codepostal'].'
,ville='".$htmlentities($frm['ville']).'
,tel='".$htmlentities($frm['tel']).'
,Portable='".$htmlentities($frm['Portable']).'
,email='".$htmlentities($frm['email']).'
WHERE carnetclef='".$frm['carnetclef']."'"
";
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
}
?>
```

La fonction de mise à jour (update\_fiche) se décompose en plusieurs étapes :

- la préparation de la requête avec tous les critères de présentation, c'est-à-dire le nom de famille en majuscules, le prénom tout en minuscules sauf la première lettre ;
- l'exécution de la requête.

## b. Mise à jour des détails

Les détails de la fiche (partie de droite) vont être traités différemment car nous avons à gérer un tableau. Nous devons mémoriser le numéro de la ligne (id) pour l'enregistrer dans la table détail (carnet\_details). Pour information, nous ne pouvons pas mémoriser la clef car elle est variable, mais plutôt le numéro de la ligne.

```
<?php
function fiche_details(&$frm)
{
global $connex;

$sql="select id FROM carnet WHERE carnetclef='".$frm['carnetclef']."' ";
assert ('mysqli_query($connex, $sql)');
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
$row=mysqli_fetch_object( $qid);
?>
```

Pour effectuer la mise à jour du carnet-details, nous allons, à partir du nombre de lignes, effectuer la modification nécessaire provenant du formulaire sauf la dernière qui sera gérée un peu différemment. Ainsi, les lignes qui possèdent des informations seront considérées comme modifiables.

```
<?php
$nligne=$frm['nligne']-1;
```

```

for ($i=0;$i<=$nligne;$i++)
{
}

$sql = "UPDATE carnet_details SET
    idcarnet='".$row->id."'
    ,idrubrique='".$frm['idrubrique'][$i]."'
    ,observation='".$htmlentities($frm['observation'][$i])."'
    WHERE id='".$frm['iddetails'][$i]."'
    ";

assert ('mysqli_query($connex, $sql)');
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
}
?>

```

Nous avons effectué la totalité de la mise à jour, nous pouvons désormais nous occuper de la dernière ligne qui était vide lorsque nous sommes arrivés sur cet écran.

Nous effectuons un test pour vérifier si cette ligne possède des valeurs. Si la ligne est toujours vide, nous continuons notre chemin.

Par contre, si la ligne possède des valeurs, nous devons insérer ces données dans la base de données.

```

<?php
if ($frm['Newidrubrique']!=-1)
{
    $sql = "
        INSERT INTO carnet_details (
            'idcarnet'
            , 'idrubrique'
            , 'observation'
        ) VALUES (
            '".$row->id."'
            , '".$frm['Newidrubrique']."' .
            '".$htmlentities($frm['NewObservation'])."'
        )";

    $qid = mysqli_query($connex, $sql);
    if(! $qid ) die ('Requête invalide : ' . mysqli_error
    ($connex));
}
}
?>

```

### c. Mise à jour de la photo

Nous avons deux possibilités :

- insérer une nouvelle photo indisponible lors de la création de la fiche ;
- afficher la photo que nous avons dans la base de données.

Si nous devons insérer une photo, nous allons procéder à quelques opérations pour l'ajouter aux autres photos.

Nous allons faire appel à différentes fonctions pour effectuer le transfert du fichier retenu de l'ordinateur vers le serveur.

Lorsque cette étape est réalisée, nous renommons le fichier correctement en utilisant le prénom du contact et enregistrons les informations dans la base de données.

carnet-view-edit.php

```

<?php
require_once ("include/fct_upload.inc.php");

if (isset($_FILES['fichier'])||!empty($_FILES['fichier'])) $fichier =

```

```

$_FILES['fichier'];
if ( $fichier && $fichier != "none")
{
    $rep=upload($destDir,$fichier);
    if ($rep[0]== TRUE)
    {
        $fichier=rename_fichier($frm['prenom'],$rep[1]);
        $sql = "UPDATE carnet SET photo='". $fichier ."'"
        WHERE carnetclef='".$frm['carnetclef']."' ";
        assert ('mysqli_query($connex, $sql)');
        $qid = mysqli_query($connex, $sql);
        if (!$qid) die ("Probleme : " . mysqli_error($connex));
    }
}
?>

```

Concernant la gestion du transfert de la photo, nous vous conseillons de vous rendre à la partie abordant ce thème puisqu'il s'agit des mêmes fonctions (cf. chapitre La préparation du développement - Transfert de fichiers).

## 5. Supprimer un contact

L'utilisateur aura besoin de supprimer une fiche avec ses différents liens.

Pour effectuer cette opération, il devra accéder à la fiche de son choix et cliquer sur le bouton **Supprimer**.

Pour être certains qu'il ne s'agit pas d'une erreur, nous demanderons la confirmation du choix par l'intermédiaire d'une fonction JavaScript.

```

<input type="submit" name="action" value="Supprimer"
onClick="Javascript:return confirm('Êtes-vous sûr de vouloir
enlever cette ligne ?');">

```

Après confirmation du choix, nous allons effectuer l'opération.

Signalons qu'il est également possible de demander la confirmation de la suppression de la fiche par l'intermédiaire d'un formulaire de saisie du mot de passe comme nous l'avons vu dans le chapitre précédent.

Pour effacer les lignes correspondant à la fiche sélectionnée, nous devons effectuer les opérations suivantes :

Nous sélectionnons le numéro de la fiche que nous obtenons par l'intermédiaire de la clef du carnet.

```

<?php
$sql="select id FROM carnet WHERE carnetclef='".$frm['carnetclef']."' ";
assert ('mysqli_query($connex,$sql)');
$qid = mysqli_query($connex,$sql);
if (!$qid) die ("Probleme : " . mysql_error());
$row=mysqli_fetch_object( $qid );
?>

```

Lorsque nous avons récupéré le numéro de la fiche choisie, nous allons supprimer les lignes correspondant aux différentes rubriques de la fiche, c'est-à-dire la partie droite de celle-ci.

```

<?php
$sql="DELETE FROM carnet_details WHERE idcarnet='".$row->id."' ";
assert ('mysqli_query($connex,$sql)');
$qid = mysqli_query($connex,$sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
?>

```

Ensuite, nous supprimons la fiche proprement dite avec le numéro de clef du carnet.

```

<?php
$sql="DELETE FROM carnet WHERE carnetclef='".$frm['carnetclef']."' ";
assert ('mysqli_query($connex,$sql)');
$qid = mysqli_query($connex,$sql);
if (!$qid) die('Requête invalide : ' . mysql_error());
?>

```

Nous affichons un message de succès concernant cette opération.

```
<?php
require_once "head.inc.php";
echo "<h1><center>";
echo "Fiche supprimée";
echo "</center></h1>";
require_once "footer.inc.php";
exit;
?>
```

Nous avons effectué la suppression de toute la fiche.

Nous procémons dans cet ordre, car si la fiche proprement dite est supprimée avant les détails, nous n'aurions pas été capables de retrouver le numéro de la fiche correspondant aux détails à supprimer.

C'est pour cela qu'il est toujours préférable de supprimer les enregistrements détails et de terminer par la fiche principale.

## La gestion de l'administration

Nous traitons toujours de la saisie, mais consacrée aux droits des utilisateurs. Il s'agit de la partie administration du site, qui va nous permettre de créer de nouveaux comptes et de leur attribuer des droits. Ces droits conditionnent l'accès à cette partie.

► Une petite parenthèse : imaginez que les utilisateurs qui vont avoir un compte, possèdent un carnet d'adresses. Pour gérer ces comptes il faut une personne qui soit responsable, donc un administrateur. L'administrateur peut créer des nouveaux comptes, les modifier ou les bloquer. Le blocage d'un compte permet de bloquer l'accès d'un compte sans supprimer les informations du carnet, c'est-à-dire en cas d'abus de l'utilisation de l'application.

Une personne qui n'est pas considérée comme administrateur ne verra pas ce menu et ne pourra pas y accéder.

Bien sûr, nous aurons la possibilité de créer plusieurs administrateurs pour gérer notre application.

Cette partie de l'application est réalisée avec le format PDO. Elle est aussi disponible avec les formats MySQL et MySQLi en téléchargement sur le site de l'éditeur.

Pour communiquer avec la base de données en PDO, nous devons activer les extensions suivantes :

- extension=php\_pdo.dll
- extension=php\_pdo\_mysql.dll

Pour activer ces extensions, vous devez vous reporter à la section PDO du chapitre La préparation du développement, qui explique comment activer ces extensions.

### 1. Ajouter un compte

Pour ajouter un compte, nous allons établir un formulaire au format HTML.

Mais avant d'afficher ce formulaire, il faut s'assurer que la personne qui s'est connectée possède bien le niveau administrateur.

Admin.php

```
<?php
if ($_SESSION['niveau']!="admin") header("Location:index.php");
?>
```

Si le visiteur essaye de se connecter sur cette page et qu'il ne possède pas le niveau requis, il sera automatiquement redirigé vers la page index du site Internet.

Si le visiteur possède bien le niveau nécessaire pour accéder à la partie administrateur, nous pouvons afficher le formulaire ci-dessous :

Dans ce formulaire, nous allons imposer la saisie obligatoire de toutes les zones.

Nous allons aussi par défaut attribuer aux nouveaux comptes qui seront enregistrés un niveau utilisateur.

Lors de la validation du formulaire, nous vérifierons que tous les champs ont été correctement remplis.

Pour effectuer le contrôle du formulaire, nous allons passer par une fonction et nous garderons la technique de mémorisation des erreurs dans un tableau vu dans les pages précédentes (carnet, rubrique).

```
<?php
function valide_form(&$frm, &$erreurs)
{
GLOBAL $cnx;

$erreurs = array();
$msg = array();
?>
```

Nous vérifions si le champ Login possède bien une valeur, sinon nous mémorisons l'erreur et son message d'alerte.

```
<?php
if (empty($frm['nlogin'])) )
{
$erreurs['nlogin'] = true;
$msg['nlogin'] = "Absence de Login";
}
?>
```

Nous contrôlons dans la base de données si ce login (identifiant) n'est pas déjà utilisé dans la table.

Pour réaliser ce contrôle, nous effectuons un certain nombre d'opérations. Tout d'abord, nous préparons la requête SQL en prenant comme critère l'identifiant qui a été saisi à partir du formulaire.

Ensuite nous l'exécutons et nous récupérons la valeur de la colonne pour savoir si cet identifiant est présent dans la table. Si aucune ligne n'existe, nous continuons le déroulement des différents tests. Si la ligne existe, nous signalons le problème.

```
<?php
$sql="SELECT 1 FROM user WHERE login = '".$frm['nlogin']."' ";
$qid = $cnx->prepare($sql);
```

```

$qid->execute();
$nbre = $qid->fetchColumn();
if ($nbre>0)
{
    $erreurs['nlogin'] = true;
    $msg['nlogin'] = " Cet identifiant existe déjà";
}
?>

```

Pour valider le mot de passe, nous devons effectuer d'autres tests de contrôle.

Tout d'abord nous allons vérifier si le premier champ de saisie du mot de passe possède une valeur.

```

<?php
if (empty($frm['npassword']))
{
    $erreurs['npassword'] = true;
    $msg['npassword'] = "Absence de mot de passe";
}
?>

```

Nous vérifions ensuite si le second champ de saisie du mot de passe possède bien une valeur.

```

<?php
if (empty($frm['npassword2']))
{
    $erreurs['npassword2'] = true;
    $msg['npassword2'] = "Absence de mot de passe";
}
?>

```

Si les deux champs **Mot de passe** et **Confirmer mot de passe** possèdent bien une valeur, nous les comparons et si la saisie est différente, nous signalons le problème.

```

<?php
if ($frm['npassword']<>$frm['npassword2'])
{
    $erreurs['npassword2'] = true;
    $msg['npassword2'] = " mots de passe différents";
}
?>

```

Nous effectuons aussi un test sur la validité et l'éventuelle présence de l'e-mail dans la liste de comptes. Pour effectuer ce test, nous préparons le champ **email** au format entités HTML

```

<?php
$email=htmlentities($_POST['email'], ENT_QUOTES, 'UTF-8');
?>

```

Nous vérifions si le champ n'est pas vide.

```

<?php
if (empty($email))
{
    $erreurs['email'] = true;
    $msg['email'] = "Absence d'email";
}
?>

```

Si le champ possède bien une valeur, nous contrôlons si l'e-mail correspond au format standard avec une expression régulière comme nous l'avons étudié dans le chapitre La préparation du développement, section preg\_Replace.

```

<?php
elseif (!preg_match('/^[[[:alnum:]]([-_.]?[[[:alnum:]]])*@[[:alnum:]]([-_.]?[[[:alnum:]])*\.( [a-z]{2,4})$/',$email))
{
    $erreurs['email'] = true;
    $msg['email'] = " Mauvais format d'email";
}
?>

```

Ensuite, nous devons vérifier si l'e-mail qui a été enregistré n'existe pas dans la base de données. Pour effectuer ce contrôle, nous allons préparer une requête SQL pour ensuite l'exécuter.

Le résultat que nous obtenons permet de vérifier si l'e-mail existe ou pas, en contrôlant si nous obtenons une valeur supérieure à 0.

```
<?php
$sql="SELECT 1 FROM user WHERE email = '". $frm['email']."' ";
$qid = $cnx->prepare($sql);
$qid->execute();
if ($qid->fetchColumn()>0)
{
    $erreurs['email'] = true;
    $msg['email'] = "L'email existe déjà";
}
?>
```

Nous testons la liste déroulante Niveau pour vérifier que la valeur sélectionnée correspond à l'une des valeurs possibles, stockées dans un tableau.

```
<?php
$niveau=array ('user','admin');
if (!in_array ($frm['niveau'],$niveau))
{
    $erreurs['niveau'] = true;
    $msg['niveau'] = " Problème avec la liste déroulante";
}
return $msg;
}
?>
```

Ayant contrôlé toutes les zones de saisie, nous pouvons passer à l'étape suivante, c'est-à-dire insérer l'enregistrement dans la base de données.

Pour insérer les données qui viennent d'être saisies, contrôlées et validées, nous allons préparer la requête SQL.

```
<?php
function insere_compte(&$frm)
{
global $cnx;
    $creat_compte=date("Y-m-j");
    $sql = "
        INSERT INTO user (
            'niveau'
            , 'idclef'
            , 'login'
            , 'password'
            , 'email'
            , 'date_creation'
            , 'page'
        ) VALUES (
            '$frm[niveau]'
            , '$recup_clef()'
            , '$htmlentities($frm[nlogin'])'
            , '$md5($frm[npassword])'
            , '$htmlentities($frm[email])'
            , '$creat_compte'
            , 'compte.php'
        ) ";
}
?>
```

Lors de la préparation de la requête, nous récupérons certaines informations que nous avons déjà utilisées dans notre application.

**idclef** : il s'agit de l'utilisation de la clef automatique, grâce à la fonction **recup\_clef** nous obtenons une clef unique autorisant ainsi le possesseur du compte à naviguer sans soucis.

**page** : nous mémorisons le nom de la page **compte.php** par défaut, ce qui nous permet si nous possédons de nombreux comptes et de nombreuses applications, de rediriger notre personne sur les sites ou les pages différentes.

Par exemple, nous pourrions faire en sorte que le visiteur, s'il possède un niveau utilisateur et lorsqu'il se connecte à son compte, soit automatiquement dirigé sur la page carnet.php de l'application et qu'un possesseur du niveau administrateur soit dirigé lui directement vers la page admin.php.

Nous effectuons ensuite la connexion de la transaction en PDO en effectuant la préparation et la validation de la requête SQL qui ont été prévues précédemment.

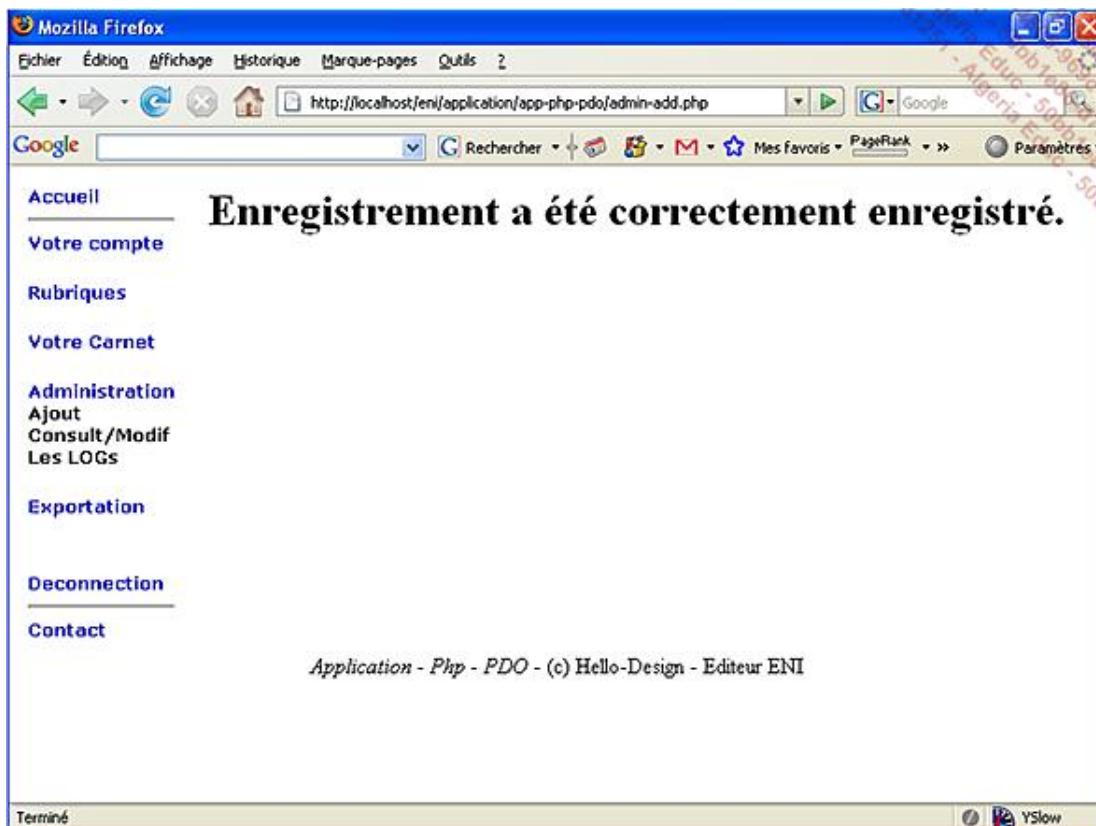
Ensuite nous exécuterons cette fonction.

```
<?php
$cnx->beginTransaction();
$qid=$cnx->prepare($sql);
$qid->execute();
?>
```

Pour finir, nous libérons et fermons la connexion avec la base de données.

```
<?php
$cnx->commit();
$cnx = null;
?>
```

Lorsque nous avons inséré l'enregistrement, nous pouvons signaler le succès de l'opération.



## 2. Afficher les comptes

Avant de commencer à afficher le contenu de la page, nous devons vérifier si le visiteur possède les droits nécessaires avec le filtre de niveau que nous avons vu dans le chapitre précédent. Si le visiteur possède le niveau administrateur, il pourra accéder à cette page.

Pour afficher les comptes présents dans la base de données, nous réalisons une routine nous permettant de visualiser le contenu de la table pour connaître les utilisateurs inscrits de notre site Internet.

Les colonnes que nous affichons sont :

- l'identifiant

- le niveau d'accès actuel
- l'e-mail
- la date de création
- la date du dernier passage. Nous n'en avons pas parlé à la création du compte car c'est un champ qui se remplit automatiquement lorsque le visiteur se connecte sur son compte. Ainsi, nous pouvons connaître les personnes qui utilisent leur compte.

Pour chaque ligne, nous avons la possibilité d'accéder au détail du compte ; comme ceci nous pouvons modifier certaines informations grâce à un lien **Cliquez ici** que nous aborderons dans le chapitre suivant.

	Login	Niveau	Email	Date creation	dernier passage	Edit
<b>Votre compte</b>	test	user	webmaster@hello-design.fr	25-06-2007	14-01-2008	<a href="#">Cliquez ici</a>
<b>Rubriques</b>	admin	admin	livre@hello-design.fr	01-08-2007	17-01-2008	<a href="#">Cliquez ici</a>

Pour réaliser cette page, nous allons établir une requête SQL en préparant au préalable la connexion vers la base de données.

```
<?php
$sql="select * from user";
$qid=$cnx->prepare($sql);
$qid->execute();
?>
```

Pour ensuite exécuter la requête et afficher le résultat en utilisant une boucle.

```
<?php
echo "<table border=1 width=100%>" ;
echo "<tr>" ;
echo "<td>Login</td>" ;
echo "<td>Niveau</td>" ;
echo "<td>Email</td>" ;
echo "<td>Date creation</td>" ;
echo "<td>dernier passage</td>" ;
echo "<td>Edition</td>" ;
echo "</tr>" ;
while( $row=$qid->fetch(PDO::FETCH_OBJ) )
```

```

{
echo "<tr valign=top>";
echo "<td>$row->login</td>";
echo "<td>$row->niveau</td>";
echo "<td>$row->email</td>";
echo "<td>".datefr($row->date_creation). "</td>";
echo "<td>".datefr($row->date_lastpass). "</td>";
echo "<td><a href=admin-view-edit.php?clef=$row->idclef class=links>Cliquez
ici</a></td>";
echo "</tr>";
}
?>

```

Nous fermons le curseur et la connexion.

```

<?php
$qid->closeCursor();
$cnx = null;

echo "</table>";
?>

```

### 3. Éditer un compte

Pour effectuer l'édition de la ligne souhaitée, nous devons nous reporter à la partie précédente ; nous allons afficher la liste des membres et si nous sélectionnons une des lignes comme ceci :

```

<?php
echo "<td><a href=admin-view-edit.php?clef=$row->idclef class=links>Cliquez
ici</a></td>";
?>

```

Nous utilisons la clef que nous avons générée pour sécuriser le numéro **id**.

L'activation du lien d'édition amène l'utilisateur sur un nouveau formulaire.

Accueil      Compte de :  
**Votre compte**      Login : test  
**Rubriques**      Mot de Passe : \*\*\*\*\*  
**Votre Carnet**      Email : webmaster@hello-desi\*  
**Administration**      Niveau : Utilisateur  
Ajout      \*  
Consult/Modif  
Les LOGs  
  
**Exportation**  
  
**Déconnection**  
  
**Contact**  
  
*Application - Php - PDO - (c) Hello-Design - Editeur ENI*

Lorsque l'utilisateur clique sur le bouton **Mise A Jour**, nous allons vérifier que les données sont toujours correctes.

Pour effectuer les tests de validation, nous utiliserons les mêmes tests que ceux étudiés dans le chapitre précédent, c'est-à-dire :

- e-mail : pour le champ Email, nous vérifions que le champ n'est pas vide et qu'il se trouve au bon format.
- Niveau utilisateur : nous effectuons un test de validation de la liste pour être certain qu'il n'y a pas eu de changement.

Lorsque la vérification du contenu des champs s'est déroulée avec succès, nous passons à la mise à jour des données. Nous allons préparer la requête SQL et établir une connexion avec le serveur de base de données, ce qui permettra d'exécuter la mise à jour. Si la vérification du contenu des champs ne se passe pas correctement, un ou plusieurs messages apparaissent signalant les erreurs (par exemple, le champ est obligatoire).

```
<?php
function update_compte(&$frm)
{
Global $cnx;

$sql = "UPDATE user SET
niveau='".$frm['niveau']."' ,
email='".$htmlentities($frm['email'])."'
WHERE idclef='".$frm['clef']."' "
;

$cnx->beginTransaction();
$qid=$cnx->prepare($sql);
$qid->execute();
?>
```

Nous fermons le curseur et la connexion.

```
<?php
$cnx->commit();
$cnx = null;
}
?>
```

# Les logs

Les logs sont utilisés pour garder une trace de la navigation de nos visiteurs, les possesseurs des comptes. Ils nous servent aussi à connaître les provenances des tentatives de pénétration (attaques).

## 1. Enregistrement

Pour mémoriser la navigation du visiteur, sur chaque page qui sera affichée nous allons insérer une petite fonction qui nous permettra d'enregistrer certaines informations.

Nous allons tout d'abord vérifier si le dossier contenant les logs existe. Si ce n'est pas le cas, nous créons ce dossier et nous lui ouvrons les droits en écriture et lecture.

Admin-log.php

```
<?php
if (!is_dir($destLog))
{
    if (!@mkdir($destLog))
    {
        return array(false,'Erreur lors de la création du dossier $destDir');
    }
}
@chmod($destLog,0777);
?>
```

Le nom du fichier sera composé à partir de la date du jour. Selon les paramètres d'ouverture spécifiés, si le fichier existe déjà, il sera simplement rouvert pour y ajouter les nouvelles informations, sinon, il sera créé. Ce mécanisme permet de n'avoir qu'un fichier log (trace) par jour. Les données que nous enregistrons dans le fichier seront :

- adresse IP ;
- date et heure d'affichage de la page ;
- nom de la page ;
- le navigateur ;
- la provenance.

```
<?php
$nom_fichier=DATE("Y-m-d");
$fp = fopen($destLog." ".$nom_fichier.".txt", "a");
export($_SERVER["REMOTE_ADDR"],$separateurLog);
export(DATE("Y-m-d H:i:s"),$separateurLog);
export($_SERVER['REQUEST_URI'],$separateurLog);
export($_SERVER['HTTP_USER_AGENT'],$separateurLog);
export($_SERVER['HTTP_REFERER']);
fclose($fp);
?>
```

Nous effectuons un appel à une fonction intitulée **export** qui nous permet de simplifier l'envoi des données dans le fichier. Cette fonction se présente comme ceci :

```
<?php
function export($champ,$separateur=' ')
{
    global $fp;
    fputs($fp,$champ);
    if ($separateur!='')
        fputs($fp,$separateur);
    else
```

```
    fputs ($fp, "\n");
}
?>
```

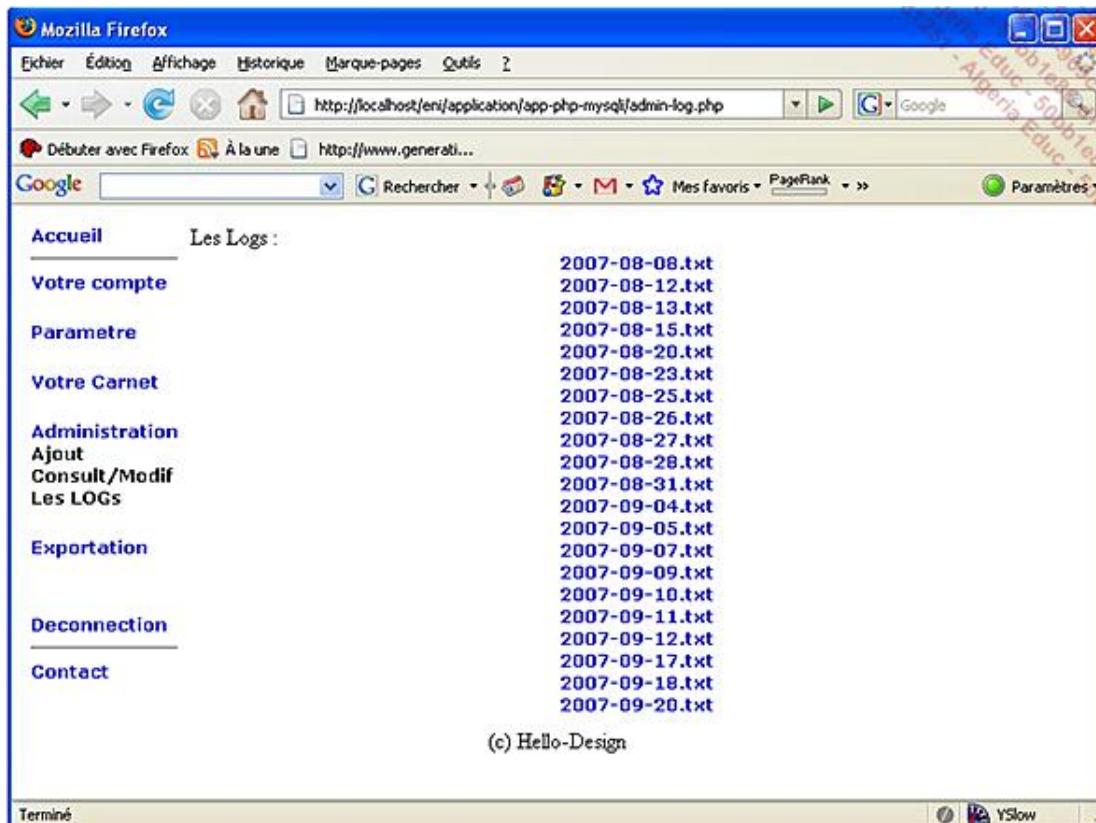
La fonction possède deux arguments. Le premier est obligatoire, par contre le deuxième peut être vide ou inexistant. Nous utilisons le caractère vide lors du dernier appel, pour passer à la ligne suivante.

## 2. Visualisation

Les logs sont accessibles pour les utilisateurs qui possèdent un niveau administrateur.

Nous allons afficher les fichiers qui se trouvent dans le dossier log.

Pour afficher les logs, nous cliquons sur la ligne du menu "Les logs" à gauche de l'écran et nous obtenons un écran comme ci-dessous :



Les fichiers qui s'affichent correspondent aux différents passages de l'ensemble des utilisateurs avec la date au format anglais (Année-mois-jour).

Pour afficher le contenu d'un dossier qui se trouve sur le serveur, nous allons utiliser les fonctions suivantes :

### opendir

Ouvre le dossier et récupère le pointeur dessus.

### readdir

Lit une entrée du dossier.

### closedir

Ferme le pointeur sur le dossier.

Pour afficher ces fichiers dans notre application :

```
<?php
echo "Les Logs : <br>" ;
```

```

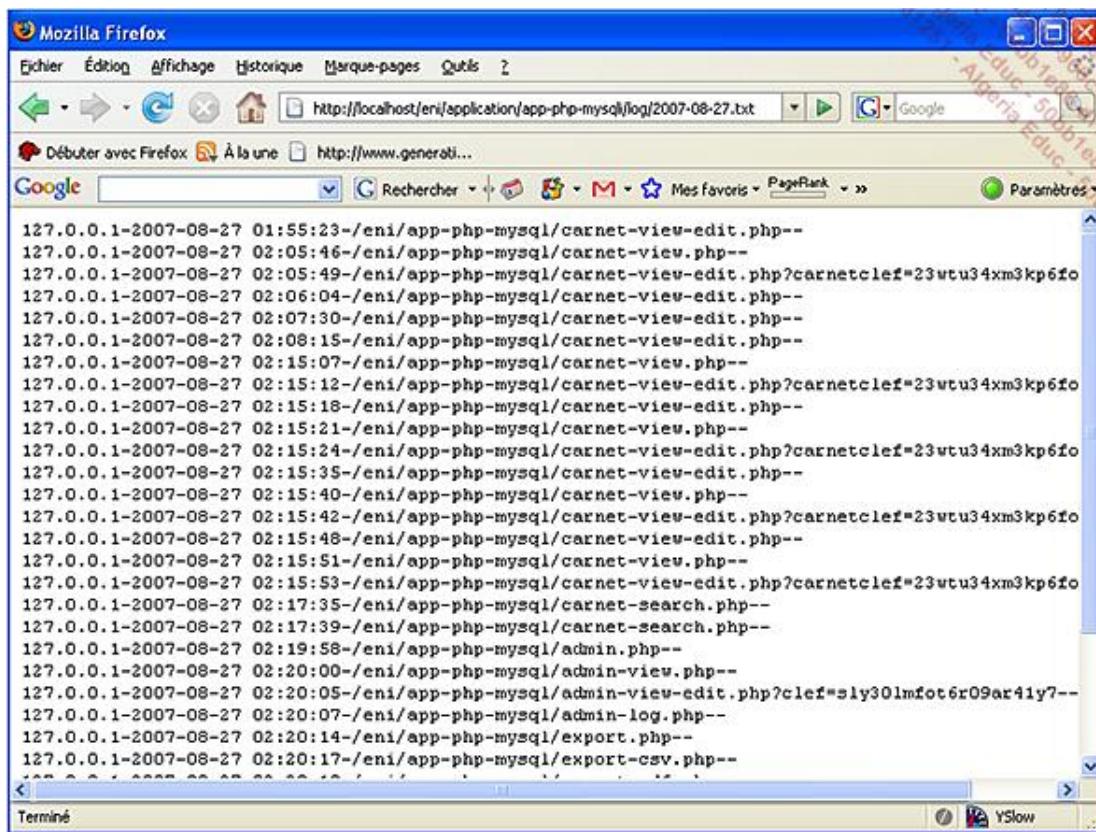
echo "<center>";
$fp = opendir ( $destLog );
while ( $fichier = readdir($fp) )
{
    if ( $fichier!= '.' && $fichier!= '...' && $fichier!= 'index.php' )
    {
        echo "<a href=$destLog$fichier class=links>".$fichier."<br>";
    }
}
closedir($fp);
echo "</center>";
?>

```

Nous n'affichons que les fichiers qui nous intéressent c'est pourquoi nous ne faisons pas apparaître le fichier index.php, ni les raccourcis qui permettent de changer de dossiers.

Sur sélection d'un des fichiers, nous obtenons les informations qui ont été mémorisées lors des différents passages des visiteurs.

Voici un exemple du résultat que nous obtenons :



Nous y voyons l'adresse IP de la machine, l'heure de passage et le chemin de navigation.

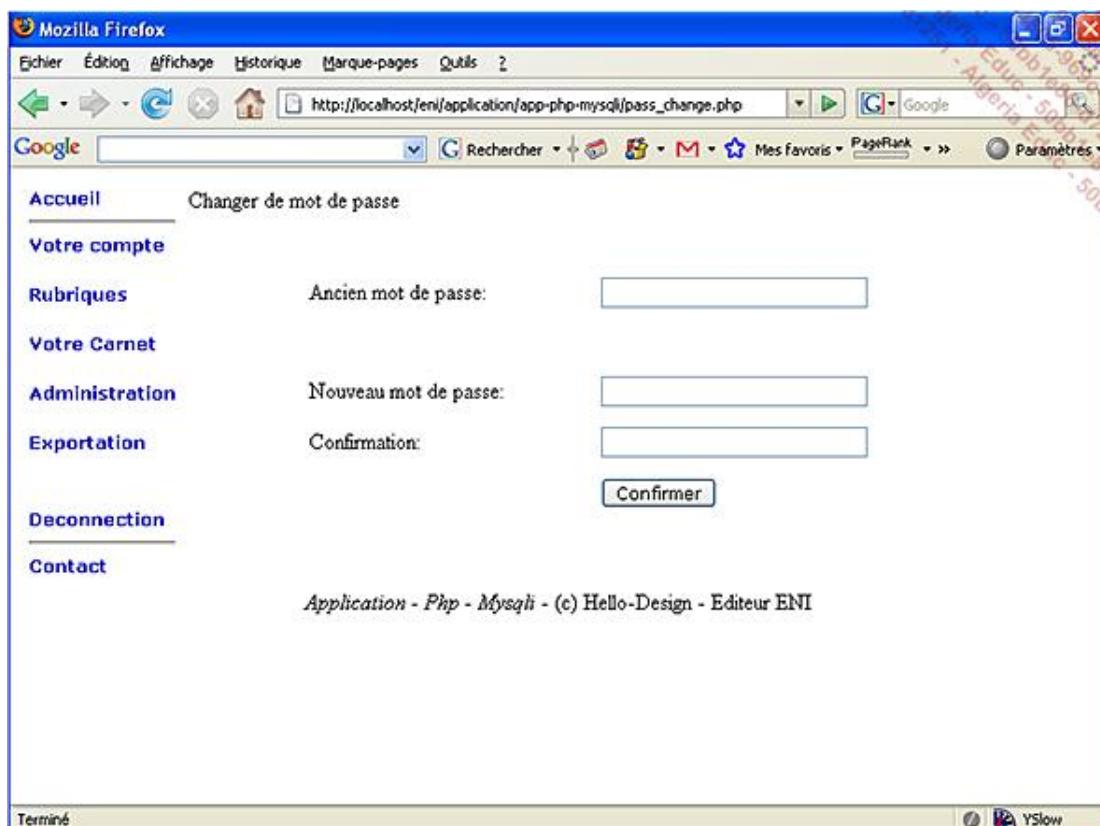
## Mots de passe

Lorsque nous sollicitons la génération automatique d'un mot de passe, notre application va en créer un avec toutes sortes de caractères.

Il est toujours possible d'apprendre ce nouveau mot de passe, mais si nous possédons de nombreux sites Internet avec des mots de passe aussi complexes, il devient difficile de se souvenir de tous.

La solution présentée consiste en ce que le possesseur du compte puisse modifier son mot de passe et créer celui de son choix. Il sera le seul à connaître son nouveau mot de passe, puisque dans la base de données celui-ci est protégé avec la fonction MD5 que nous avons déjà étudiée.

Pour cela, l'utilisateur accédera à un formulaire par l'intermédiaire du menu compte :



The screenshot shows a Mozilla Firefox browser window. The address bar displays the URL [http://localhost/en/application/app-php-mysql/pass\\_change.php](http://localhost/en/application/app-php-mysql/pass_change.php). The page content is a form titled "Changer de mot de passe". The form has the following fields:

- Rubriques**: Ancien mot de passe:
- Administration**: Nouveau mot de passe:
- Exportation**: Confirmation:

At the bottom of the form is a **Confirmer** button. On the left side of the page, there is a sidebar with links: Accueil, Votre compte, Rubriques, Votre Carnet, Administration, Exportation, Deconnection, and Contact. The footer of the page contains the text "Application - Php - Mysqli - (c) Hello-Design - Editeur ENI".

Dans ce formulaire, tous les champs sont obligatoires. Mais lorsque le visiteur clique sur **Confirmer**, nous devons effectuer un certain nombre d'opérations et de vérifications avant d'effectuer la modification.

 Cette partie est traitée avec la base de données au format MySQLi mais elle est aussi disponible avec les formats MySQL et PDO en téléchargement sur le site de l'éditeur.

Nous allons protéger les caractères spéciaux dans les trois champs pour permettre leur utilisation dans une requête SQL :

```
<?php
$oldpassword=mysqli_real_escape_string($connex,$_POST['oldpassword']);
$newpassword=mysqli_real_escape_string($connex,$_POST['newpassword']);
$newpassword2=mysqli_real_escape_string($connex,$_POST['newpassword2']);
?>
```

Concernant la mémorisation des mots, nous allons utiliser une autre approche que celle que nous avons déjà vue dans les chapitres précédents. Étant donné qu'il s'agit d'effectuer la mise à jour d'un seul champ, nous pouvons réaliser un contrôle global.

Nous allons préparer les messages au cas où un ou plusieurs champs ne seraient pas renseignés et nous en profitons pour comparer les deux champs de saisie du nouveau mot de passe.

```
<?php
$msg=" ";
```

```

if (empty($oldpassword)) $msg .= "Vous n'avez pas entré votre ancien mot
de passe.<br>";
if (empty($newpassword)) $msg .= "Vous n'avez pas entré de nouveau mot
de passe.<br>";
if (empty($newpassword2)) $msg .= "Vous n'avez pas confirmé le nouveau mot
de passe.<br>";
if ($newpassword!=$newpassword2) $msg .= "Vos nouveaux mot
de passe sont différents.<br>";
?>

```

Si la comparaison des champs est correcte et que le formulaire ne possède aucun champ vide, nous pouvons passer à l'étape suivante.

Cette étape consiste à récupérer le mot de passe actuellement stocké dans la base de données, et que nous ne connaissons pas. Pour obtenir ce mot de passe, nous devons préparer une requête avec comme critère la clef de session que nous possédons en mémoire.

```

<?php
if ($msg == " ")
{
$sql="select * from user where idclef='".$_SESSION['idclef']."' ";
assert ('mysqli_query($connex, $sql)');
$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
$row=mysqli_fetch_object( $qid);
$memoPass=$row->password;
?>

```

Nous devons obligatoirement contrôler si la personne a saisi le même mot de passe que celui que nous possédons dans la base de données. Pour effectuer ce test, nous traitons le mot saisi dans le champ Ancien mot de passe avec la fonction **MD5** et effectuons une comparaison avec la valeur extraite de la base de données. S'il est différent, nous stoppons les contrôles et signalons où se situe le problème par l'intermédiaire d'un message.

```

<?php
if ($memoPass != md5($oldpassword))
{
    $msg="Votre ancien mot de passe n'est pas valide.<br>";
}
?>

```

Si ce test s'est correctement déroulé, nous pouvons protéger le nouveau mot de passe et mettre à jour la base de données.

```

<?php
else
{
    $newpassword=md5($newpassword);
    $newpassword=mysqli_real_escape_string($connex,$newpassword);
    $sql="update user set password='".$newpassword' where idclef='".$_SESSION['idclef'].'
    ";
    assert ('mysqli_query($connex, $sql)');
    $qid=mysqli_query($connex, $sql);
    if (!$qid) die ("Probleme : " . mysqli_error($connex));
?>

```

La dernière étape s'est bien passée, donc nous affichons un message de succès, nous libérons et fermons la connexion.

```

<?php
else
{
    require_once "head.inc.php";
    echo "Votre nouveau mot de passe a été pris en compte.";
    include("footer.inc.php");
    mysqli_free_result($qid);
    mysqli_close($connex);

    exit;
}

```

```
 }  
?>
```

Comme nous pouvons le constater dans le code source, chaque étape peut générer un ou plusieurs messages d'alertes.

Ces messages vont s'afficher en haut de l'écran. Cependant, il est tout à fait possible de les afficher à l'endroit qui vous convient le mieux.

```
<html>  
<body>  
<p>  
<?php echo $msg; ?>  
</p>  
</body>  
</html>
```

# Captcha

Nous allons ajouter dans notre formulaire d'identification un filtre supplémentaire, appelé captcha.

## 1. Conception

Le captcha représente une image avec un code, que nous allons générer dynamiquement. Ce code permet de vérifier si notre visiteur est bien un internaute et non un robot, pour éviter de recevoir des messages non souhaités.

Ce captcha peut être placé dans n'importe quel écran de saisie puisqu'il est généré à la volée.

Nous allons mettre en pratique l'utilisation des fonctions étudiées dans le chapitre précédent.

Ce chapitre est commun à l'ensemble des différentes applications de l'ouvrage car il n'y a aucune implication touchant les bases de données. Les différentes opérations sont effectuées du côté du navigateur.

### a. Déclaration de l'image

Dans un premier temps, il est nécessaire de déclarer la session, même si la fonction a déjà été utilisée lors de l'ouverture d'une page Internet. Nous réalisons cette opération pour être certain de ne pas perdre les données.

Ensuite, nous devons déclarer les variables que nous aurons besoin d'afficher.

Fichier include/fct\_captcha.inc.php

```
<?php
session_start();
-----
$largCaptcha=130; // largeur captcha
$hautCaptcha=65; // hauteur captcha
$longChaineCaptcha=6; // nombre de caractères pour le captcha
$fontTTF="Swiss721 BT.TTF"; // police de caractères pour le captcha
-----
?>
```

Les lignes de programmation ci-dessus montrent que nous devons préparer certaines données comme la largeur et la hauteur du captcha.

Par ailleurs, il est important de définir le nombre de caractères qui seront affichés (ici nous afficherons six caractères) et une police de caractères.

Ensuite, nous devons aussi générer une chaîne de caractères. Il existe différentes méthodes permettant de générer une chaîne de caractères. La méthode choisie est celle de l'horloge, en utilisant les microsecondes pour éviter d'avoir deux textes identiques.

```
<?php
function chaine_captcha($longueur)
{
    $md5 = md5(microtime() * mktime());
    $chaine = substr($md5,0,$longueur);
    return $chaine;
}

$_SESSION['textCaptcha']=chaine_captcha($longChaineCaptcha);
?>
```

Nous en profitons pour protéger cette chaîne en utilisant la fonction MD5 que nous découpons, car pour rappel, la fonction MD5 génère une longueur de 32 caractères alors que dans notre exemple, nous utilisons six caractères.

Le résultat obtenu sera stocké dans une session, ainsi nous possédonns la valeur qui sera affichée dans notre image et nous utiliserons cette valeur lorsque nous effectuerons le test de comparaison par rapport à la saisie.

Nous vous conseillons de lire la partie détaillée un peu plus loin.

### b. Paramètres de l'image

Maintenant, nous commençons à générer notre captcha. Nous déclarons les dimensions de l'image avec les valeurs définies un peu plus haut.

```
<?php
header('Content-type: image/png');
$img = imagecreatetruecolor($largCaptcha,$hautCaptcha);
?>
```

Nous plaçons un fond de couleur blanc en utilisant un rectangle blanc.

```
<?php
$fondCol = imagecolorallocate($img, 255,255,255);
imagefilledrectangle($img,0,0,$largCaptcha,$hautCaptcha,$fondCol);
?>
```

La couleur blanche pour le fond permet de rester dans la couleur du fond de l'application.

En plus de la couleur de fond, nous créons un effet de lignes comprenant deux types de lignes aléatoires, générées par l'intermédiaire d'une boucle. Ces lignes se positionnent dans l'image par rapport à la dimension de celle-ci.

Nous utilisons la fonction suivante :

### Rand()

Génère une valeur aléatoire.

```
<?php
$position=0;
for($ligne=-30;$ligne<$largCaptcha+30;$ligne+=10)
{
    $color = imagecolorallocate($img, rand(80,250), rand(18,250), rand(100,250));
    if ($position==0)
    {
        imageline($img,$ligne,rand(1,10),$largCaptcha+rand(20,60),
$hautCaptcha, $color);
        imageline($img,$largCaptcha-$ligne,5,rand($hautCaptcha-10,
$hautCaptcha+10),$largCaptcha, $color);
        $position=1;
    }
    else
    {
        imageline($img,$ligne,5,rand(10,30),$hautCaptcha, $color);
        imageline($img,$largCaptcha-$ligne,rand(5,20),$hautCaptcha+rand
($hautCaptcha-10,$hautCaptcha+10),$largCaptcha, $color);
        $position=0;
    }
}
?>
```

Comme nous le voyons chaque ligne possède une couleur différente. À chaque passage, nous sélectionnons l'un ou l'autre type de lignes.

### c. Le texte

Le texte sera composé de six caractères et proviendra de la fonction utilisée dans les pages précédentes.

Cependant si nous affichons le texte sans appliquer d'effets dessus, les robots pourront reconnaître le texte et notre filtre deviendra dépassé.

Un effet sera appliqué sur chaque caractère aléatoirement comme ceci :

```
<?php
$x=rand(4,35);
for($c=0;$c<$longChaineCaptcha;$c++)
{
    $angle=mt_rand(10, 50);
    $size=mt_rand(14, 36);
    $text=$_SESSION['textCaptcha'][$c];
```

```

$y=30+rand(1,40);
$color=imagecolorallocate($img,
rand(10,99),rand(10,99),rand(10,99));
imagettftext($img, $size, $angle, $x+18*$c, $y, $color, $fontTTF,
$text);
}
?>

```

Dans le code source ci-dessus, nous réalisons une boucle permettant de découper caractère par caractère. Nous définissons une taille variable du texte avec un angle.

Nous imposons une position de départ pour être certain de commencer dans l'image et une position en hauteur (en Y) aléatoirement pour gêner un peu plus les robots.

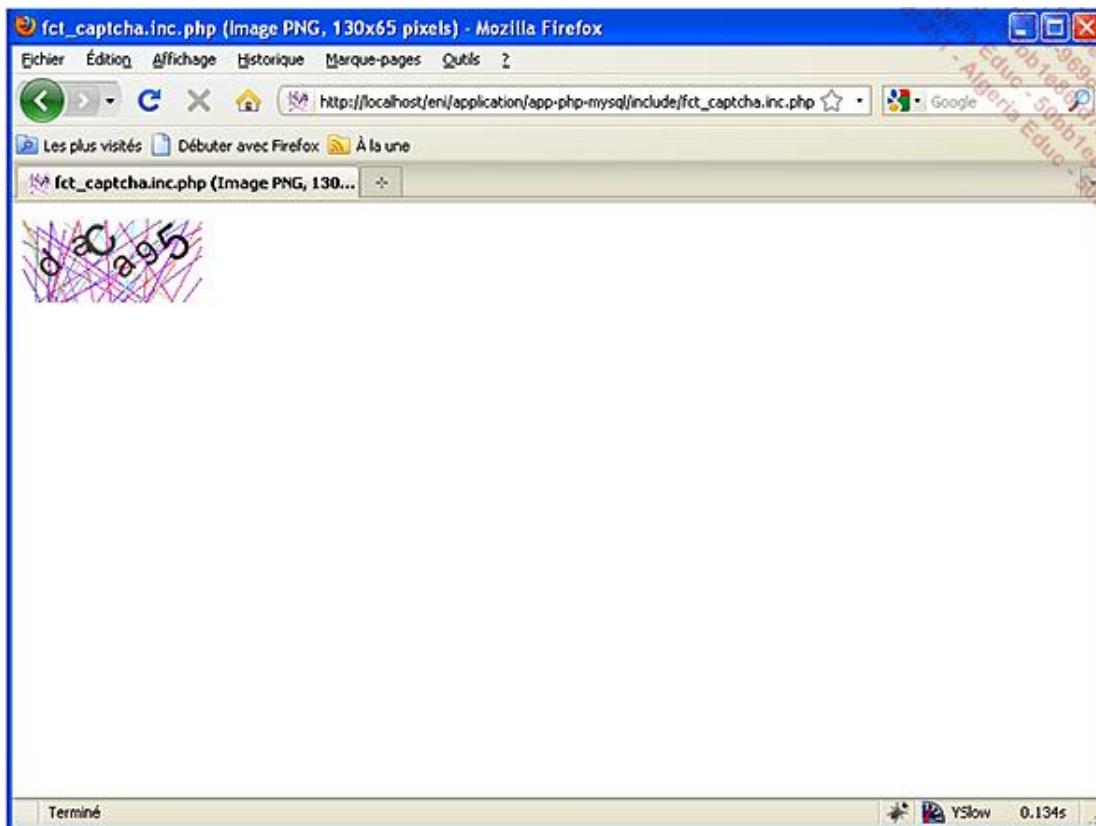
Nous affichons l'image que nous libérons ensuite :

```

<?php
imagepng($img);
imagedestroy($img);
?>

```

Lorsque nous effectuons un test d'affichage, nous obtenons le résultat suivant :



Maintenant que notre routine fonctionne, nous pouvons insérer ce captcha dans notre formulaire.

## 2. Le formulaire

Ajoutons le captcha dans notre formulaire de saisie de la façon suivante :

Fichier carnet-add.php

```


<input type='text' name='textCaptcha' size='10'> *
<?php
if (isset($erreurs['textCaptcha']))
echo $message_erreur['textCaptcha']
?>

```

Dans notre formulaire de saisie, nous ajoutons une ligne supplémentaire. Cette ligne affiche :

- notre image, en faisant appel au fichier externe que nous créons à la volée,
- une zone de saisie pour que l'internaute puisse saisir le code.

Une petite condition apparaît sous la forme d'une étoile, qui informe l'internaute que ce champ est obligatoire.

Lors de la validation de notre formulaire, il effectue un test sur notre captcha, qui se décompose comme ceci :

```
<?php
if ($_SESSION['textCaptcha'] != $frm['textCaptcha'])
{
    $erreurs['textCaptcha'] = true;
    $msg['textCaptcha'] = "Erreur de saisie";
}
?>
```

Nous récupérons le texte mémorisé dans notre session car cette valeur a été stockée lors de la conception de notre captcha. Nous effectuons une comparaison avec la valeur saisie.

Si une erreur de saisie apparaît, un message est affiché, et donc le formulaire ne sera pas enregistré.

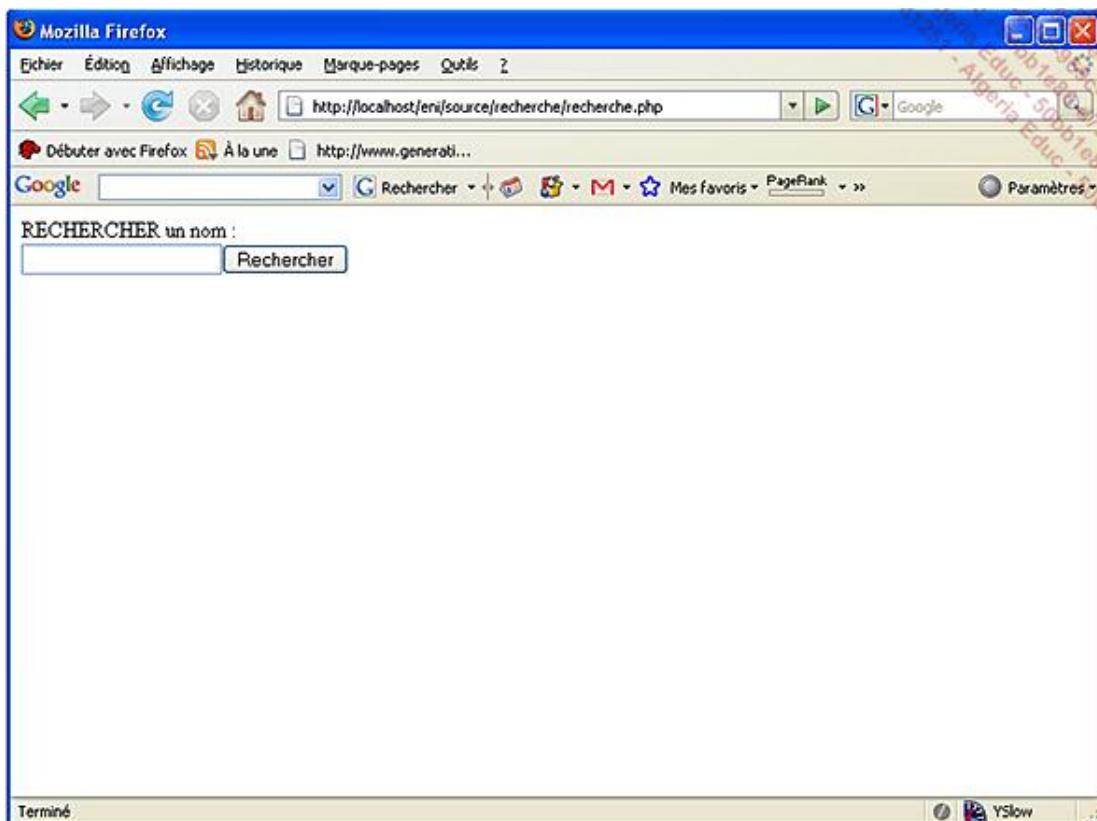
Si l'opération s'est correctement déroulée, l'application continue la procédure de test.

# Recherche

## 1. Technique de recherche

Dans n'importe quelle application Internet, il existe une fonctionnalité à laquelle nous ne pouvons pas échapper, il s'agit de la recherche. Cette fonction permet d'effectuer une recherche basée sur une partie d'un mot ou un mot complet dans un champ de la table.

Pour effectuer cette recherche, nous proposons un formulaire de saisie :



L'exemple sera réalisé en MySQL pour faciliter la découverte de cette fonction.

Lorsque l'utilisateur confirme la saisie par le bouton **Rechercher**, nous allons pour commencer recharger le formulaire de saisie. Ainsi, le visiteur pourra effectuer une nouvelle recherche si celle en cours ne lui convient pas.

Nous allons tester si le formulaire contient bien une valeur, sinon nous n'irons pas plus loin :

```
recherche/recherche.php
```

```
<?php
if (!empty($_POST['action'])) $action=$_POST['action'];
if (!empty($_POST['mots'])) $mots=$_POST['mots']; else $mots="";
if ($mots=="") exit;
?>
```

Pour effectuer la recherche dans une table, nous utilisons l'opérateur standard **LIKE** avec le symbole % permettant la recherche au début, au milieu et à la fin du champ.

L'avantage est que si l'utilisateur ne saisit que quelques caractères, nous pourrons lui fournir un résultat plus complet, comme le mot en entier.

```
<?php
$sql="SELECT nom, prenom FROM exemple WHERE nom LIKE '%" . $mots . "%'";
$qid = mysql_query($sql);
if (!$qid) die ("Probleme : " . mysql_error());
```

Si la chaîne recherchée n'est pas présente dans le champ, nous signalons que nous n'avons pas trouvé de résultat.

```
<?php
if (mysql_num_rows($qid) == 0)
{
echo "Nous n'avons pas trouvé de résultats";
} else {
?>
```

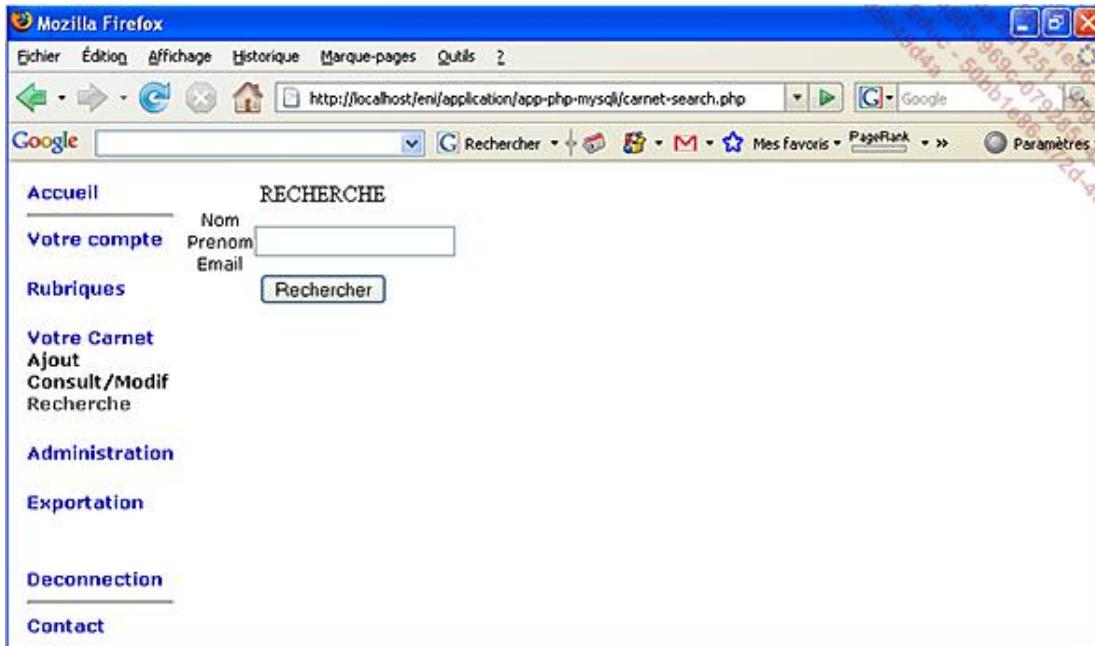
Sinon nous pouvons afficher le résultat à l'aide d'une boucle, car il peut y avoir plusieurs correspondances.

```
<?php
while($recherche=mysql_fetch_object( $qid ) )
{
echo "$recherche->nom ";
echo "$recherche->prenom";
echo "<br>";
}
?>
```

Dans notre application, pour élargir les possibilités de recherche, nous allons autoriser la recherche sur plusieurs champs.

 Nous allons étudier cette fonction avec une base de données MySQLi. Cette partie est aussi disponible avec les formats MySQL et PDO en téléchargement sur le site de l'éditeur.

Nous allons réaliser un formulaire, composé comme ci-après :



L'utilisateur pourra effectuer une recherche sur plusieurs champs à partir d'une même zone de saisie. Ici, nous allons permettre d'effectuer une recherche sur les champs Nom, Prénom, et E-mail.

Avec la particularité de pouvoir effectuer la recherche dans tout le champ, c'est-à-dire que le mot recherché peut se situer au début, au milieu ou à la fin du champ.

Par exemple, si nous désirons effectuer une recherche avec deux lettres comme **EN**, nous pouvons obtenir dans le résultat les mots Editions ENI car ils peuvent se trouver dans au moins un des trois champs que nous avons retenus pour effectuer la recherche.

Dans l'application, nous allons effectuer la recherche dans le carnet d'adresses du titulaire d'un compte. Pour cela nous devons valider la saisie, ce qui va nous permettre de recharger la page pour effectuer la recherche des données dans la base de données.

Nous allons commencer par vérifier le contenu du champ utilisé dans le formulaire. Si aucun mot n'existe, nous n'irons

pas plus loin.

carnet-search.php

```
<?php
$action=verif_GetPost($_POST['action']);
$mots=verif_GetPost($_POST['mots']);
if ($mots=="") exit;
?>
```

Nous convertissons les caractères au format HTML afin de préparer la requête.

```
<?php
$mots=htmlentities($_POST['mots'], ENT_QUOTES, 'UTF-8');
?>
```

Pour préparer la requête, nous devons prendre en compte plusieurs critères importants :

- la recherche dans plusieurs champs en même temps ;
- retenir le carnet qui correspond à la bonne personne.

Pour cela, nous allons regrouper le premier critère entre parenthèses. Ainsi, nous pouvons effectuer la recherche sur plusieurs champs.

L'autre critère, c'est-à-dire le iduser (identification du compte), sera inséré en dehors des parenthèses.

Nous en profitons pour effectuer un tri alphabétique pour améliorer l'affichage du résultat.

```
<?php
$sql="SELECT carnetclef,nom,prenom,e-mail,tel,portable
      FROM carnet
      WHERE (
      nom LIKE '%" . $mots . "%'
      OR prenom LIKE '%" . $mots . "%'
      OR e-mail LIKE '%" . $mots . "%'
      )
      AND iduser='".$_SESSION['iduser']."'"
      ORDER BY nom,prenom ";

$qid = mysqli_query($connex, $sql);
if (!$qid) die ("Probleme : " . mysqli_error($connex));
?>
```

Si nous obtenons des résultats, nous allons afficher les informations les plus demandées et qui nous intéressent : le nom, le prénom, le téléphone, l'e-mail. Bien sûr, nous donnons la possibilité d'accéder à la fiche complète de cette personne pour consulter toutes les informations la concernant.

```
<?php
if (mysqli_num_rows($qid) == 0)
{
echo "Nous n'avons pas trouvé de résultats";
} else {
?>


|     |        |       |     |          |         |
|-----|--------|-------|-----|----------|---------|
| Nom | Prenom | Email | Tel | Portable | Edition |
|-----|--------|-------|-----|----------|---------|


```

```
echo "<td>$recherche->prenom</td>" ;
echo "<td>$recherche->e-mail</td>" ;
echo "<td>$recherche->tel</td>" ;
echo "<td>$recherche->portable</td>" ;
echo "<td><a href=carnet-view-edit.php?carnetclef=$recherche->carnetclef
class=links>Cliquez ici</a></td>" ;
echo "</tr>" ;
}
?>
```

Nous libérons le résultat et fermons la connexion.

```
<?php
mysqli_free_result($qid);
mysqli_close($connex);
?>
</table>
<?php } ?>
```

## 2. Multipages

Le résultat peut contenir beaucoup de lignes. Pour faciliter la lecture du résultat, il peut être nécessaire de proposer son affichage sur plusieurs pages.

Nous pouvons réutiliser ce que nous avons étudié dans le chapitre précédent car il s'agit du même principe de gestion.

# Exportation CSV

## 1. Création

Le format CSV correspond à un fichier texte structuré. Cette structure permet de définir une interprétation en colonnes qui est utilisée par certains logiciels comme les tableurs.

Le symbole par défaut qui est utilisé pour la séparation des colonnes est le point-virgule ";". Il est tout à fait possible de définir un autre caractère de son choix mais avec un risque d'incompatibilité.

Nous allons exporter la totalité de la liste du carnet d'adresses pour l'enregistrer sur le disque dur au format CSV.

Pour préparer l'exportation, nous devons vérifier si le dossier de sauvegarde existe bien. Si ce n'est pas le cas, nous allons le créer et lui donner les droits en écriture.

Ce dossier de réception s'appellera "dl" qui correspond au mot download (téléchargement) et sera représenté par la variable \$destDL.

```
< ?php
if (!is_dir($destDL))
{
    if (!@mkdir($destDL))
    {
        return array(false,'Erreur lors de la création du dossier $destDir');
    }
}
@chmod($destDL,0777);
?>
```

Nous vérifions si le fichier a déjà été créé ou s'il est encore présent dans le dossier. Si le fichier existe, nous le supprimons avec la fonction suivante :

### unlink

Efface un fichier.

```
< ?php
if(file_exists($destDL."".$_SESSION['login'].".csv"))
{
unlink($destDL."".$_session['login'].".csv");
}
?>
```

Nous préparons la requête pour récupérer les informations qui sont mémorisées dans la base de données et nous l'exécutons.

```
< ?php
$sql="select carnet.*,carnet.id as idlinks,user.id,user.idclef
      FROM carnet,user
      WHERE user.id=carnet.iduser
      AND carnet.iduser='".$SESSION['iduser']."'"
      AND user.idclef='".$SESSION['idclef']."' ";
assert ('mysqli_query($connex, $sql)');
$qid=mysqli_query($connex, $sql);
if (!$qid) echo ('Requête invalide : ' . mysqli_error($connex));
?>
```

Passons maintenant à la création du fichier. Nous ouvrons le fichier qui nous servira pour la sauvegarde à partir du nom de l'utilisateur du compte.

Chaque information que nous allons écrire est séparée par le symbole de délimitation, c'est-à-dire le signe ";".

Ce séparateur se trouve dans le fichier include/config.inc.php.

Nous allons inscrire certains champs :

Nom

Prénom

Adresse

Code Postal

Ville

Email

Téléphone

Numéro du portable

La première ligne que nous écrivons correspond aux en-têtes de colonnes nécessaires pour respecter ce format.

```
< ?php
$fp = fopen($destDL."".$_SESSION['login'].".csv", "a");
export("Nom",$separateur);
export("Prenom",$separateur);
export("Adresse",$separateur);
export("Code Postal",$separateur);
export("Ville",$separateur);
export("e-mail",$separateur);
export("Tel",$separateur);
export("Portable");

$i=1;
while ($list=mysqli_fetch_object($qid))
{
export($list->nom,$separateur);
export($list->prenom,$separateur);
export($list->adresse,$separateur);
export($list->codepostal,$separateur);
export($list->ville,$separateur);
export($list->e-mail,$separateur);
export($list->tel,$separateur);
export($list->portable);

}
?>
```

Lorsque nous avons terminé l'écriture du fichier, nous le fermons.

```
< ?php
fclose($fp);
?>
```

## 2. Téléchargement

Nous allons récupérer le fichier que nous venons de créer sous la forme d'un fichier à télécharger.

Pour réaliser cette manipulation, nous allons utiliser un lien classique HTML. Il pointe sur un fichier possédant le rajout d'extension INC que nous avons vu au début de l'ouvrage.

```
<a href="download.inc.php class=links>Cliquer ici</a>
```

Pour effectuer le téléchargement, nous n'allons envoyer aucun paramètre, juste proposer la sauvegarde du fichier. Pour créer ce fichier, nous allons utiliser les valeurs que nous possédons dans la session.

```
<?php
session_start();
$file=$_SESSION['login'].".csv";
?>
```

Avant de permettre le téléchargement du fichier qui sera proposé, nous allons effectuer quelques petites vérifications

portant sur le nom du fichier. Si un problème apparaît, nous renvoyons le visiteur à la page index du site pour qu'il s'identifie à nouveau.

```
<?php
if (!isset($_SESSION['dernier_passage']))  )
{
    session_regenerate_id();
}

if (!isset($_SESSION['login']))  )
{
    header("Location:index.php");
}
?>
```

Nous utilisons les lignes de programme suivantes pour proposer le téléchargement :

```
<?php
header('Content-type: application/csv');
header('Content-Disposition: attachment; filename='.basename($file));
header('Accept-Ranges: bytes');
header('Content-Length: '.filesize($file) );
readfile($file);
?>
```

Lorsque l'utilisateur aura téléchargé son fichier, nous pouvons le supprimer car il est inutile de le garder chez nous.

```
<?php
unlink($file);
?>
```

# Exportation PDF

## 1. Présentation

Le format PDF existe depuis de nombreuses années et est devenu une des possibilités d'exportation standards dans le langage PHP. Le format PDF peut être généré à la volée, c'est-à-dire en réalisant un document sécurisé grâce à la classe FPDF. Le F avant le mot PDF correspond à FREE, donc vous êtes libre d'utiliser et de modifier le document si vous le souhaitez.

Le site Internet officiel FPDF propose beaucoup de fonctions permettant de réaliser un document PDF.

Avant de commencer, il faut télécharger la bibliothèque FPDF disponible sur le site suivant : <http://www.fpdf.org>.

Suite au téléchargement, nous devons décompresser le fichier ZIP. Nous garderons un fichier **fpdf.php** et un dossier **font** qui sont nécessaires pour la création correcte d'un document PDF.

## 2. Principes de base

Commençons par un exemple simple : nous allons inscrire dans un fichier PDF, le message **Editions ENI**, centré sur une ligne et avec une police de caractères (appelée aussi typo) en Arial avec une taille de 16.

Pour réaliser ceci il existe des fonctions prédéfinies. Ces fonctions sont accessibles sous la forme d'une aide en ligne et donc directement sur le site Internet de l'auteur de la bibliothèque FPDF.

Les fonctions que nous avons utilisées sont :

### **define**

Définit une constante.

### **FPDF**

Permet de définir la dimension d'une feuille PDF, par défaut il s'agit du format A4.

### **AddPage**

Ajoute une nouvelle page.

### **Ln**

Saut de ligne. Cela correspond à la hauteur d'un retour chariot (retour à la ligne) car au lieu d'effectuer plusieurs retours à la ligne, nous pouvons effectuer cette opération en une seule fois en spécifiant une valeur correspondant à une hauteur.

### **SetFont**

Fixe la police de caractères.

### **Cell**

Affiche une cellule sous la forme d'un rectangle et d'une dimension précise.

### **Output**

Envoie le document à l'écran ou enregistre le document sur le disque dur.

Nous devons obligatoirement inclure le fichier **fpdf.php** et définir le répertoire des polices de caractères.

```
pdf/exemple.php
```

```
<?php
define('FPDF_FONTPATH','font/');
require('fpdf.php');
```

?>

Nous créons le document au format PDF en instanciant la classe FPDF :

```
<?php  
$pdf=new FPDF();  
?>
```

Nous créons une page vide :

```
<?php  
$pdf->AddPage();  
?>
```

Nous laissons un espace vide entre le haut de la feuille et le texte, pour afficher le texte EDITIONS ENI.

Nous allons faire un retour chariot, puis définir un style de police de caractère c'est-à-dire ici, une police Times en gras de taille 16 et ensuite, afficher le titre qui sera centré sur toute la largeur de la feuille.

```
<?php  
$pdf->Ln(30);  
$pdf->SetFont('Times','B',16);  
$pdf->Cell(0,5,'EDITIONS ENI',0,1,'C');  
?>
```

Nous affichons le résultat à l'écran :

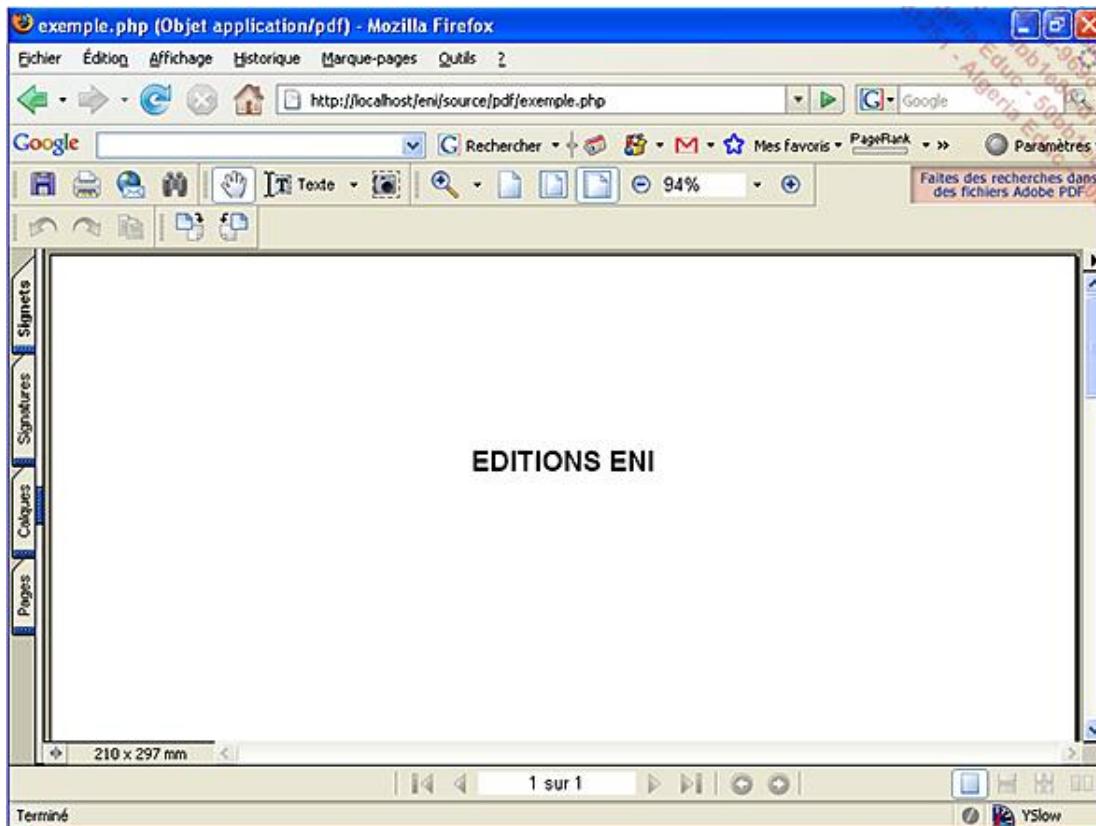
```
<?php  
$pdf->Output();  
?>
```

Il est possible de forcer la sauvegarde du document sous forme de fichier sur le disque dur en modifiant la dernière ligne comme ceci :

```
<?php  
$pdf->Output("c:\\exemple.pdf");  
?>
```

Ainsi le fichier sera sauvegardé sur le disque dur C de l'utilisateur (=possesseur du compte).

Nous obtenons le résultat suivant :



### 3. Notre application

Dans notre application, à partir du carnet d'adresses nous allons choisir un contact et au lieu d'afficher sa fiche à l'écran, nous allons l'éditer au format PDF et ainsi obtenir un document avec une présentation personnalisée.

 L'exportation dans ce format va être réalisée avec la base de données MySQL. Bien sûr, cette partie est disponible avec les connexions MySQLi et PDO en téléchargement sur le site de l'éditeur.

Nous allons afficher le contenu du carnet d'adresses, avec certaines colonnes importantes :

- Nom
- Prenom
- Email
- Téléphone
- Portable
- Photo
- Etc.

Lorsque l'utilisateur choisit une des entrées proposées, nous allons générer le fichier PDF.

Voici le résultat à obtenir :

**FICHE**

Nom : VILLENEUVE	
Prénom : Christophe	
Adresse : adr	
99999 ville	
Tel : Portable : Email : livre@hello-design.fr	

Rubrique	Observations
site internet	www.hello-design.fr
Editeur	www.eni.fr

Pour cela, nous allons effectuer certaines déclarations. Dans l'exemple ci-dessous, nous utilisons en plus les fonctions suivantes :

#### **AliasNbPages**

Définit un alias pour le nombre de pages total.

#### **Rect**

Trace un rectangle à l'endroit désiré.

#### **Multicell**

Affiche un texte sur plusieurs lignes avec retours de chariot.

#### **Image**

Affiche une image.

#### **SetY**

Fixe la position courante en Y (en hauteur).

Avant d'afficher la page PDF, nous devons penser à insérer un en-tête et un pied de page. Ces deux éléments ne sont pas obligatoires, mais ces informations sont ainsi répétées sur toutes les pages si le document PDF en comporte plusieurs.

Dans l'en-tête, nous allons sauter une ligne pour ne pas écrire en dehors de la feuille, comme ceci :

```
<?php
function Header()
{

```

```

    $this->Ln(1);
}
?>

```

Pour le bas de page, nous allons insérer le numéro de la page si le document en possède plusieurs, comme ceci :

```

<?php
function Footer()
{
    $this->SetY(-15);
    $this->SetFont('Times','I',8);
    $this->Cell(0,10,'Page '.$this->PageNo().'/{nb}',0,0,'C');
}
?>

```

Nous nous positionnons en bas de la page et remontons d'une hauteur (-15) pour déterminer l'emplacement du bas de page. Nous pouvons alors afficher le numéro de la page centré par rapport à la largeur totale du document.

Pour réaliser à la volée le document en PDF nous devons déclarer un nouveau fichier PDF qui sera au format A4. Nous lui signalons la possibilité d'avoir plusieurs pages et demandons qu'il prépare une page vierge.

```

<?php
$pdf=new PDF();
$pdf->AliasNbPages();
$pdf->AddPage();
?>

```

Ensuite, nous allons nous connecter à la base de données MySQL et compter le nombre de lignes récupérées. Étant donné que nous avons sélectionné une entrée et que nous avons envoyé la clef du carnet qui correspond à celle-ci, la requête doit nous retourner une ligne.

```

<?php
$sql="SELECT * FROM carnet WHERE carnetclef='".$carnetclef' ";
$qid=mysql_query($sql);
if( ! $qid ) die("Probleme : " . mysql_error());
$nligne= mysql_num_rows($qid);
$row=mysql_fetch_object( $qid );
?>

```

Nous affichons le titre de la feuille, ici se sera **Fiche**, centré sur la largeur de la page :

```

<?php
$pdf->SetFont('Times','B',30);
$pdf->Cell(0,5,'FICHE',0,1,'C');
?>

```

Nous pouvons afficher les coordonnées de la personne, encadrées par un rectangle :

```

<?php
$pdf->Ln(10);
$pdf->SetFont('Times','B',12);
$pdf->Cell(0,5,'Nom : '.$row->nom);
$pdf->Ln(5);
$pdf->Cell(0,5,'Prenom : '.$row->prenom);
$pdf->Ln(5);
$pdf->Cell(0,5,'Adresse : '.$row->adresse1);
$pdf->Ln(5);
$pdf->Cell(20);
$pdf->Cell(0,5,$row->adresse2);
$pdf->Ln(5);
$pdf->Cell(20);
$pdf->Cell(0,5,$row->codepostal." ".$row->ville);
$pdf->Ln(5);
$pdf->Rect(10,22,100,35);
?>

```

Nous affichons dans un autre cadre le numéro de téléphone, le numéro de portable et l'e-mail :

```

<?php
$pdf->Ln(10);

```

```

$pdf->MultiCell(100,5,
    "Tel : ".$row->tel."\n\rPortable : ".$row->portable."\n\rEmail :
    ".$row->e-mail."\n\r",1,1);
?>

```

Si la fiche possède une photo, nous en profitons pour l'afficher dans le document.

```

<?php
if ($row->photo)
{
    $pdf->Image($destDir.$row->photo,120,20);
}

```

Nous allons maintenant afficher les informations que nous possédons sur les rubriques. Nous préparons la même requête que celle qui nous permet d'effectuer les modifications dans une fiche du carnet d'adresses. Nous allons utiliser une fonction appelée **tableau** que nous étudierons ensuite.

```

<?php
$sqlDetails="SELECT carnet_details.*,rubrique.id,rubrique.nom
    FROM carnet_details, rubrique
    WHERE carnet_details.idrubrique=rubrique.id
        AND idcarnet='".$row->id."' ";
$qidDetails=mysql_query($sqlDetails);
if (!$qidDetails) die("Probleme : " . mysql_error());

$pdf->Ln(20);
$pdf->Cell(20);
$pdf->tableau("Rubrique","Observations");
while ($rowDetails=mysql_fetch_object($qidDetails))
{
    $pdf->tableau($rowDetails->nom,stripslashes($rowDetails->observation));
}
$pdf->Ln(5);
?>

```

Nous libérons les connexions et affichons le résultat à l'écran :

```

<?php
mysql_free_result($qidDetails);
mysql_free_result($qid);

mysql_close();
$pdf->Output();
?>

```

La fonction **tableau** correspond à une certaine présentation prédéfinie. En effet, au lieu de répéter à plusieurs reprises les instructions avec différentes positions, il est préférable d'utiliser une fonction pour optimiser le traitement.

```

<?php
function tableau($col,$col2)
{
    $this->Cell(1,5,$col);
    $this->Cell(50);
    $this->Cell(0,5,$col2);
    $this->Ln(10);
}
?>

```

Lorsque nous construisons le squelette pour créer le fichier PDF, il est préférable (même si ce n'est pas obligatoire) d'insérer les fonctions dans la classe de FPDF. C'est pourquoi, nous utilisons **\$this** qui fait appel à la classe proprement dit.

Pour l'utiliser, nous lui envoyons deux valeurs :

- une valeur se positionne au début de la colonne,
- l'autre valeur se positionne avec un décalage de 50 pixels.

Ensuite nous effectuons un retour à la ligne pour permettre d'afficher les autres informations.

Lorsque nous affichons le résultat à l'écran, le visiteur peut effectuer la sauvegarde directement sur son disque dur puisqu'il se trouve devant un document PDF classique mais qui a été réalisé à la volée grâce au PHP.

# Exportation XML

## 1. Structure

Le format XML (*eXtended Markup Language*) est un langage utilisant des balises. Il est différent du format HTML car il s'agit d'un langage structurant. Nous pouvons lire ce format en mode texte avec un éditeur de texte classique.

Il est possible de créer un fichier XML grâce au PHP, ainsi nous pouvons proposer différentes fonctionnalités :

- être lu par un autre langage,
- créer un flux RSS (*Real Simple Syndication*),
- correspondre avec d'autres applications.

La structure d'un fichier XML est stricte, il est juste nécessaire de respecter les noms des balises comme ceci :

```
<racine>
  <rubrique>
    <Ligne>Message</ligne>
  </rubrique>
</racine>
```

Il est important de noter que les balises ne peuvent pas contenir de caractères spéciaux (-,;...éèàù/). Ne sont autorisés que des caractères alphanumériques (lettres et chiffres) avec majuscule et/ou minuscule.

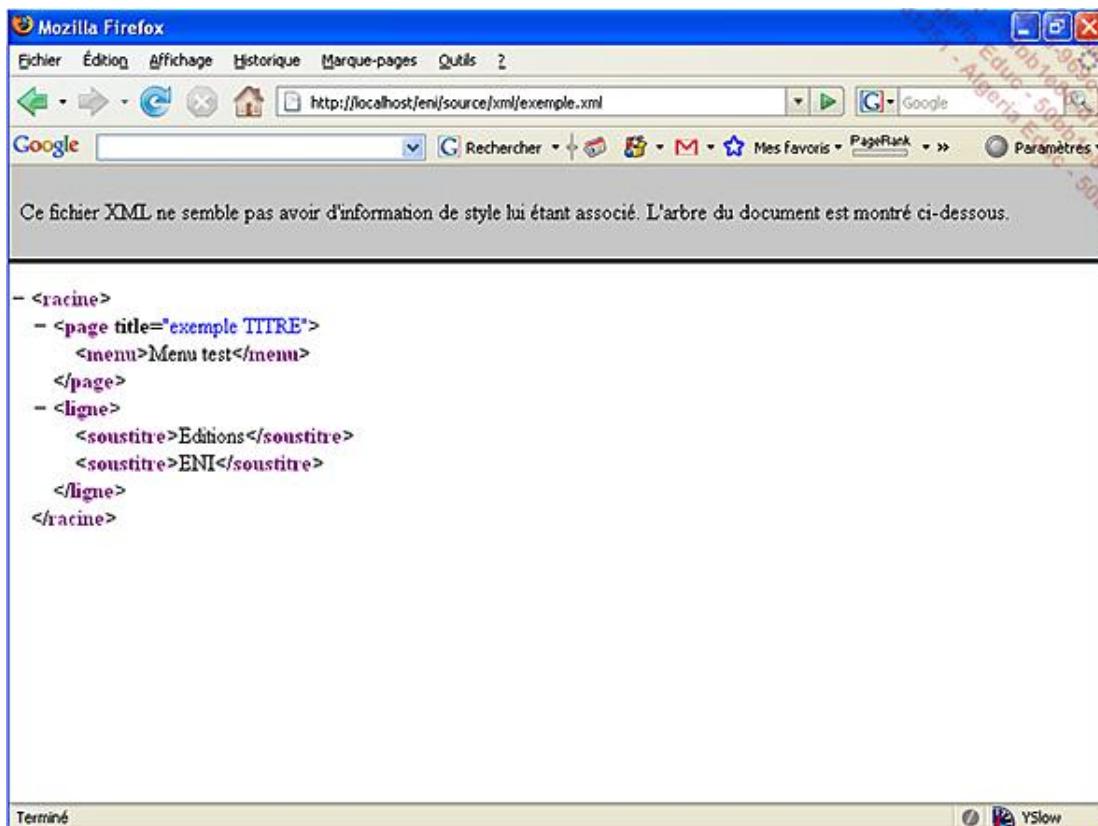
Concernant la déclaration des balises, nous allons commencer par `<racine>` et terminer par `</racine>` comme sur l'exemple ci-dessus.

Réalisons un petit exemple de fichier XML avec un titre et un message Editions ENI.

xml/exemple.xml

```
<?xml version="1.0" ?>
<racine>
<page title="exemple TITRE">
<menu>"Menu test"</menu>
</page>
<ligne>
<soustitre>Editions</soustitre>
<soustitre>ENI</soustitre>
</ligne>
</racine>
```

Voici le résultat que nous obtenons :



Maintenant, automatisons la création du fichier XML sans passer par un éditeur de texte. Pour générer le fichier, nous stockons les informations dans une variable pour réaliser deux fonctions :

- afficher le résultat à l'écran,
- créer le fichier en une seule fois.

xml/exemple1.php

```
<?php
$_xml = "<?xml version=\"1.0\" encoding=\" UTF-8\" ?>\r\n";
$_xml .= "<racine>\r\n";
$_xml .= "<page title=\"exemple TITRE\">\r\n";
$_xml .= "<menu>\"Menu test\"</menu>\r\n";
$_xml .= "</page>\r\n";

$_xml .= "<ligne>\r\n";
$_xml .= "<titre>Editions</titre>\r\n";
$_xml .= "<soustitre>ENI</soustitre>\r\n";
$_xml .= "</ligne>\r\n";
$_xml .= "</racine>\r\n";
```

Pour afficher le fichier à l'écran, nous procédons comme ci-dessous :

```
header("Content-type: text/xml");
echo $_xml;
```

Pour générer le fichier XML, nous l'écrivons avec les fonctions standards du PHP.

```
$file = fopen("exemple1.xml", "w");
fwrite($file, $_xml);
fclose($file);
?>
```

## 2. Base de données

## a. Généralités

Pour notre application, ce qui est intéressant est de générer un fichier XML à partir d'une base de données. Nous allons nous connecter à la base de données pour afficher le nom et le prénom, qui seront extraits de la table exemple. Réalisons un exemple avec une connexion MySQLi :

```
xml/exemple2.php
<?php
$serveur = "localhost";
$user    = "root";
$passwd  = "";
$bdd     = "ouvrage";

$connex = mysqli_connect($serveur, $user, $passwd, $bdd);
if (mysqli_connect_errno()) die ("Echec de la connexion :
". mysqli_connect_error());
?>
```

Pour créer le fichier XML, nous allons effectuer par l'intermédiaire d'une boucle la préparation du contenu pour ensuite l'enregistrer dans un fichier avec la nouvelle fonction **file\_put\_contents** disponible en PHP 5.

Pour rappel, le format de fichier XML est un format structuré, c'est pourquoi il est souvent nécessaire d'insérer une boucle pour afficher le contenu, car ce contenu correspond à l'affichage de lignes résultats.

```
<?php
$sql="select nom,prenom from exemple ";
$valeur=mysqli_query($connex,$sql);
if( ! $valeur ) echo "Probleme dans la table exemple :
" . mysql_error();

if (mysqli_num_rows($valeur)>0)
{
    $_xml = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>\r\n";
    $_xml .= "<personne>\r\n";
    while ($row = mysqli_fetch_object($valeur))
    {
        $_xml .= "\t\t<ligne>";
        $_xml .= "\t\t<nom>$row->nom</nom>\r\n";
        $_xml .= "\t\t<prenom>$row->prenom</prenom>\r\n";
        $_xml .= "</ligne>\r\n";
    }
    $_xml .= "</personne>";

    file_put_contents($nom_fichier,$_xml);
    echo "<a href=\"exemple2.xml\">Voir le fichier Resultat.</a>";
} else {
    echo "Aucun enregistrement";
}
?>
```

Nous fermons la connexion quand nous avons terminé :

```
<?php
mysqli_close($connex);
?>
```

La construction du fichier terminée, nous proposons d'afficher le résultat. Ce résultat peut ensuite être réaffiché ailleurs pour accéder aux informations sans avoir besoin d'aller sur le site Internet.

Par ailleurs, l'exemple nous montre qu'une boucle est possible avec comme balise `<ligne></ligne>`. Ainsi, nous pouvons permettre à un autre site Internet ou à une autre application de réutiliser le résultat des données.

## b. Mise en pratique

Dans notre application, nous allons exporter au format XML les données du carnet d'adresses pour permettre l'utilisation des données avec un autre logiciel.

Les différentes possibilités que nous rencontrons concernent entre autres l'utilisation de mailings, publipostages ou bien pour effectuer d'autres traitements divers.

 Pour information, cette partie est aussi disponible avec les formats MySQL et PDO disponibles en téléchargement sur le site de l'éditeur.

Pour commencer l'exportation, nous vérifions si le dossier **DL** existe, si ce n'est pas le cas nous le créons et ouvrons les droits en écriture dessus :

Export-XML.PHP

```
<?php
if (!is_dir($destDL))
{
    if (!@mkdir($destDL))
    {
        return array(false, 'Erreur lors de la création du dossier $destDir');
    }
}
@chmod($destDL, 0777);
?>
```

Après, nous vérifions si le fichier XML existe déjà ou pas. S'il existe, nous le supprimons :

```
<?php
$nom_fichier=$destDL."".$_SESSION['login'].".xml";
if(file_exists($nom_fichier))
{
    unlink($nom_fichier);
}
?>
```

Nous allons préparer la requête SQL pour sélectionner les adresses qui appartiennent au bon carnet d'adresses et au bon compte :

```
<?php
$sql="select carnet.*,carnet.id as idlinks,user.id,user.idclef
      FROM carnet,user
      WHERE user.id=carnet.iduser AND carnet.iduser='".$_SESSION
      ['iduser']."'"
      AND user.idclef='".$_SESSION['idclef']."' ";
assert ('mysqli_query($connex, $sql)');
$qid = mysqli_query($connex,$sql);
if (!$qid) die('Requête invalide : ' . mysqli_error($connex));
?>
```

Si nous trouvons des informations, nous allons pouvoir construire le fichier. Pour construire le fichier adresse, nous allons inscrire les champs suivants : Nom, Prénom, Adresse, Code Postal, Ville.

Pour l'explication sur la composition du fichier XML proprement dit, nous vous conseillons de vous reporter aux pages précédentes.

```
<?php
if (mysqli_num_rows($qid)>0)
{
    $_xml ="<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>\r\n";
    $_xml .="<carnet_adresse>\r\n";
    while ($row = mysqli_fetch_object($qid))
    {
        $_xml .="<personne>\r\n";
        $_xml .="\t\t<nom>$row->nom</nom>\r\n";
        $_xml .="\t\t<prenom>$row->prenom</prenom>\r\n";
        $_xml .="\t\t<adresse1>$row->adresse1</adresse1>\r\n";
        $_xml .="\t\t<adresse2>$row->adresse2</adresse2>\r\n";
        $_xml .="\t\t<codepostal>$row->codepostal</codepostal>\r\n";
        $_xml .="\t\t<ville>$row->ville</ville>\r\n";
        $_xml .="</personne>\r\n";
    }
}
```

```

    }
$_xml .= "</carnet_adresse>";
?>

```

Lorsque le contenu est défini, nous l'écrivons dans un fichier informatique, qui sera enregistré directement sur l'espace disque du dossier que nous avons choisi.

```

<?php
file_put_contents($nom_fichier,$_xml);

echo "Fichier crée<br>";
echo "<a href='".$destDL.$_SESSION['login'].".xml'>Visualier
le fichier
XML.</a>'";
} else {
echo "Aucun enregistrement possible";
}
?>

```

Nous donnons la possibilité de visualiser ce fichier pour contrôler les données. Ensuite nous fermons la connexion :

```

<?php
mysqli_close($connex);
?>

```

### 3. Le jeu de caractères

Le jeu de caractères est un point qui ne peut pas être contourné car tout d'abord votre fichier XML est un format structuré. Par contre, il est possible d'utiliser un encodage pour permettre à une autre application de respecter exactement votre format.

Cet encodage n'est pas obligatoire, mais fortement conseillé car si nous utilisons des accents, ils risquent de ne pas être interprétés correctement. C'est pourquoi lorsque nous déclarons un fichier XML, il est nécessaire de spécifier le type d'encodage.

Pour la langue française, l'encodage standard correspond au format ISO-8859-1, ce qui donne ceci dans le fichier :

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

Par contre, cette déclaration pourra être réutilisée principalement pour des applications françaises. En effet, certains caractères de la langue française peuvent être mal interprétés.

Heureusement, il existe un type d'encodage unicode, qui permet d'utiliser une table universelle : le format UTF-8.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Pour décoder les caractères UTF-8, c'est-à-dire pour la langue française convertir un caractère UTF-8 vers la norme ISO-8859-1, nous pouvons utiliser la fonction **utf8\_decode**.

#### **utf8\_decode**

Convertit une chaîne UTF-8 en ISO-8859-1.

Le format **UTF\_8** est étudié plus en détail dans le chapitre Fonctionnalités supplémentaires - **UTF8**.

### 4. SimpleXML

Depuis de nombreuses années, nous pouvons lire des fichiers XML. À partir de la version PHP 5, une nouvelle fonction est apparue **simpleXML**, permettant de lire plus facilement les fichiers RSS.

#### **simpleXML**

Convertit un fichier XML en objet.

Lorsque nous lisons un fichier et qu'il se transforme en objet, il peut être manipulé comme nous le désirons. Nous allons prendre le fichier **exemple1.xml** qui a été généré pour en afficher son contenu.

```
<?php
$nom_fichier="exemple1.xml";
?>
```

Nous devons vérifier si le fichier est bien présent pour pouvoir le charger et le convertir en objet.

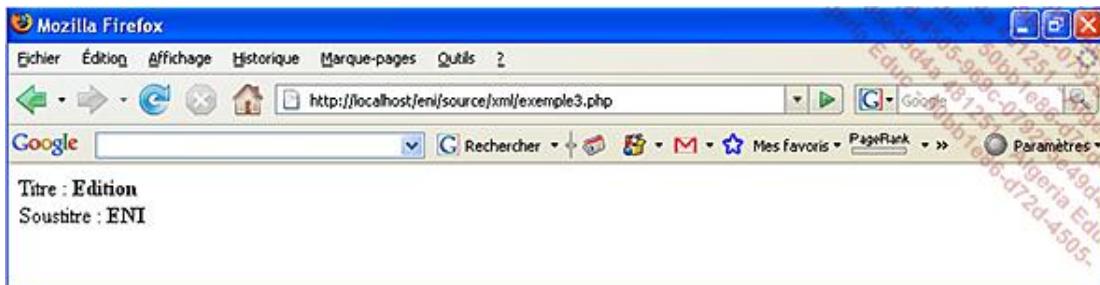
```
<?php
if (file_exists($nom_fichier))
{
    $xml = simplexml_load_file($nom_fichier);
?>
```

Lorsque l'objet est créé, nous pouvons le passer en revu pour en extraire les champs.

Nous lisons le fichier pour l'insérer dans une variable objet **\$\_xml**. Pour extraire les données de ce fichier, nous effectuons une recherche des lignes comprenant la balise **<ligne>** que nous convertissons en variables pour pouvoir l'afficher.

Ensuite, nous utilisons le changement de codage des caractères, que nous avons vu dans les pages précédentes, pour afficher les champs qui nous intéressent.

```
<?php
foreach($_xml->ligne as $ligne)
{
    echo 'Titre :<b> ' ,utf8_decode($ligne->titre).'</b><br> ';
    echo 'Soustitre :<b> ' ,utf8_decode($ligne->soustitre).'</b> ';
}
else
{
    die('Echec lors de l\'ouverture du fichier $nom_fichier.');
}
?>
```



## 5. Flux RSS

### a. Présentation

Un flux RSS, appelé aussi fil RSS, correspond à un format de syndication de contenu sur Internet, structuré en XML et qui regroupe un ensemble d'actualités, de nouvelles (news) venant de sources différentes.

Un flux RSS permet de fidéliser les lecteurs d'un site Internet, en étant au courant des nouvelles sans avoir besoin de venir régulièrement sur le site.

Pour lire ou écrire un flux RSS, il suffit de respecter la structure que nous avons vue précédemment et de remplir le fichier :

- une partie en-tête,
- une partie contenue.

L'en-tête va se présenter sous la forme suivante :

```

<title>Hello-design</title>
<link>www.hello-design.fr</link>
<description>site Hello design</description>
<langage>fr-fr</langage>

```

Le contenu, c'est-à-dire la partie qui suit, sera défini par les balises `<item>...</item>` avec comme information :

```

<item>
<title>Titre</title>
<link>Liens</link>
<description>Description</description>
</item>

```

## b. RSS manuel

Nous allons générer un fichier RSS pour montrer comment passer de la théorie à la pratique. Nous allons réaliser l'exemple avec PHP5 et MySQL.

Tout d'abord, nous établissons la connexion avec le serveur de données et lui demandons de sélectionner 5 actualités :

`rss/ex1_creat.php`

```

<?php
$connex = mysql_pconnect("localhost", "root", "");
if (!$connex) die ("Impossible de se connecter : " . mysql_error());
mysql_select_db("ouvrage", $connex);

$sql = "select titre,lien, description from 'actualite' limit 5";
$qid = mysql_query($sql);
?>

```

Nous mémorisons le contenu dans un tableau :

```

<?php
while ($row = mysql_fetch_array($qid))
{
    $return[] = $row;
}

$now = date("D, d M Y H:i:s T");
?>

```

Nous réalisons maintenant le fichier XML en le préparant dans une variable avec son en-tête et les 5 lignes dont nous avons besoin, insérées grâce à une boucle :

```

<?php
$xml = "<?xml version=\"1.0\"?>
<rss version=\"2.0\">
<channel>
    <title>Exemple Flux RSS</title>
    <link>http://www.hello-design.fr</link>
    <description>Le Flux RSS su site Hello pour exemple</description>
    <language>fr-fr</language>
    <pubDate>$now</pubDate>
    <lastBuildDate>$now</lastBuildDate>
";

foreach ($return as $line)
{
    $xml .= "\t\t<item>";
    $xml .= "\t\t<title>".htmlentities($line['titre'])."</title>\r\n";
    $xml .= "\t\t<link>".htmlentities($line['lien']).".
    </link>\r\n";
    $xml .= "\t\t<description>".htmlentities(strip_tags($line['description'])).".
    </description>\r\n";
}

```

```

        $_xml .= "\t\t</item>";
    }
$_xml .= "\t\t</channel>";
$_xml .= "\t\t</rss>";
?>

```

Il nous reste à écrire le fichier et à signaler la fin de l'opération :

```

<?php
file_put_contents ("ouvrage.rss",$_xml);
echo "Creation terminé";
?>

```

Après avoir réalisé le fichier RSS, nous devons permettre la lecture de son contenu pour nos visiteurs. Pour cela, nous chargeons le fichier RSS dans un objet pour le faire lire sous la forme d'un tableau.

`rss/ex2_view.php`

```

<?php
$nom_fichier="ouvrage.rss";
$_xml = simplexml_load_file($nom_fichier);
?>

```

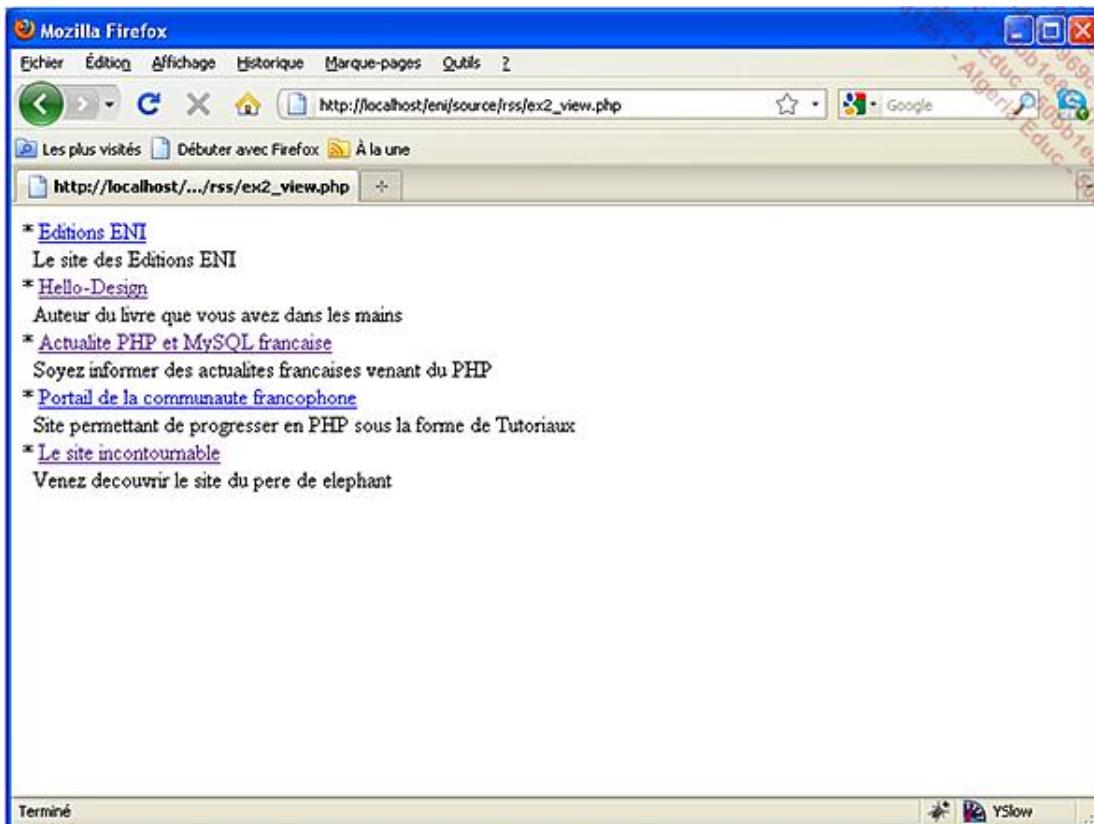
Nous pouvons ensuite personnaliser la présentation et afficher juste ce que nous désirons voir. Ici, nous affichons un lien permettant de naviguer sur la page du site Internet où se trouve l'article complet, et en dessous la description.

```

<?php
foreach($_xml->channel->item as $item)
{
    echo "* ";
    echo "<a href=\"".$item->link."\" target=\"_blank\">";
    echo $item->title;
    echo "</a><br />";
    echo "&nbsp;&nbsp;".$item->description;
    echo "<br />";
}
?>

```

Nous obtenons ceci :



### c. RSS évolué

Le flux RSS peut contenir différentes informations supplémentaires ou une présentation, comme des icônes... Bien sûr, il est tout à fait possible de générer le fichier à la main. Cependant, il existe depuis de nombreuses années des bibliothèques complètement intégrées dans le PHP. Nous allons vous montrer comment utiliser **magpieRSS**.

 Il en existe d'autres, mais celle-ci est la plus connue car c'est une des plus anciennes.

La classe magpieRSS fonctionne avec toutes les versions de PHP et permet de créer et de lire les fichiers RSS facilement.

Vous pouvez télécharger les fichiers sur le lien suivant : <http://magpierss.sourceforge.net/>

Mais la classe sera déjà installée pour l'exemple qui va suivre, si vous ne souhaitez pas effectuer cette manipulation (dans les fichiers mis à disposition en téléchargement).

Nous allons inclure la classe MAGPIERSS :

`rss/ex3_rss.php`

```
<?php
require_once( "magpierss/rss_fetch.inc" );
?>
```

Nous préparons la présentation que nous voulons afficher sous la forme d'une fonction.

Cette fonction se décompose comme ceci :

- Nous lisons le flux XML.
- Si nous avons pu lire le fichier, nous vérifions qu'il a bien été inséré dans un tableau.
- Nous récupérons les éléments les plus récents en nous limitant aux 5 derniers.
- Nous préparons les différents affichages au format HTML.

```

<?php
function FluxRSS($url_feed, $nb_items_affiches=5)
{
    $rss = fetch_rss($url_feed);
    if (is_array($rss->items))
    {
        $items = array_slice($rss->items, 0, $nb_items_affiches);
        $html = "<ul>\n";
        foreach ($items as $item)
        {
            $html .= "<li>";
            $html .= "<a href=\"".$item['link']."'>";
            $html .= $item['title']."</a></li>\n";
        }
        $html .= "</ul>\n<br><br>";
    }
    return $html;
}
?>

```

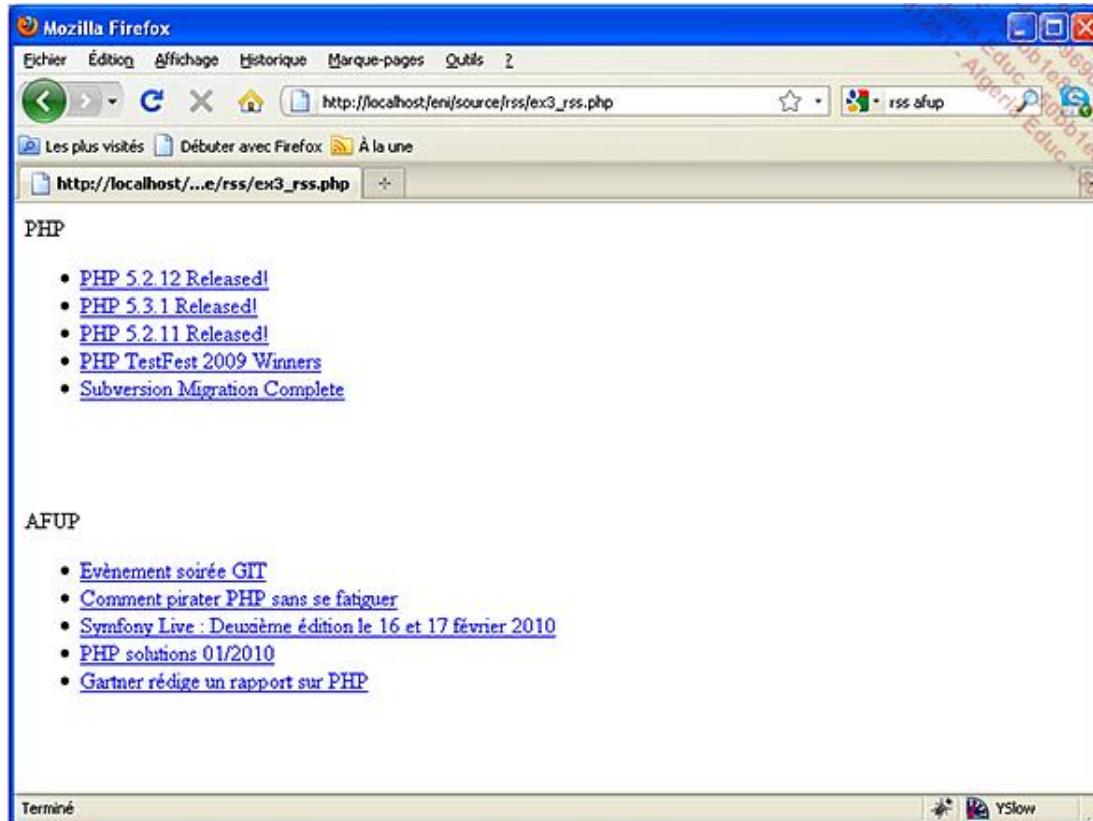
Pour afficher les différents flux qui nous intéressent, nous ajoutons directement le lien de la page internet, par exemple les dernières actualités de php.net et de l'afup.org comme ceci :

```

<?php
echo "PHP<br>";
echo FluxRSS("http://www.php.net/news.rss");
echo "<br>AFUP : <br>";
echo FluxRSS("http://www.afup.org/pages/site/rss.php");
?>

```

Pour le résultat suivant :



Cet exemple d'utilisation de flux RSS peut être inséré dans notre application pour permettre aux visiteurs de notre site Internet d'être au courant des actualités.

# Extension Json

## 1. Introduction

L'extension JSON (*JavaScript Object Notation*) existe depuis la version 5.2 de PHP. Il s'agit d'un format d'échange de données entre le navigateur et le serveur. Cette extension prend de plus en plus d'importance avec les différents modes de communication entre les sites, entre les formats comme les fichiers flash (SWF), mais est aussi utilisée par de nombreux moteurs de recherche comme Yahoo, Google.

Cette extension a été réalisée dans le but d'effectuer des échanges de données en toute simplicité. La conception de ce format est facile à écrire et à lire pour l'ensemble des utilisateurs, comme pour les ordinateurs. Il se base sur deux structures :

- une liste de couples nom/valeur comme le propose d'autres formats (XML par exemple),
- une liste de valeurs ordonnées.

## 2. Structure

La structure d'un fichier JSON est stricte, il est nécessaire de respecter la syntaxe :

```
{  
  « nom » : « valeur »  
}
```

Comme le montre la structure ci-dessus, le contenu d'une ligne représente une ligne « membre », c'est pourquoi nous pouvons mettre plusieurs lignes à la suite, avec à la fin de chaque ligne une virgule, comme ceci :

```
{  
  « nom » : « valeur »,  
  « nom » : « valeur »,  
  « nom » : « valeur »  
}
```

Ce format peut contenir des valeurs en tableaux, permettant d'envoyer des informations structurées.

```
{  
  « nom » : « valeur »,  
  « data » :  
  [  
    {  
      « nom » : « valeur »,  
      « action » : « valeur »  
    },  
    {  
      « nom » : « valeur »,  
      « action » : « valeur »  
    }  
  ]  
}
```

Réalisons un petit exemple de format JSON avec le langage PHP, en utilisant les fonctions suivantes :

**Json\_encode()** : retourne la présentation JSON d'une valeur.

**Json\_decode()** : décode une chaîne JSON.

Json/exemple1.php

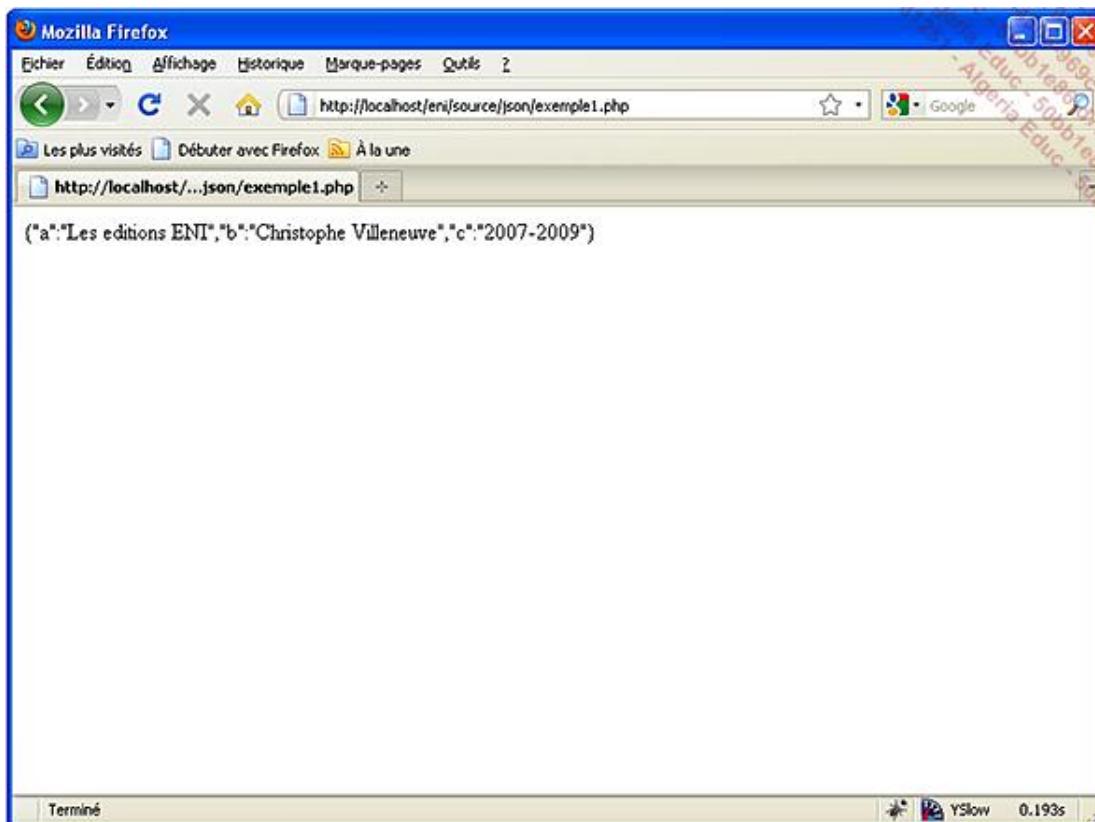
```
<?php
```

```

$arr = array (
    'a'=>'Les editions ENI',
    'b'=>'Christophe Villeneuve',
    'c'=>'2007-2009');
echo json_encode($arr);
?>

```

Voici le résultat que nous obtenons :



Pour décoder le résultat obtenu, nous prenons le résultat de l'exercice précédent et nous utilisons l'autre fonction proposée par PHP : json\_decode :

Json/exemple2.php

```

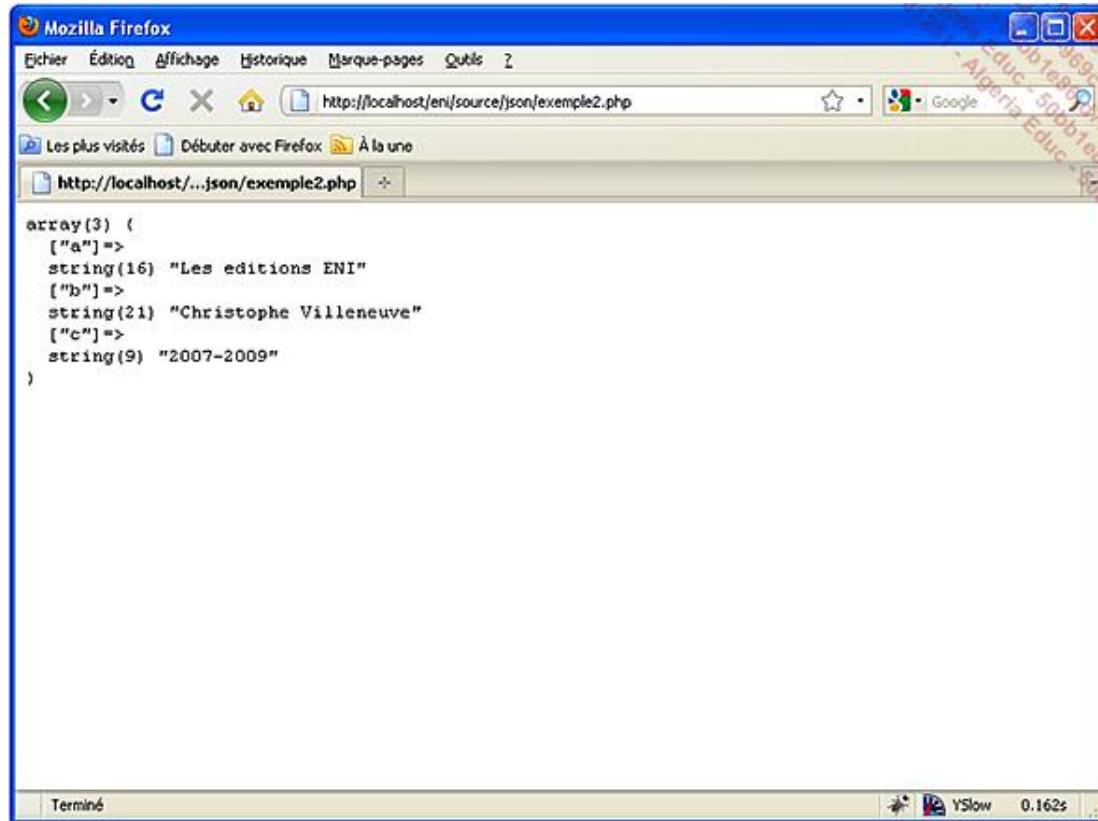
<?php
$json = ' {"a":"Les editions ENI","b":"Christophe
Villeneuve","c":"2007-2009"}';

echo "<pre>";
var_dump(json_decode($json, true));
echo "</pre>";
?>

```

Nous ajoutons à la fonction json\_decode une valeur optionnelle « true », signifiant que nous souhaitons obtenir un résultat sous la forme d'un tableau associatif.

Voici le résultat que nous obtenons :



### 3. Base de données

#### a. Généralités

Pour illustrer l'utilisation de ce format dans notre application, nous allons nous connecter à une base de données pour afficher le nom et le prénom qui seront extraits de la table « exemple ». Réalisons cet exemple avec une connexion MySQLi :

Json/exemple3.php

```
<?php
$serveur = "localhost";
$user= "root";
$passwd = "";
$bdd = "ouvrage";

$connex = mysqli_connect($serveur, $user, $passwd, $bdd);
if (mysqli_connect_errno())
    die ("Echec de la connexion : ". mysqli_connect_error());
?>
```

Pour créer le format JSON, nous allons, par l'intermédiaire d'une boucle, préparer le contenu que nous mémorisons dans un tableau. Ensuite, nous encodons notre tableau en JSON :

```
<?php
$sql="select nom,prenom from exemple ";
$result = mysqli_query($connex, $sql);

$tableau = array();
while($obj = mysqli_fetch_object($result))
{
    $tableau[] = $obj;
}

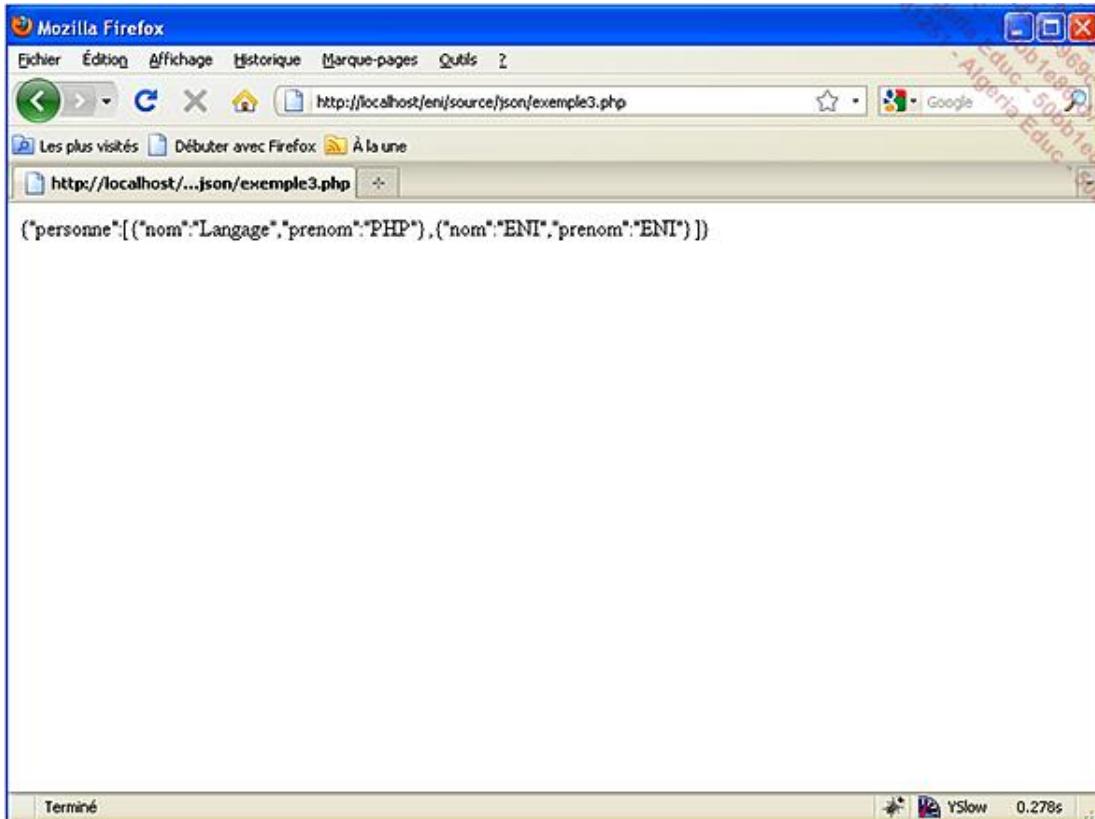
$json= '{"personne":'.json_encode($tableau).'}';
```

```
print_r ($json);
?>
```

Nous fermons la connexion quand nous avons terminé :

```
<?php
mysqli_close($connex);
?>
```

Voici le résultat que nous obtenons :



Cet exemple a été inspiré de l'exemple 2 du dossier XML (fichier xml/exemple2.php), vous pourrez ainsi comparer les deux façons d'obtenir le même résultat.

## b. Mise en pratique

Dans notre application, nous allons exporter au format JSON les données du carnet d'adresses.

Ici, nous allons demander au serveur de données les coordonnées de notre carnet d'adresses qui seront envoyées au format JSON. Nous les récupérerons pour afficher le résultat du contenu.

Bien sûr, nous avons plusieurs possibilités d'exportation comme une exportation à partir d'un fichier.

---

► Pour information, cette partie est aussi disponible avec les formats MySQL et PDO disponibles en téléchargement sur le site de l'éditeur.

---

Pour commencer, nous allons préparer la requête SQL pour sélectionner les adresses qui appartiennent au bon carnet d'adresses et au bon compte :

Fichier : export-json.php

```
<?php
$sql="select carnet.*,carnet.id as idlinks,user.id,user.idclef
      FROM carnet,user
      WHERE user.id=carnet.iduser AND carnet.iduser='". $_SESSION['iduser']."'"
```

```

AND user.idclef='". $_SESSION['idclef']."' ";
$qid = mysqli_query($connex,$sql);
if (!$qid) die('Requête invalide : ' . mysqli_error($connex));
?>

```

Si nous trouvons des informations, nous pouvons construire les données pour les proposer au format JSON.

 Pour l'explication sur la composition du JSON proprement dit, nous vous conseillons de vous reporter aux pages précédentes.

```

<?php
if (mysqli_num_rows($qid)>0)
{
$_json = array();
while($obj = mysqli_fetch_object($qid))
{
    $_json[ ] = $obj;
}

$json= json_encode($_json);
?>

```

Nous possédons maintenant une variable avec toutes les informations envoyées par le serveur au format JSON. Avant de les traiter, nous fermons la connexion avec le serveur.

```

<?php
mysqli_close($connex);
?>

```

Cette variable stocke les informations que nous venons d'extraire. Pour voir le principe d'utilisation, nous allons privilégier le décodage en JSON pour afficher le résultat dans un tableau HTML.

Pour commencer, nous décodons la variable en lui spécifiant que nous avons besoin d'un tableau associatif :

```

<?php
$resultat=json_decode($json, true);
?>

```

Ensuite, nous procédons à l'affichage de notre tableau HTML avec une boucle FOREACH.

 Pour l'utilisation de la fonction FOREACH, nous vous conseillons de vous reporter aux pages concernées de cet ouvrage (cf. chapitre La préparation du développement - Les bases du langage PHP).

```

<table border="1" width="100%">
<tr>
<td>Nom</td>
<td>Prenom</td>
<td>Adresse1</td>
<td>Adresse2</td>
<td>Code Postal</td>
<td>Ville</td>
</tr>
<?php
foreach ($resultat as $cle=>$row)
{
echo "<tr>";
echo "<td>".$row[nom]."</td>";
echo "<td>".$row[prenom]."</td>";
echo "<td>".$row[adresse1]."</td>";
echo "<td>".$row[adresse2]."</td>";
echo "<td>".$row[codepostal]."</td>";
echo "<td>".$row[ville]."</td>";
echo "</tr>";
}

```

```
?>  
</table>
```

Pour obtenir le résultat suivant :

The screenshot shows a Mozilla Firefox browser window with the following details:

- Address Bar:** http://localhost/eni/application/app-php-mysqli/export-json.php
- Page Content:**
  - Profile:** Nom: VILLENEUVE, Prenom: Christophe, Adresse1: route de Paris, Adresse2: null, Code Postal: 75000, Ville: Paris
  - Navigation Menu:**
    - Accueil
    - Votre compte
    - Rubriques
    - Votre Carnet
    - Administration
    - Exportation
      - CSV
      - PDF
      - XML
      - JSON
    - Déconnection
    - Contact
  - Page Footer:** Application - Php - Mysqli - (c) Hello-Design - Editeur ENI

# HTAccess

La fonction HTAccess est un filtre utilisé par les serveurs Apache. Elle permet :

- la protection de dossiers
- la redirection d'adresse Internet
- la gestion des erreurs de pages (ErrorDocument)

## 1. Protection de dossiers

La protection d'un dossier consiste à réserver l'accès à un répertoire uniquement à certaines personnes munies d'un identifiant (login) et d'un mot de passe (password). La protection d'un dossier s'effectue via deux fichiers :

- .htaccess
- .htpasswd

Le fichier **.htaccess** contient l'adresse du fichier **.htpasswd** avec la possibilité d'insérer quelques options si vous le désirez.

Le fichier **.htpasswd** contient les identifiants des personnes qui peuvent accéder au répertoire protégé. Nous y trouverons le login et le mot de passe de chaque personne.

Voici ce que doivent contenir les deux fichiers en question :

```
protection/.htaccess
```

```
AuthName "Accès protégé"
AuthType Basic
Require valid-user
AuthUserFile /opt3/local/apache/htdocs/votresite/.htpasswd
```

Dans l'exemple ci-dessus, deux lignes peuvent être personnalisées :

- AuthName : il s'agit du message d'alerte demandant au visiteur de remplir les cases du formulaire.
- AuthUserFile : il faut mettre le chemin complet du fichier .htpasswd

De nombreux hébergeurs proposent la création automatique de ces fichiers à partir d'une interface. Vous devrez vous rendre sur votre compte pour visualiser les options et le paramétrage.

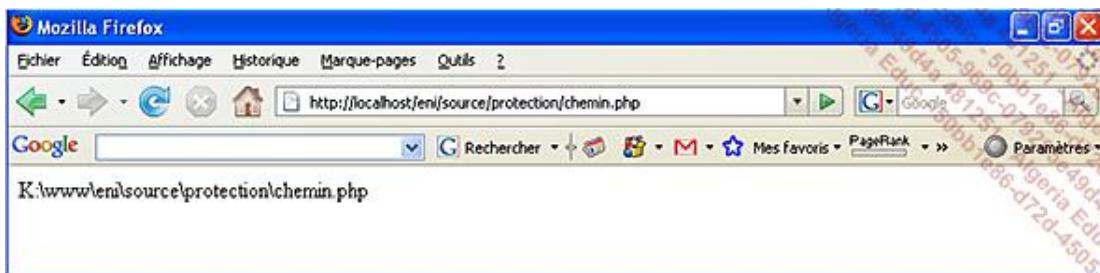
Si votre hébergeur ne propose pas ce genre d'option, il est tout à fait possible de trouver le chemin complet avec une petite fonction PHP.

### realpath

Retourne le chemin canonique absolu.

```
protection/chemin.php
```

```
<?php
echo realpath('chemin.php');
?>
```



Nous devons noter tout le chemin sauf le nom du fichier comme nous le présente l'exemple précédent. Nous devons ensuite penser à supprimer ce fichier du répertoire car il n'est pas utile de le laisser dans le dossier.

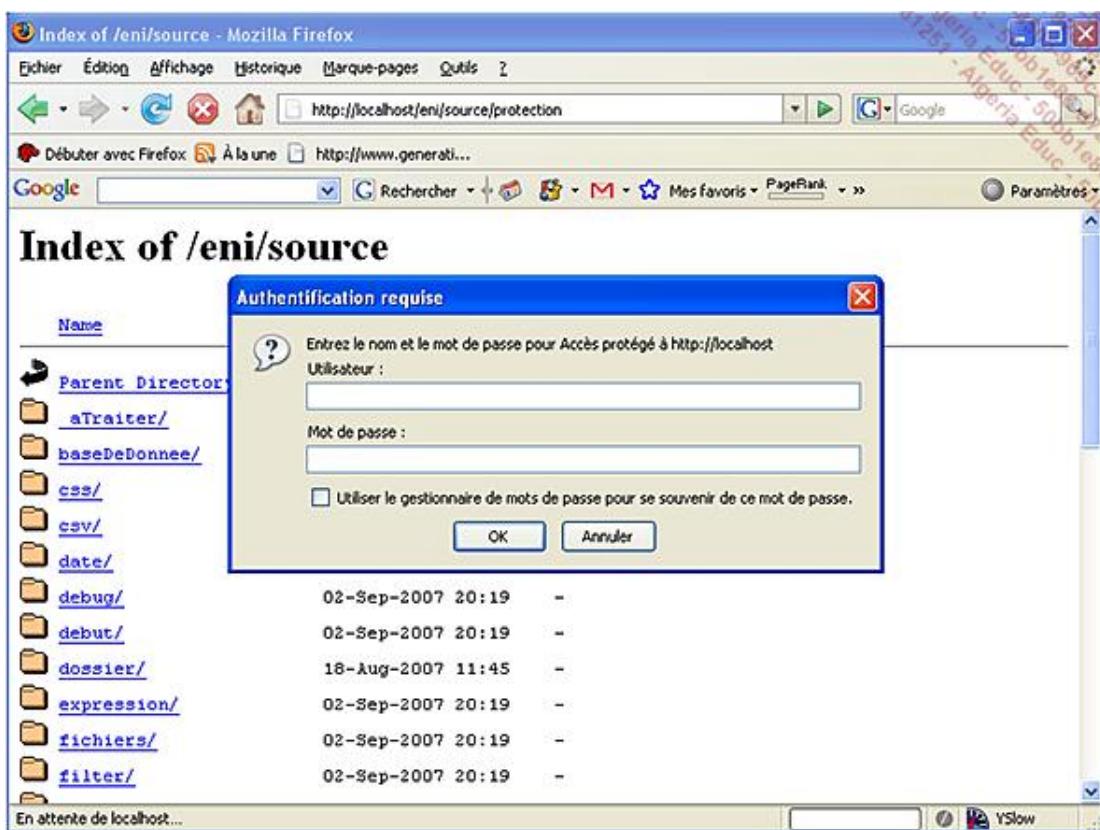
Concernant l'autre fichier ; voici comment il est composé :

```
protection/.htpasswd
```

```
login : Xfv46GgHz
test : nKd4g94qz
```

Pour chaque ligne, il est déterminé l'identifiant et son mot de passe. Ce mot de passe peut être crypté (comme au-dessus).

Le fichier .htpasswd doit toujours se trouver accompagné du fichier .htaccess.



## 2. Redirection d'adresse

### a. Présentation

Le référencement d'un site Internet n'est jamais évident pour être bien positionné dans les moteurs de recherche.

La redirection (appelée aussi rewriting), permet d'optimiser la position de votre site Internet dans les moteurs de recherche. Cette technique ne va pas vous assurer la première position, mais un gain au niveau positionnement.

Le principe du rewriting est de réécrire une adresse Internet (URL) complexe en une adresse dynamique plus clair.

Lors de la navigation dans le site Internet, nous pouvons trouver ce genre l'URL : <http://nomdusite.com/exemple.php?id=1&code=3&ligne=4&valide=1>

Ce type de lien peut être très long c'est pourquoi il est intéressant de réaliser, grâce à la redirection, la présentation de l'URL comme ceci : <http://nomdusite.com/exemple-1-3-4-1.html>

## b. Configuration du serveur Apache

Lorsque nous considérons PHP et la réécriture d'URL, le serveur le plus utilisé se trouve être Apache, c'est pourquoi il est nécessaire d'étudier comment le configurer.

Habituellement, la configuration d'un serveur Apache n'est pas nécessaire car l'ensemble des hébergeurs proposent cette technique par défaut.

Sur un serveur localhost, ou si vous possédez un serveur dédié, il est nécessaire d'effectuer cette configuration.

Pour activer ce mode sur un serveur Apache, nous devons éditer le fichier httpd.conf et activer les deux lignes suivantes :

- LoadModule rewrite\_module modules/mod\_rewrite.so
- AddModule mod\_rewrite.c

Puis, redémarrer le serveur.

Ensuite, nous devons créer un fichier .htaccess :

```
rewriting/.htaccess
```

```
Options +FollowSymlinks
RewriteEngine on

RewriteRule ^exemple\.html$ /exemple.php [L]
```

Nous devons mettre ce fichier .htaccess à la racine de notre espace, comme ceci nous saisissons l'adresse suivante <http://votresite.com/exemple.html>.

Le serveur Apache va traduire ce lien en <http://votresite.com/exemple.php>

Bien sûr, il est possible lorsque nous accédons à une page Internet, de voir des informations comprenant des valeurs après le "?". Nous allons envoyer comme information ceci :

```
Id=1
Code=3
Ligne=4
Valide=1
```

Pour obtenir comme lien URL ceci :

```
<html>
<body>
<a href='details-1-3-4-1.html'>Cliquer ici</a>
</body>
</html>
```

Lorsque le visiteur aura sélectionné le lien, nous rechargerons la page avec ces critères.

Le serveur, par l'intermédiaire du fichier .htaccess, va interpréter ce lien en le traduisant comme ceci :

```
details.php?id=1&code=3&ligne=4&valide=1
```

Pour convertir ce lien, nous insérons dans le fichier .htaccess la ligne suivante :

```
rewriting/.htaccess
```

```
Options +FollowSymlinks
RewriteEngine on
```

```
RewriteRule ^details-([0-9]+)-([0-9]+)-([0-9]+)-([0-9]+)\.html$  
/details.php?id=$1&code=$2&ligne=$3&valide=$4 [L]
```

Cette ligne de redirection comprend certains critères à connaître :

- ^ : détermine le début de l'url à réécrire.
- () : sert à encadrer une valeur.
- [0-9]+ : la variable possède un ou plusieurs chiffres.
- [L] : arrêt de la ligne de redirection.

En résumé, les délimiteurs de valeurs sont définis par le symbole "-". Dans la première partie, nous envoyons quatre valeurs entrantes. C'est pourquoi nous devons obtenir quatre valeurs sortantes représentées par \$1, \$2, \$3 et \$4.

D'autres symboles peuvent être utilisés :

- [a-z] : détermine les caractères alphabétiques (lettres),
- (.\*): autorise tous types de valeurs (chiffres et lettres).

### c. Utilisation

Nous pouvons utiliser cette technique dans notre application pour éviter de donner trop d'information dans la barre de navigation.

Nous allons prendre un exemple, comme la partie "Votre Carnet" et le sous menu "Consult/Modif".

```
echo "<td><a href=carnet-view-edit-".$row->carnetclef." class=links>  
Cliquez ici</a></td>";
```

Lorsque nous sélectionnons une des lignes, le serveur Apache va retraduire le lien grâce au fichier .htaccess :

```
Options +FollowSymLinks  
RewriteEngine on  
RewriteRule ^.*carnet-view-edit-(.*)\.html$ /carnet-view-edit.php?carnetclef=$1 [L]
```

Comme ceci, le serveur traduira la bonne page et se dirigera vers la bonne fiche.

## 3. Gestion des erreurs de page (Error Document)

Il s'agit là encore d'utiliser le fichier .htaccess pour gérer les différentes erreurs susceptibles d'être renvoyées par le serveur Apache. L'erreur la plus fréquente est l'erreur 404 qui signale que la page n'existe pas.

La directive appelée ErrorDocument possède un certain nombre de codes, permettant de définir le type de problème. Les principaux problèmes que nous pouvons rencontrer sont :

- Erreur 401 : le visiteur n'a pas saisi les bons codes d'identification.
- Erreur 403 : le serveur n'a pas le droit de répondre à la requête.
- Erreur 404 : le serveur n'a pas trouvé la page Internet demandée.
- Erreur 500 : erreur interne du serveur.

Il existe d'autres types d'erreurs qui ne sont pas traités dans ce livre.

Pour communiquer avec le serveur et donc utiliser ces erreurs, nous devons placer à la racine de notre site Internet le fichier .htaccess pour qu'il vérifie les demandes des utilisateurs.

```
htaccess/.htaccess
```

```
RewriteEngine on

# les erreurs
ErrorDocument 401 /error.php?err=401
ErrorDocument 403 /error.php?err=403
ErrorDocument 404 /error.php?err=404
ErrorDocument 500 /error.php?err=500
```

Lorsque le serveur repère une erreur que nous avons déterminée, il va exécuter automatiquement la page que nous avons définie. Ici, il s'agit de la page erreur.php.

Nous en profitons pour envoyer le numéro de l'erreur, ce qui nous permet d'afficher le bon message.

Nous récupérons la valeur qui nous est envoyée par l'intermédiaire de l'URL et nous mémorisons notre site Internet :

```
htaccess/error.php
```

```
<?php
if (isset($_POST['err']) || !empty($_POST['err']))      $err = $_POST['err'];
$url="http://www.notre_site.com";
?>
```

Nous allons récupérer l'erreur pour définir le bon message. La technique utilisée ci-dessous a été étudiée dans le chapitre La préparation du développement - Les conditions.

```
<?php
switch($err)
{
case "401":
    $msg = "Authetification nécessaire<br>";
    $msg .= "Votre IP : ".$_SERVER['REMOTE_ADDR']."<br>";
break;

case "403":
    $msg = "La page ".$_SERVER['REQUEST_URI']."' est en accès interdit<br>";
    $msg .= "Votre IP : ".$_SERVER['REMOTE_ADDR']."<br>";
    $msg .= "<ADDRESS>Apache/1.3.33 Server at ".$_SERVER['HTTP_HOST']."'<br>Port 80</ADDRESS>";
break;

case "404":
    $msg = "Erreur<br>";
    $msg .= "La page ".$_SERVER['REQUEST_URI']."' demandée n'existe pas<br>";
    $msg .= "Votre IP : ".$_SERVER['REMOTE_ADDR']."<br>";
break;

case "500":
    $msg = "Erreur interne du serveur<br>";
    $msg .= "Votre IP : ".$_SERVER['REMOTE_ADDR']."<br>";
break;
}
```

Nous affichons le message d'erreur dans une page HTML classique avec un lieu permettant de retourner sur le site directement.

```
<html>
<head></head>
<body>
<?php echo $msg."<br>"; ?>
<br>
Retour sur le site : <a href="<?php echo $url; ?>"><?php echo $url; ?></a>
</body></html>
```

## Extension Filter

L'extension Filter existe depuis la version 5.2 de PHP. À l'avenir, cette fonction prendra de plus en plus de place dans les futurs développements.

Cette extension permet de valider et de filtrer les données provenant de sources non sécurisées comme par exemple la saisie dans les formulaires ou un lien Internet.

Pour sécuriser la valeur de champ de saisie, et éviter de faire soi-même les fonctions de test et de contrôle à partir des fonctions déjà existantes de PHP comme les expressions régulières (cf. chapitre La préparation du développement - Expression régulière), nous allons pouvoir effectuer la même chose mais avec quelques nouvelles fonctions.

Cette extension permet de simplifier la validation des formulaires pour les développeurs PHP en proposant une interface standard avec de nombreuses fonctions. Énumérons les principales :

Filter\_validate\_int : Valeur d'un entier

Filter\_validate\_float : Nombre flottant

Filter\_validate\_regexp : Expression régulière

Filter\_validate\_url : URL

Filter\_validate\_email : Email

Filter\_validate\_string : String

Nous allons en étudier quelques-unes, par exemple : Filter\_input, Filter\_var.

### 1. Filter\_input

Cette fonction va nous permettre de valider les données provenant de la saisie d'un formulaire :

```
filter_input ( type, nom du champ [, nom du filtre [,options]] )
```

- type : détermine le type de récupération des données (GET, POST, Cookie, Server...) ;
- nom du champ : le nom du champ ;
- nom du filtre : si un filtre spécifique doit être obligatoirement exécuté.

Il existe de nombreuses sources de provenance (url, cookie, server...) pour le paramètre type :

- Input\_get
- Input\_post
- Input\_cookie
- Input\_server
- Input\_env

Pour utiliser la fonction filter\_input, étudions comment nous effectuons les tests actuellement et ensuite comment nous le faisons avec cette nouvelle fonction.

Nous allons tout d'abord utiliser un formulaire classique, pour saisir par exemple un code postal ne contenant que des chiffres.

```
filter/filter1.php
```

```

<html>
<body>
<form name="saisie" method="POST" action="filter1.php">
Code Postal <input name="codepostal" type="text"><br /><br />
<input name="Confirmer" type="submit" value="Confirmer">
</form>
</body>
</html>

```

Lors de la validation du formulaire, nous effectuons un certain nombre de tests comme ceci :

filter/filter1.php

```

<?php
if (sizeof($_POST) > 0)
{
    echo "Resultat : ".$_POST['codepostal']."<br>";

    if (empty($_POST['codepostal']) && isset($_POST['codepostal']))  )
        echo "champ obligatoire";
    elseif ( !is_numeric($_POST['codepostal']))
&& (intval(0 + $_POST['codepostal']) == $_POST['codepostal'])  )
    {
        echo ('Le champ ne correspond pas au format demandé');
    } else {
        echo "test réussi";
    }
}
?>

```

Avec ces tests, nous avons vérifié si le champ contient bien une valeur. Ensuite nous avons testé la valeur du champ, c'est-à-dire vérifié qu'il ne possède bien que des chiffres ; si ce n'est pas le cas, un problème sera signalé.

Pour finir, si tout s'est correctement déroulé, nous pouvons considérer que le champ est d'un format correct.

Nous allons maintenant effectuer les mêmes tests sur le même formulaire de saisie, mais avec la fonction filter\_input.

filter/filter2.php

```

<?php
if (sizeof($_POST) > 0)
{
$resultat=filter_input(INPUT_POST, 'codepostal', FILTER_VALIDATE_INT);
if(is_null($resultat))
    echo "Champ obligatoire";
elseif($resultat === false)
    echo "Le champ ne correspond pas au format demandé";
else
    echo "test réussi";
}
?>

```

Le résultat est identique, la seule différence ce sont les lignes de code utilisées.

## 2. Filter\_var

Il est possible de filtrer une variable avec un filtre spécifique :

filter\_var (nom du champ [, nom du filtre [,options]] )

### Nom du champ

Le nom du champ de la variable.

## Nom du filtre

Si un filtre spécifique doit être obligatoirement exécuté.

Nous allons effectuer un test sur une variable par exemple le contrôle d'un e-mail. Si le contrôle de l'e-mail ne correspond pas au format standard, nous obtenons un format invalide comme ceci :

filter/filter3.php

```
<?php
$e-mail = "livre@example.com";

if(!filter_var($e-mail, FILTER_VALIDATE_EMAIL))
    echo "Erreur format Email";
else
    echo "Email valide";
?>
```

## 3. Filter avec une expression régulière

Il est possible d'utiliser les expressions régulières avec l'extension filter.

Nous allons reprendre l'exemple que nous avons utilisé dans la partie consacrée aux expressions régulières. Nous effectuons de nouveau un test pour obtenir les lignes comme ceci :

filter/filter4.php

```
<?php
$chaine = 'Un éditeur de qualité Editions ENI';
$expression="[^Editions]";
if(filter_var($chaine, FILTER_VALIDATE_REGEXP,
            array("options"=>array("regexp" => $expression))) !== false)
    echo "Vrai";
else
    echo "Faux";
?>
```

Nous obtenons le résultat vrai, car le mot "Editions" se trouve bien dans la chaîne de caractères.

## 4. Sécurité avec Filter

Lorsque nous validons des données en provenance de l'extérieur, il est nécessaire de prévoir un peu de sécurité pour éviter un certain type d'attaque.

filter/filter5.php

```
<?php
$chaine = '<script type="text/javascript">alert("Editions ENI");</script>';
echo filter_var($chaine, FILTER_SANITIZE_STRING, FILTER_FLAG_NO_ENCODE_QUOTES);
?>
```

Nous avons la possibilité de supprimer les balises qui auraient pu être injectées à partir d'un de nos formulaires.

Le résultat, c'est qu'au lieu d'avoir subi une attaque, nous en profitons pour sécuriser la valeur de la variable.

# Cartographie

Quand nous parlons de cartographie, nous évoquons un plan dynamique car dans cet ouvrage nous réalisons des applications pour le Web.

La cartographie par l'intermédiaire d'un plan affiche une partie d'une carte, mais nous avons la possibilité de nous déplacer pour visualiser les parties cachées.

Pour afficher ces parties cachées sans recharger la page complètement, nous allons faire appel à Ajax qui est orienté Web 2.0.

Le Web 2.0 correspond à la seconde génération de l'Internet. Il se veut être une interaction entre les utilisateurs et les réseaux dynamiques (services et applications en ligne).

## 1. Ajax

Ajax est une évolution de Javascript qui propose une autre approche du développement sur Internet. Ajax permet de :

- Réaliser une gestion dynamique des données à afficher.
- Manipuler les données facilement (HTTP Request).
- Une présentation évoluée.

Les bases d'Ajax reposent sur d'anciennes technologies mais utilisent les moyens de communications modernes pour se positionner comme un langage dynamique pour l'utilisateur, donc Web 2.0.

Le principal attrait d'Ajax, est de mettre à jour ou de modifier une partie de l'écran sans être obligé de recharger complètement la page Internet. Ce rafraîchissement s'effectue en arrière-plan sans que l'utilisateur attende le rechargeement complet de la page.

Ce langage se veut plus proche de l'utilisateur et plus rapide, c'est pour cela que le langage PHP est souvent associé à Ajax pour répondre au mieux à ses attentes.

Certaines applications (API) sont souvent arrivées au premier plan car elles utilisaient le langage PHP et Ajax, citons :

- chat dynamique ;
- gestion de contenu distant ;
- les applications dynamiques sur certains téléphones mobiles ;
- des frameworks ;
- des calendriers ;
- des orientations sous la forme de plans (Maps).

Nous ne verrons pas comment programmer en Ajax. Par contre, nous allons utiliser une application déjà existante et allons la faire communiquer avec le langage PHP.

Nous allons utiliser la gestion des cartes dynamiques et plus particulièrement la plus connue : Google Maps.

Grâce à cette API, et l'application que nous avons étudiée dans cet ouvrage, nous allons pouvoir situer les personnes du carnet d'adresses sur ces cartes.

## 2. Google Maps

Google Maps propose un outil gratuit de cartographie sur Internet. La cartographie peut être présentée sous trois formes :

- un plan satellitaire pour voir le monde entier,
- un plan routier pour voir les rues et les quartiers,
- un plan mixte (satellitaire et routier).

Google Maps est basé sur une API en JavaScript, il permettra de zoomer ou de dézoomer, de se déplacer sur une carte, d'ajouter des points, des icônes personnalisées, etc.

### a. Installation

Dans un premier temps, il est nécessaire de posséder un compte Google. Pour obtenir un compte, il faut se rendre sur la page suivante : <https://www.google.com/accounts/Login?hl=fr>

Ensuite, il est nécessaire d'obtenir une clef Google. Celle-ci est gratuite, il suffit de s'inscrire en vous rendant directement sur le lien suivant : <http://www.google.com/apis/maps/signup.html>

La clé qui vous sera fournie sera liée avec le nom de domaine de votre site Internet.

Par exemple le genre de clé qui nous est fournie quand nous saisissons le nom de domaine <http://hello-design.fr> est : ABQIAABAIBuN8Mgv-QLbcJrQi8IkpBRIXxZPnFaVh\_97PJ47RT1LEJW2eVomAWSThx374GwcuRM3A

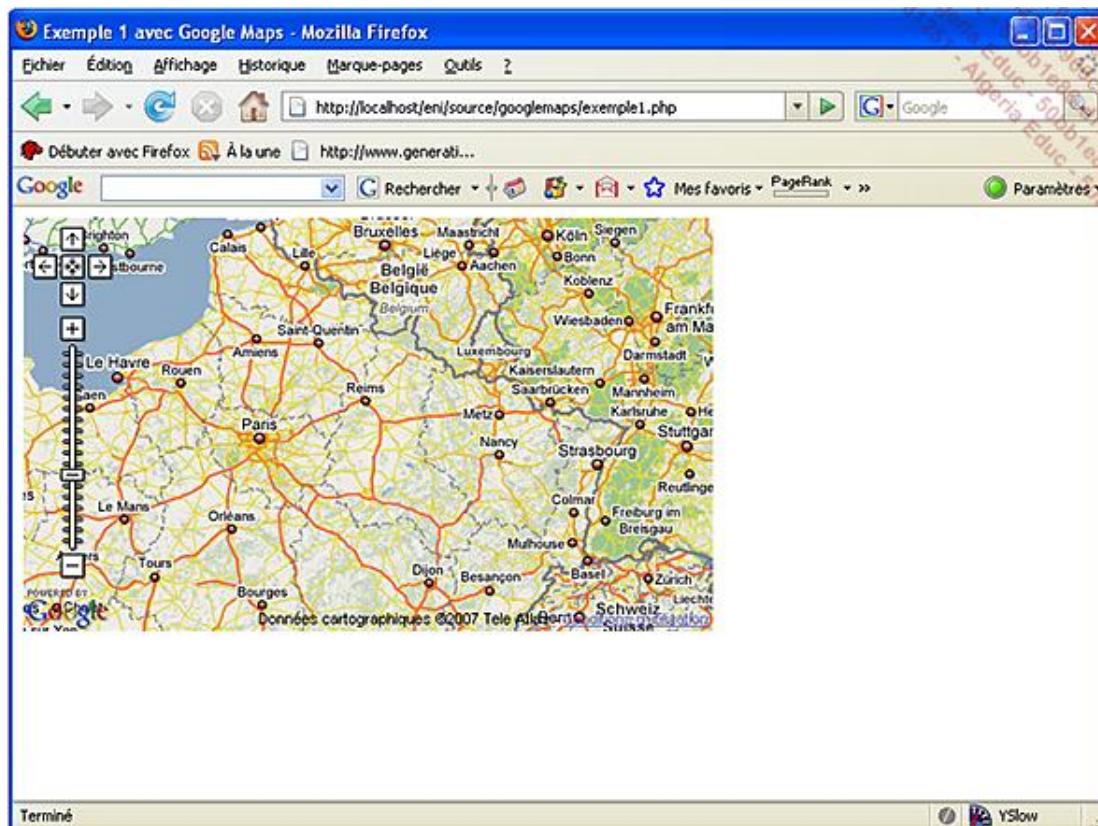
 Une petite remarque sur cette clef : si vous ne spécifiez pas la bonne clef, l'API ne pourra pas fonctionner et vous serez obligé d'effectuer une nouvelle demande.

### b. Utilisation basique

Cette API fournit tout le nécessaire pour réaliser et afficher une carte personnalisée sur votre site Internet.

Pour commencer, nous allons afficher juste une partie de la carte de France avec la possibilité de naviguer dans la carte avec le contrôleur de Map à gauche de l'écran et aussi avec la souris, dans une dimension de 400x300 pixels.

Nous obtenons le résultat suivant :



Pour afficher cette carte, nous utilisons la source ci-dessous. Tout d'abord, il est nécessaire de mettre la clef que Google vous a fournie lors de la création de votre compte. Ensuite, vous devez insérer le code JavaScript, défini par

les balises <script></script> qui permet de se connecter sur les cartes de Google.

googlemaps/exemple1.php

```
<?php $clef="VOTRE CLEF"; ?>
<html>
  <head>
    <title>Exemple 1 avec Google Maps</title>
    <script src="http://maps.google.com/maps?file=api&v=2&key=<?php echo $clef; ?>" type="text/javascript"></script>
  </head>
```

Pour afficher le résultat dans la page HTML, nous allons utiliser certaines fonctions nous permettant de visualiser la carte demandée :

- Déclarer une nouvelle carte qui se placera dans la balise DIV de la page HTML.
- Définir une position en longitude et latitude.
- Définir une taille au niveau du Zoom.
- Afficher la barre de navigation sur la gauche.

googlemaps/exemple1.php

```
<body>
<div id="map" style="width: 400px; height: 300px"></div>
<script type="text/javascript">
//<![CDATA[
  var map = new GMap2(document.getElementById("map"));
  var point = new GLatLng(48.9, 2.3);
  map.setCenter(point, 6);
  map.addControl(new GLargeMapControl());
//]]>
</script>
</body>
</html>
```

Google Maps propose de nombreuses fonctions concernant la gestion et l'affichage d'une cartographie. Découvrons les plus utilisées de ces fonctions :

- Trouver une position (longitude, latitude) : Point=new GLatLng(48.9, 2.3);
- Se positionner au centre d'un point par rapport au Zoom : map.setCenter(point, 6);
- Bloquer le déplacement dans la carte par la souris : map.disableDragging();
- Afficher une bulle avec un texte : map.openInfoWindow (map.getCenteg(), document.createTextNode (« votre texte »));
- Afficher les contrôles du zoom en petit : map.addControl(new GSmallMapControl());
- Afficher les contrôles du zoom en grand : map.addControl(new GLargeMapControl());
- Ajouter des positions sur la carte : Var point=(new GlatLng (48.9, 2.3,6);  
map.setCenter (new GlatLng (48.9, 2.3,6);  
map.addOverlay(new GGMarker(point));
- Afficher le format d'affichage : Carte, Satellite ou Mixte

```
map.addControl(new GMapTypeControl());
```

Pour retrouver toutes les fonctions existantes, accédez à la page :  
<http://www.google.com/apis/maps/documentation/reference.html>

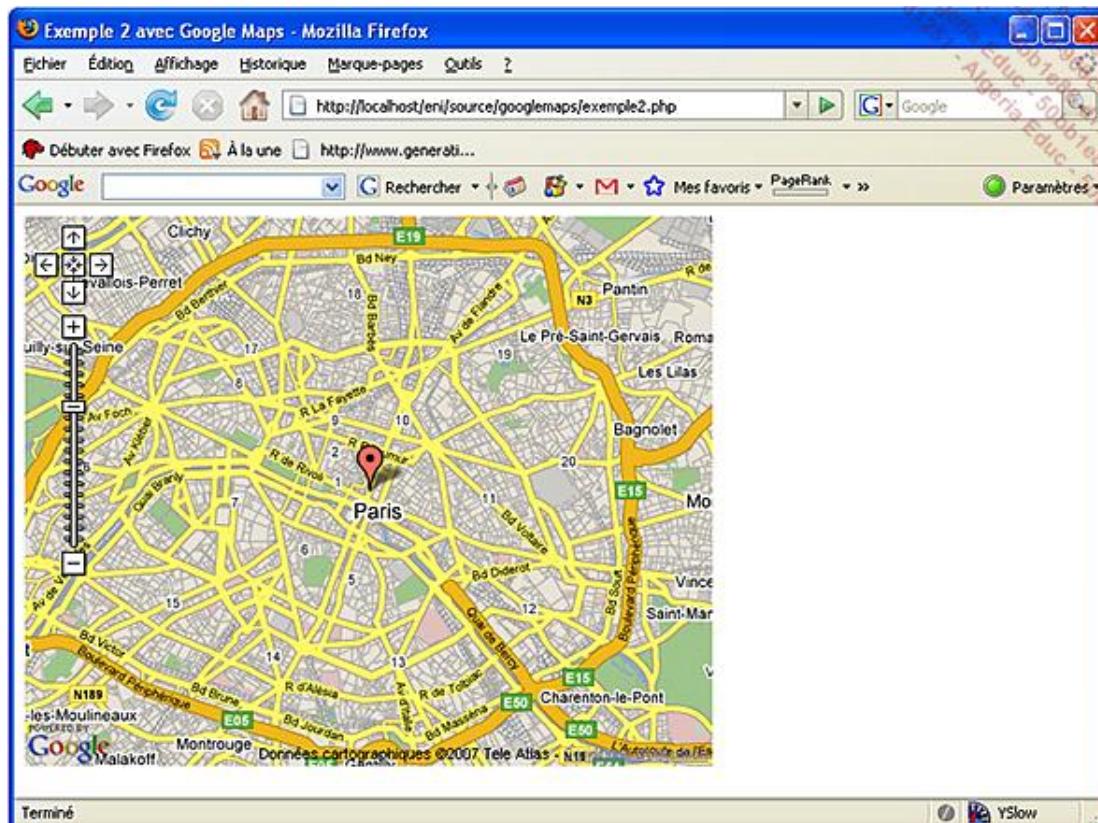
Nous pouvons ajouter un point sur une carte pour situer exactement où se trouve une adresse ou une position.

Nous utilisons le même exemple que précédemment avec la nouvelle ligne de commande et en modifiant la grosseur du zoom :

```
googlemaps/exemple2.php
```

```
<div id="map" style="width: 500px; height: 400px"></div>
<script type="text/javascript">
//<![CDATA[
var map = new GMap2(document.getElementById("map"));
var point = new GLatLng(48.86, 2.3486);
map.setCenter(point, 12);
map.addControl(new GLargeMapControl());
map.addOverlay(new GMarker(point));
//]]
&lt;/script&gt;</pre>
```

Pour obtenir le résultat suivant :



Nous pouvons en plus de la position du point, marquer un message d'indication supplémentaire.

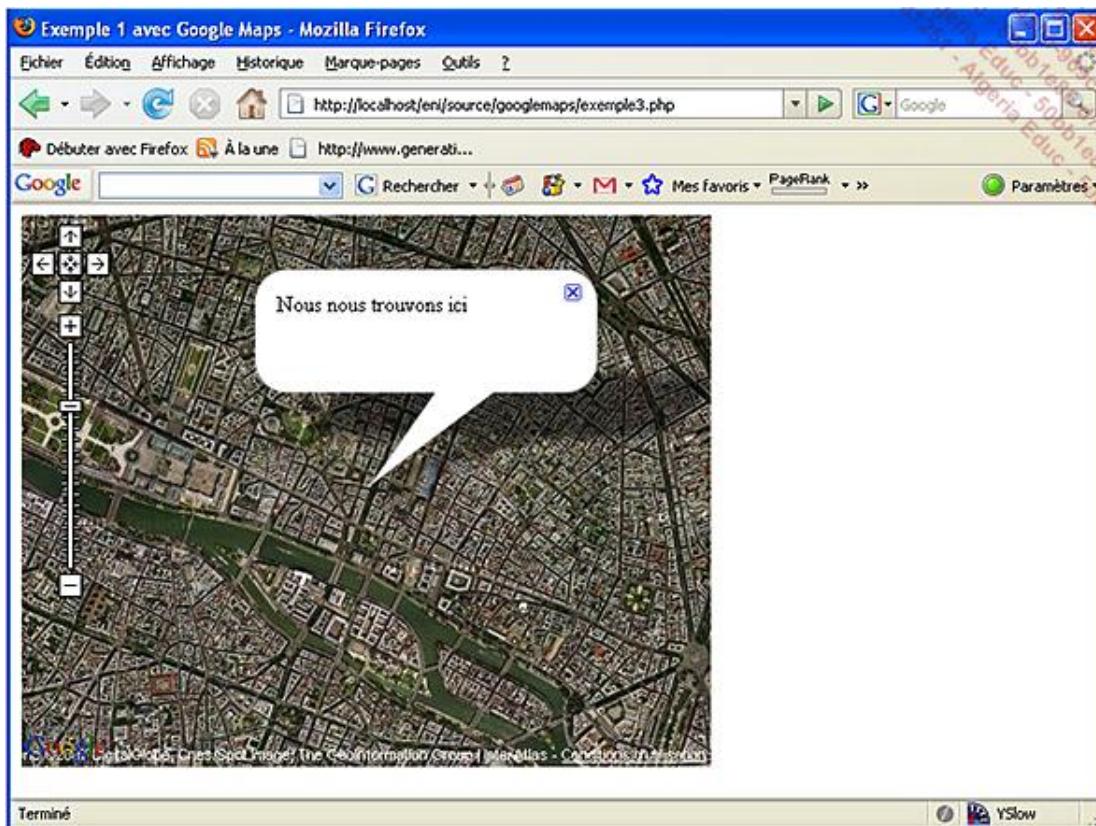
Par ailleurs, nous pouvons effectuer un affichage en mode satellite qui utilise Google Earth (une autre application de Google) :

```
googlemaps/exemple3.php
```

```
//<![CDATA[
var map = new GMap2(document.getElementById("map"));
var point = new GLatLng(48.9, 2.3);
map.setCenter(point, 14,G_SATELLITE_MAP);</pre>
```

```
map.addControl(new GLargeMapControl());
map.openInfoWindow(point, document.createTextNode("Nous nous trouvons ici"));
//]]>
```

Et nous obtenons le résultat suivant :



### c. Utilisation avec PHP

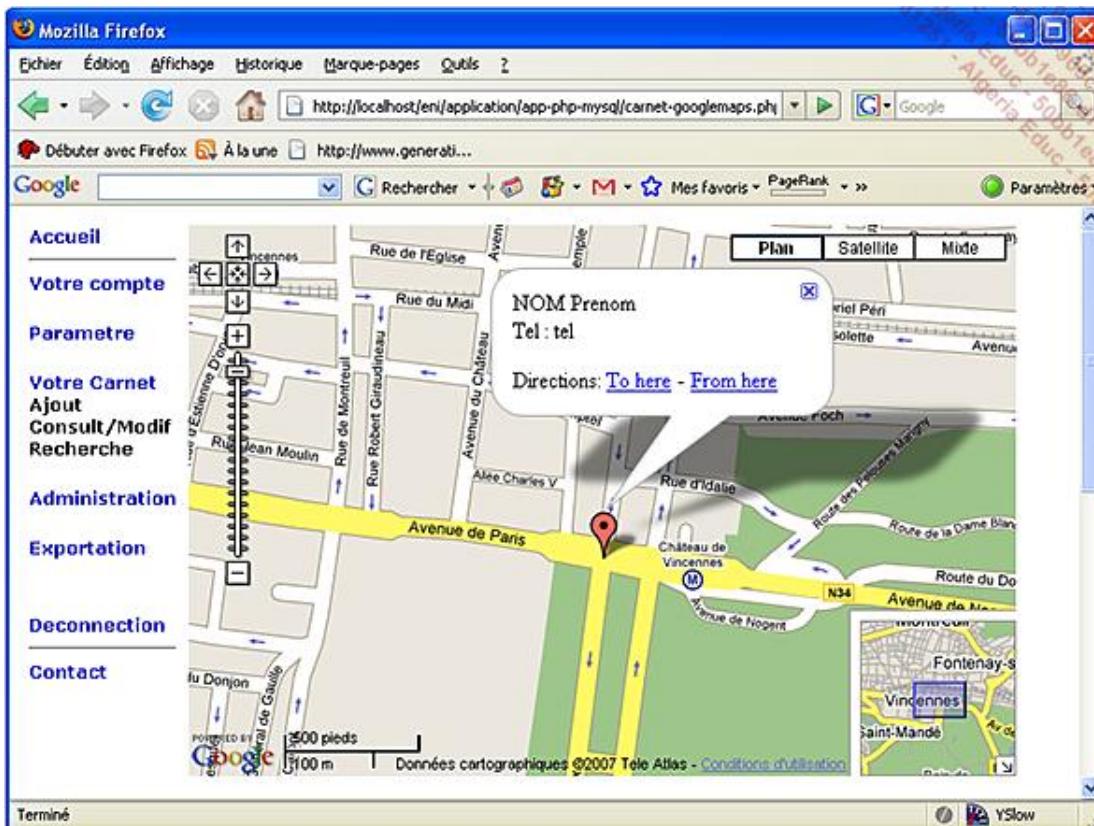
Pour revenir à notre application, nous allons proposer aux titulaires d'un compte de pouvoir situer les contacts de leur carnet d'adresses en les positionnant sur une carte.

Pour afficher la position géographique, nous utilisons la classe fournie par Google : GoogleMapAPI.class.php

Au moment de la rédaction de ce livre, la version utilisée correspond à la version 2.4.

Pour visualiser le plan, l'utilisateur devra sélectionner le sous-menu Consult/Modif du menu Carnet et sélectionner une entrée. Une fois sur la fiche du contact, il est possible d'effectuer les modifications de coordonnées. Lorsqu'il se trouvera sur cette page, il pourra accéder au plan de googleMaps.

Voici le résultat obtenu :



Pour obtenir ce résultat en PHP, nous allons utiliser la classe que nous venons de récupérer. Nous pouvons utiliser la même clef que dans les pages précédentes.

```
<?php
require_once('include/GoogleMapAPI.class.php');
$clef="VOTRE CLEF";
?>
```

Nous utilisons PHP et nous créons un objet carte avec une largeur et hauteur précise.

```
<?php
$map = new GoogleMapAPI('map', 'map');
$map->setAPIKey($clef);
$map->setWidth("600");
$map->setHeight("400");
?>
```

Nous déterminons le type d'affichage en affichant le menu pour sélectionner la présentation en forme de carte, par satellite ou les deux.

Nous affichons en même temps une petite carte en bas à droite pour avoir une vue un peu plus éloignée.

```
<?php
$map->setMapType('map');
$map->enableOverviewControl();
?>
```

Nous préparons les coordonnées postales avec les mêmes critères que pour l'édition de la fiche. Ici, il s'agit de l'utilisation avec une base de données MySQL (les applications avec les formats MySQLi et PDO sont disponible en téléchargement sur le site de l'éditeur).

```
<?php
$sql="select carnet.*,user.id,user.idclef
FROM carnet,user
WHERE user.id=carnet.iduser
AND carnet.carnetclef='".$carnetclef."'
AND carnet.iduser='".$_SESSION['iduser']."'"
AND user.idclef='".$_SESSION['idclef']."'"
";
```

```
assert ('mysql_query($sql)');
$qid = mysql_query($sql);
if (!$qid) die ("Probleme : " . mysql_error());
?>
```

Nous mémorisons les informations sous une certaine forme pour définir un point de marquage et le texte associé à ce point. Ici il comprend le nom et prénom et le numéro de téléphone.

```
<?php
$row=mysql_fetch_object( $qid );
$adresse=$row->adresse1.", ".$row->codepostal.", ".$row->ville;
$autre=$row->nom." ".$row->prenom."<br>Tel : ".$row->tel;
$map->addMarkerByAddress($adresse,$titre,$autre);
?>
```

Lorsque nous avons préparé toutes les informations, nous pouvons charger l'API dans la page HTML.

```
<?php
$map->printHeaderJS();
$map->printMapJS();
$map->printMap();
?>
```

Les trois lignes ci-dessus permettent de positionner correctement l'adresse que nous avons retenue.

Il existe de nombreuses autres fonctions qui sont fournies avec cette classe, mais nous vous laissons le soin de les découvrir.

# Les fonctions particulières

Nous ne saurions clôturer cet ouvrage sans aborder certains aspects plus généraux de php. Il s'agit d'éléments qui sont à connaître pour l'ensemble des développements, et quel que soit l'état d'avancement de votre projet :

- avant de commencer ;
- pendant le développement ;
- lors de la mise en production.

## 1. PHPINFO

La fonction **phpinfo()** permet d'obtenir certaines informations sur votre environnement, votre serveur et la version du langage utilisée.

### phpinfo

Affiche de nombreuses informations sur PHP.

particularite/phpinfo.php

```
<?php
echo phpinfo();
?>
```

Le résultat sera obtenu sous la forme d'un tableau comme ci-dessous. Nous n'expliquerons pas toutes les lignes, car le résultat est très volumineux et fournit de nombreuses lignes de fonctions et d'accès.

PHP Version 5.2.11	
System	Windows NT HELLO-66D609EBF 5.1 build 2600
Build Date	Sep 16 2009 19:39:11
Configure Command	./configure --enable-snapshot-build --enable-debug-pack --with-snapshot-template=d:\php-sdk\snaps_5_2\vc6\8\php_build --with-pdo-oci=D:\php-sdk\oracle\instantclient10\lsdk\shared --with-oci8=D:\php-sdk\oracle\instantclient10\lsdk\shared
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\wamp\bin\apache\Apache2.2.11\bin\php.ini
Scan this dir for additional ini files	(none)
additional .ini files parsed	(none)
PHP API	20041225

Cette fonction est à utiliser avec précaution car elle affiche beaucoup d'informations sur le serveur et aussi sur le visiteur. Elle sera utile pour connaître exactement les fonctions disponibles sur le serveur. Mais lorsque nous connaissons les possibilités des fonctions, et donc avant de mettre en ligne le site internet, il est préférable de supprimer cette ligne car elle pourrait être utilisée pour des attaques malveillantes. Les lignes qui nous intéressent le plus correspondent à :

- la langue de l'ordinateur qui est la plus utilisée ;
- le navigateur ;

- la version de PHP ;
- les bases de données acceptées ;
- l'IP du visiteur ;
- l'Adresse de provenance.

Cette fonction **PHPINFO** est une fonction à double tranchant : elle est très complète, donc très utile, mais elle permet de vérifier le résultat par rapport aux différentes fonctions PHP pouvant exister.

## 2. Informations du serveur

Les informations provenant du serveur peuvent être utilisées grâce à différentes fonctions spécifiques du PHP. Cela nous permet d'obtenir des éléments utiles qui seront nécessaires dans un cas précis. Ces fonctions peuvent aussi permettre d'exécuter certaines manipulations ou redirections vers une autre page Internet.

Nous avons déjà étudié certaines d'entre-elles dans cet ouvrage, nous allons les revoir mais aussi en découvrir d'autres.

### PHP\_SELF

Cette fonction est souvent utilisée pour le développement et la gestion des formulaires. Elle s'écrit de la façon suivante :

```
$_server['PHP_SELF']
```

Dans ce livre, nous utilisons souvent cette fonction car lors du développement avec des formulaires nous sommes souvent obligés de recharger la page pour contrôler les données saisies.

L'avantage de cette fonction est de recharger la page en cours, sans chercher à savoir si le nom de fichier qui se trouve dans la page Internet est le bon. Si nous décidons de changer le nom de la page, il serait nécessaire de mettre ce nouveau nom à la place et pour éviter ce genre de manipulations, cette fonction le fait tout seul.

### HTTP\_POST

Cette fonction affiche l'hôte, c'est-à-dire le nom de notre espace Web. Elle s'écrit de la façon suivante :

```
$_SERVER['HTTP_POST']
```

### HTTP\_REFERER

Cette fonction affiche la provenance de votre visiteur. Elle s'écrit de la façon suivante :

```
$_SERVER ['HTTP_REFERER']
```

### DOCUMENT\_ROOT

Cette fonction affiche le répertoire racine de l'arborescence des documents sur le serveur. Elle s'écrit de la façon suivante :

```
$_SERVER ['DOCUMENT_ROOT']
```

### QUERY\_STRING

Cette fonction affiche le contenu de la chaîne qui suit l'url de la page. C'est-à-dire tout ce qui se trouve après le point interrogation « ? ». Elle s'écrit de la façon suivante :

```
$_SERVER ['QUERY_STRING']
```

### REQUEST\_METHOD

Cette fonction permet de connaître la méthode utilisée lors de l'envoi des données à partir d'un formulaire. Elle s'écrit de la façon suivante :

```
$_SERVER ['REQUEST_METHOD']
```

### REMOTE\_ADDR

Cette fonction permet de connaître l'adresse ip de la machine qu'utilise le visiteur. Elle s'écrit de la façon suivante :

```
$_SERVER ['REMOTE_ADDR']
```

### **[REQUEST\_URI]**

Cette fonction permet de connaître le chemin du script. Elle s'écrit de la façon suivante : `$_SERVER ['REQUEST_URI']`

Nous pouvons montrer comment obtenir ces informations, grâce à un petit script :

```
particularite/serveur.php
```

```
<?php
echo "Hôte : ".$_SERVER['HTTP_POST']."<br>";
echo "Provenance : ".$_SERVER['HTTP_REFERER']."<br>";
echo "Racine du serveur : ".$_SERVER['DOCUMENT_ROOT']."<br>";
echo "Paramètres : ".$_SERVER['QUERY_STRING']."<br>";
echo "Méthode : ".$_SERVER['REQUEST_METHOD']."<br>";
echo "IP : ".$_SERVER['REMOTE_ADDR']."<br>";
echo "Chemin : ".$_SERVER['REQUEST_URI']."<br>";
?>
```



## **3. Les visiteurs**

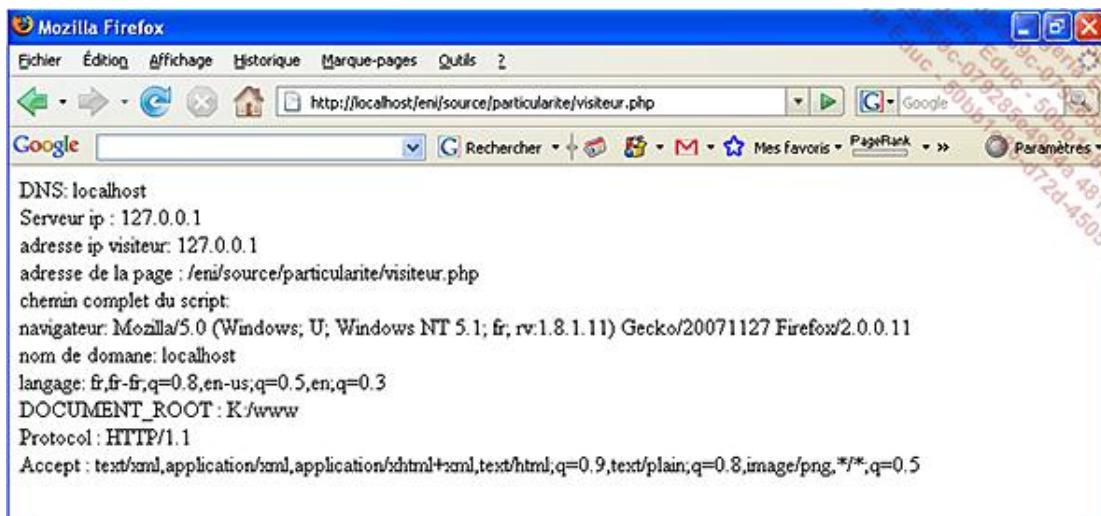
Lorsqu'une personne accède à notre site Internet, il est parfois utile de disposer de certaines informations comme son environnement ou sa provenance réelle.

Ce qui suit présente un aperçu global, car ce sont des points qui ont été déjà abordés dans l'ouvrage. Nous pouvons considérer qu'il s'agit d'un petit résumé, que nous pouvons insérer sans problème dans nos différentes applications.

```
particularite/visiteur.php
```

```
<?php
echo "DNS: ".gethostbyaddr($_SERVER['SERVER_ADDR'])."<br>";
echo "Serveur ip : ".$_SERVER['SERVER_ADDR']."<br>";
echo "adresse ip visiteur: ".$_SERVER['REMOTE_ADDR']."<br>";
echo "adresse de la page : ".$_SERVER['REQUEST_URI']."<br>";
echo "chemin complet du script: ".$_SERVER['PATH_TRANSLATED']."<br>";
echo "navigateur: ".$_SERVER['HTTP_USER_AGENT']."<br>";
echo "nom de domaine: ".$_SERVER['HTTP_HOST']."<br>";
echo "langage: ".$_SERVER['HTTP_ACCEPT_LANGUAGE']."<br>";
echo "DOCUMENT_ROOT : ".$_SERVER['DOCUMENT_ROOT']."<br>";
echo "Protocol : ".$_SERVER['SERVER_PROTOCOL']."<br>";
echo "Accept : ".$_SERVER['HTTP_ACCEPT']."<br>";
?>
```

Nous obtenons le résultat suivant :



# UTF-8

## 1. Introduction

Dans l'ouvrage, certaines pages signalent l'utilisation de la norme UTF-8. Cette norme se répand énormément pour permettre de communiquer entre différents sites internet qui se situent dans différents pays, avec des langues différentes de la langue française.

La norme UTF-8 permet de remplacer les différents encodages ISO européens pouvant exister. Le choix d'une norme se trouve souvent lié à l'utilisation des caractères spéciaux et des caractères accentués.

## 2. Généralités

Lorsqu'une norme est retenue, elle est souvent liée à la configuration d'hébergement du serveur. Cette norme peut être modifiée grâce aux développements.

Pour imposer UTF-8 aux navigateurs des internautes, nous pouvons insérer une ligne HTML spécifique, comme ceci :

Utf-8/exemple1.html

```
<html>
<head>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8" />
</head>
<body>
Mode UTF-8
</body>
</html>
```

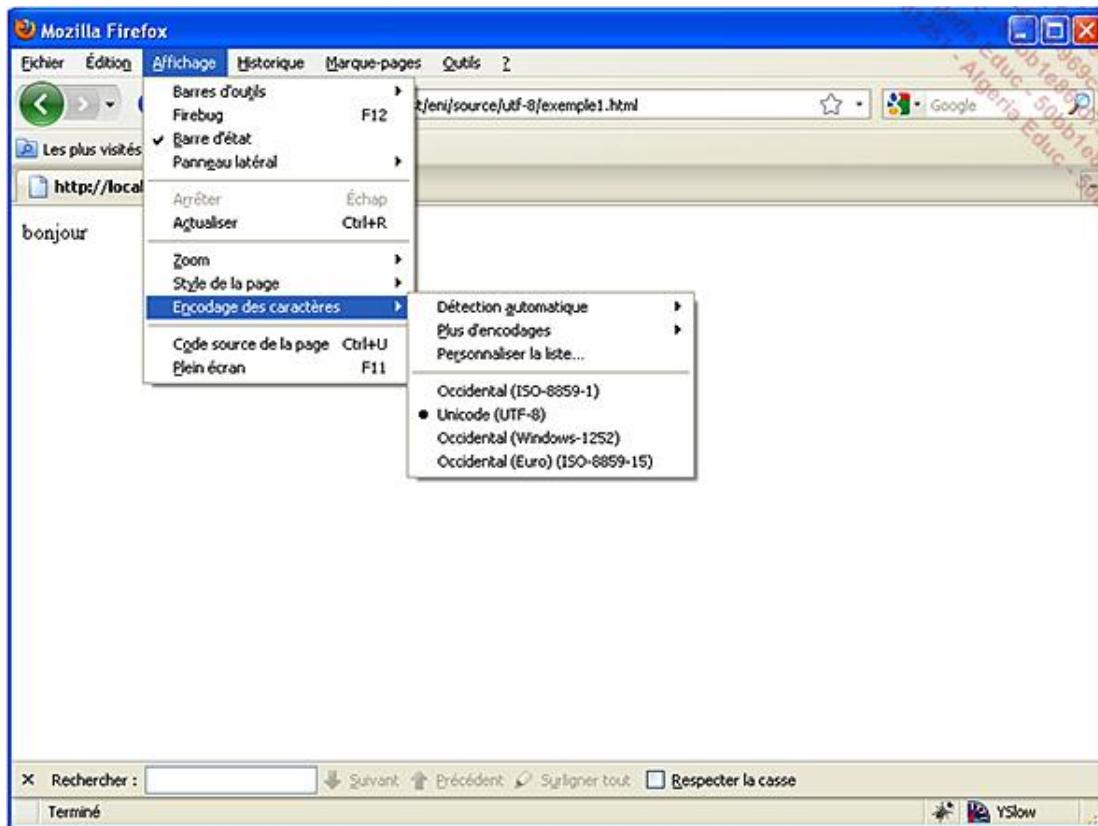
ou de la façon suivante :

Utf-8/exemple2.php

```
<?php
header ('Content-type: text/html; charset=UTF-8');
echo "Mode UTF-8";
?>
```

Lorsque nous vérifions l'encodage de notre navigateur, nous pouvons lui imposer la norme ISO qui est utilisée en Europe et lancer notre exemple ci-dessus.

Le résultat obtenu est le suivant :



La norme actuellement utilisée en PHP est la norme ISO. En PHP 6, la norme d'encodage sera UTF-8 permettant ainsi de communiquer entre les différentes langues de chaque pays.

### 3. Mise en pratique

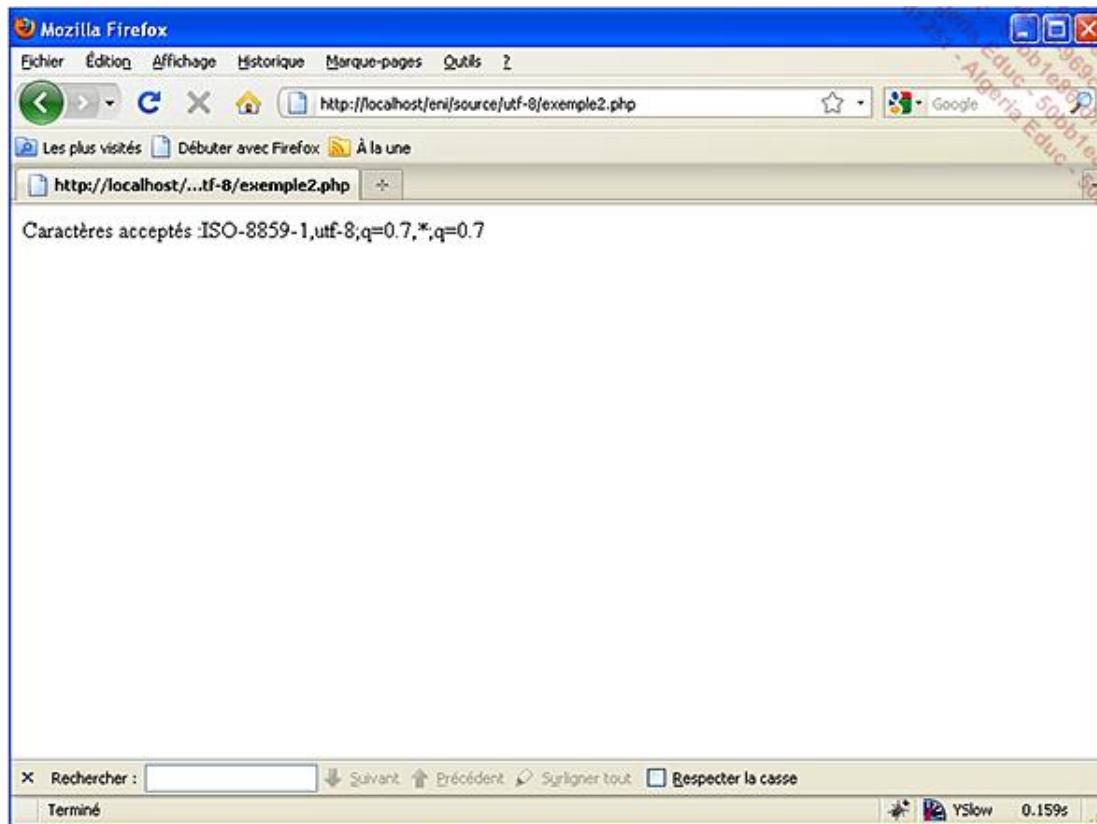
#### a. Vérification

Nous pouvons vérifier la norme présente sur notre serveur en effectuant ce petit test :

Utf-8/exemple3.php

```
<?php
echo "Caractères acceptés : ".$_SERVER['HTTP_ACCEPT_CHARSET']."<br>";
?>
```

Pour obtenir le résultat suivant :



## b. Identifications

Lorsque nous exécutons une page web, nous pouvons obtenir l'affichage de caractères différents suivant l'interprétation et la configuration de notre ordinateur.

Les caractères se présentent sous différentes formes :

- "Ã©", "Ã®", "Ã..." : l'enregistrement des données est au format UTF-8 et notre navigateur affiche au format ISO.
- "?" : l'enregistrement des données est au format ISO et notre navigateur affiche au format UTF-8.

## c. Manipulations

Les manipulations des normes de codages sont possibles lorsque les données proviennent de normes différentes comme lors de l'utilisation d'une base de données.

Les fonctions disponibles :

**utf8\_encode()** : convertit une chaîne ISO-8859-1 en UTF-8.

**utf8\_decode()** : convertit une chaîne UTF-8 en ISO-8859-1.

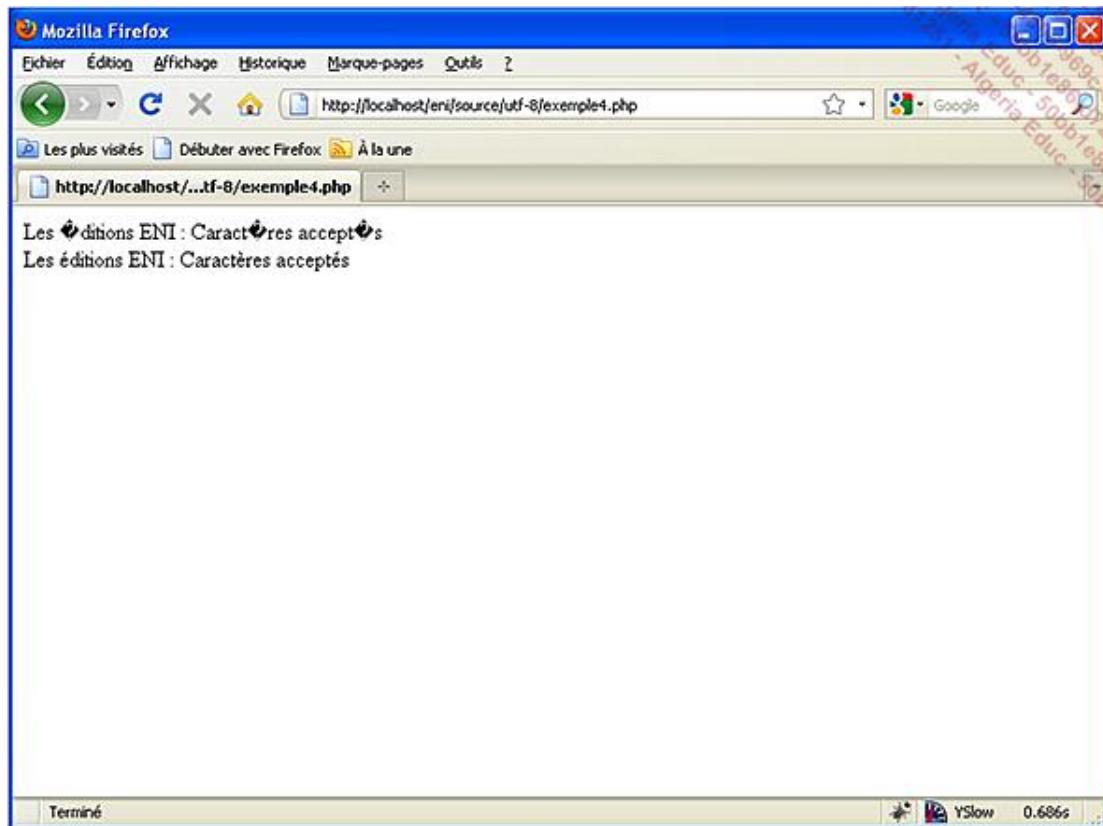
Elles s'utilisent de la façon suivante :

Utf-8/exemple4.php

```
<?php
$chaine="Les éditions ENI : Caractères acceptés ";
echo $chaine."<br>";

$chaine_encoder=utf8_encode($chaine);
echo $chaine_encoder."<br>";
?>
```

Nous obtenons le résultat suivant :



# CURL

cURL signifie "client URL Request Library" et possède une interface en ligne de commande destinée à récupérer le contenu d'une ressource accessible sur le Web. Grâce à cette extension, nous pouvons échanger des données provenant de différents sites Internet (Twitter, Facebook...).

## 1. Généralités

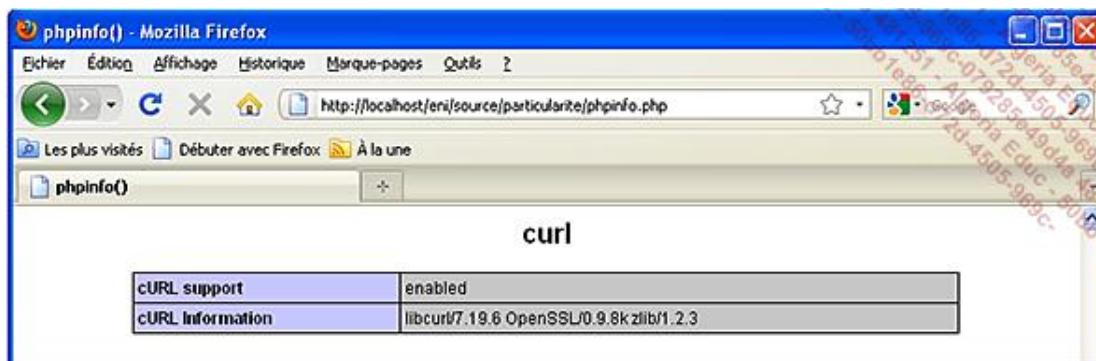
L'extension cURL n'est pas toujours disponible avec la version de PHP que vous utilisez. Pour savoir si cette extension est disponible, nous allons effectuer un petit test :

Particularité /phpinfo.php

```
<?php  
echo phpinfo();  
?>
```

➤ L'utilisation de cette fonction est décrite dans l'ouvrage et il est conseillé de relire cette partie (cf. section PHPINFO).

Le résultat que nous obtenons va nous permettre de trouver la ligne représentée par l'écran suivant :



Si nous ne voyons pas la ligne, nous devons éditer le fichier php.ini qui est disponible sur notre serveur local et enlever le « ; » devant la ligne `php_curl`.

➤ Attention sur chaque système d'exploitation, les extensions sont différentes : Mac : `extension=php_curl.so`, Windows : `extension=php_curl.dll`, Linux : `extension=php_curl.so`

Après avoir supprimé le point-virgule de la ligne, nous sauvegardons et relançons notre serveur pour que cette modification soit prise en compte.

Nous allons vérifier que notre extension fonctionne en utilisant les fonctions suivantes :

**curl\_init()** : initialise une session cURL.

**curl\_setopt()** : définit une option de transmission cURL.

**curl\_errno()** : retourne le dernier message d'erreur cURL.

**curl\_exec()** : exécute une session cURL.

**curl\_close()** : ferme une session cURL.

Pour obtenir un code source exemple comme celui-ci :

Curl/curl.php

```
<?php  
$chemin = curl_init("http://www.php.net/");  
curl_setopt($chemin, CURLOPT_RETURNTRANSFER, true);
```

```

$data = curl_exec($chemin);

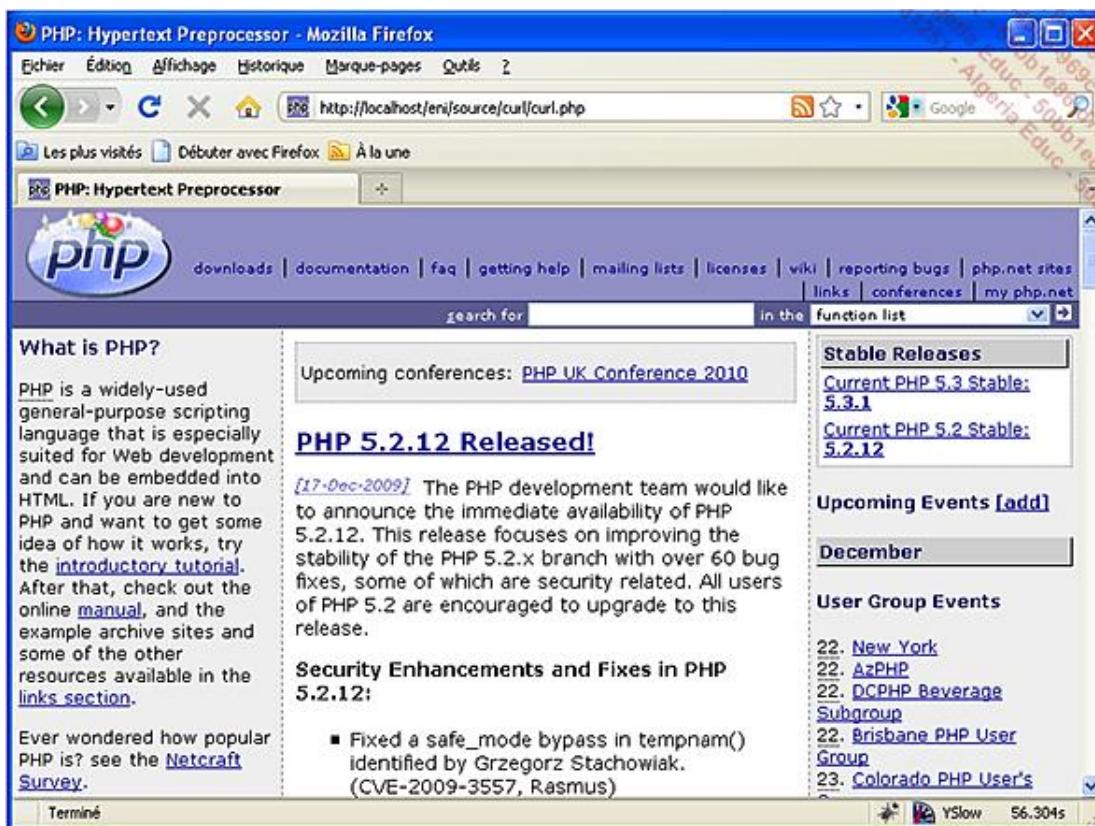
if(curl_errno($chemin))
    echo 'Erreur Curl : ' . curl_error($chemin);
curl_close($chemin);
print_r($data);
?>

```

L'exemple ci-dessus se décompose ainsi :

- initialisation vers le site Internet
- définition des données à retourner
- exécution
- affichage des erreurs si un problème apparaît
- fermeture de la connexion

Nous affichons le contenu brut des données que nous avons réceptionnées comme ceci :



## 2. Mise en pratique

Dans notre application, nous pouvons utiliser cette extension pour valider les liens Internet que nous enregistrons dans notre carnet d'adresses.

Pour réaliser cette opération, nous utilisons plusieurs options de « curl\_setopt » qui nous sont proposées sur le site php.net. Les options retenues sont :

- **CURLOPT\_URL** : l'URL à récupérer.
- **CURLOPT\_HEADER** : true pour inclure l'en-tête dans la valeur de retour.

- **CURLOPT\_NOBODY** : true pour que le corps du transfert ne soit pas inclus dans la valeur de retour.
- **CURLOPT\_RETURNTRANSFER** : true retourne directement le transfert sous la forme d'une chaîne de la valeur renvoyée par curl\_exec().

L'autre fonction utile pour notre validation des liens Internet est la suivante :

**Curl\_getinfo()** : lit les informations détaillant un transfert cURL.

Nous mettons dans une fonction le test de validation des liens internet :

Fichier : include/fct.php

```
<?php
function url_test($url)
{
    $chemin = curl_init();
    curl_setopt($chemin, CURLOPT_URL, $url);
    curl_setopt($chemin, CURLOPT_HEADER, true);
    curl_setopt($chemin, CURLOPT_NOBODY, true);
    curl_setopt($chemin, CURLOPT_RETURNTRANSFER, true);
    curl_exec($chemin);
    $ret = curl_getinfo($chemin, CURLINFO_HTTP_CODE);
    curl_close($chemin);

    if ($ret==200)
        return "URL oki";
    else
        return "";
}

?>
```

pour obtenir le résultat suivant :

The screenshot shows a Mozilla Firefox browser window with the address bar set to `http://localhost/...tizspj0lqvkhyns9`. The page content is a PHP application form. On the left, there is a sidebar with links: Accueil, Votre compte, Rubriques, Votre Carnet, Ajout, Consult/Modif, Recherche, Administration, Exportation, Deconnection, Contact. The main form area has several fields:

	Fiche :	Rubrique	Observations
Votre compte	Genre : <input type="text"/>	Site internet <input type="text"/>	www.hello-design.fr
Rubriques	Nom : <input type="text"/> VILLENEUVE	URL oki	
Votre Carnet	Prénom : <input type="text"/> Christophe	Editeur <input type="text"/>	www.editions-eni.fr
Ajout	Adresse : <input type="text"/> route de Paris		
Consult/Modif	Code postal : <input type="text"/> 75000	URL oki	
Recherche	Ville : <input type="text"/> Paris		
Administration	Email : <input type="text"/> livre@hello-design.fr		
Exportation	Téléphone : <input type="text"/>		
Deconnection	Portable : <input type="text"/>		
Contact	Photo : 		

Comme le montre notre écran, les liens existent et sont bien valides.

## Introduction

POO signifie Programmation Orientée Objet et consiste à définir l'interaction de briques logicielles appelées objets.

Cette méthode a pris plus d'importance avec le déploiement de PHP 5 et l'arrêt de PHP 4, même si la technique existe depuis de nombreuses anciennes versions, mais aujourd'hui la définition de l'ensemble des fonctions disponibles permet de les utiliser plus facilement.

La programmation orientée objet est avant tout une autre méthode de programmation par rapport à la méthode procédurale du livre et cette méthode a aussi ses avantages et ses inconvénients.

# Les bases de la programmation Objet

## 1. Généralités

La programmation objet se définit par l'utilisation des classes, déclarées par le mot « class » suivi par le nom de la classe (il ne doit pas s'agir d'un mot réservé en PHP). Ensuite la fonction « class » se délimite par des parenthèses et se présente sous la forme suivante :

```
class NomClasse
{
    var $variable;           // variable sans valeur
    var $variable2='exemple'; // variable avec valeur
}
```

Dans une classe, il est possible d'insérer une ou plusieurs fonctions permettant d'avoir les mêmes possibilités qu'une autre méthode de programmation.

Par ailleurs, l'utilisation de variables se fait avec la fonction « this », suivie d'un tiret « - » puis du signe supérieur « > » ce qui signifie qu'elle appelle le contenu d'une variable de la classe proprement dite.

Nous allons présenter sous la forme d'un script PHP la décomposition d'une classe avec l'ensemble des possibilités évoquées.

Dans un premier temps, nous devons définir la classe comprenant une fonction qui va afficher le contenu d'une variable.

Poo/class.php

```
<?php
class MaClasse
{
    public $nom = 'Les éditions ENI';
    public function afficheNom()
    {
        echo $this->nom;
    }
?>
```

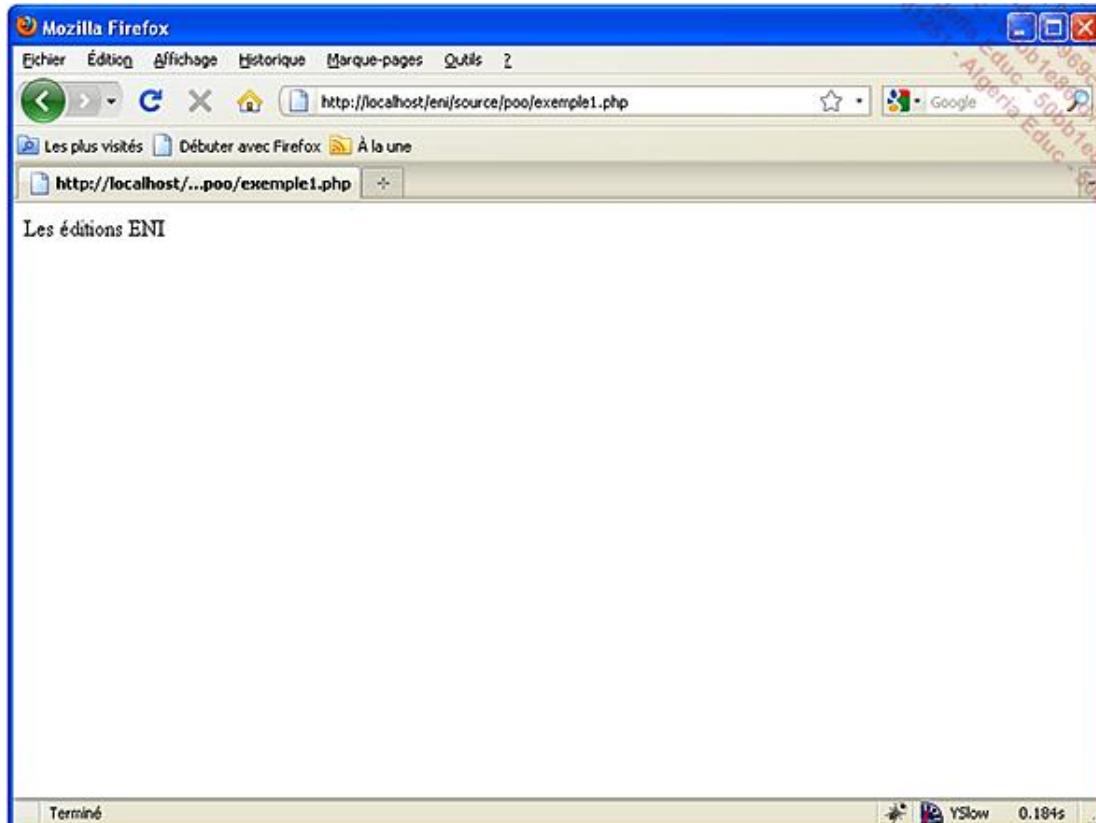
Lorsque notre classe est définie, nous devons l'appeler pour en afficher le résultat.

Poo/exemple1.php

```
<?php
$MonObject=new MaClasse();
$MonObject->afficheNom();
unset($MonObject) ;
?>
```

La première ligne déclare une nouvelle classe et la deuxième ligne appelle la fonction « afficheNom » de la classe. La ligne suivante libère notre classe dont nous n'avons plus besoin.

Voici le résultat obtenu :



## 2. Constructeur et Destructeur

### a. Constructeur

Une particularité peut exister dans une classe. Il s'agit d'un constructeur de la classe qui s'écrit de la façon suivante :

```
__construct()
```

Lorsqu'un constructeur existe dans une classe, il est automatiquement appelé lors de la création de l'objet. Cette méthode permet d'initialiser l'objet lors de sa création. Cela signifie que dès que vous exécutez une classe, la méthode `__construct` sera exécutée avant toute autre chose, par exemple, les paramètres de connexions à une base de données.

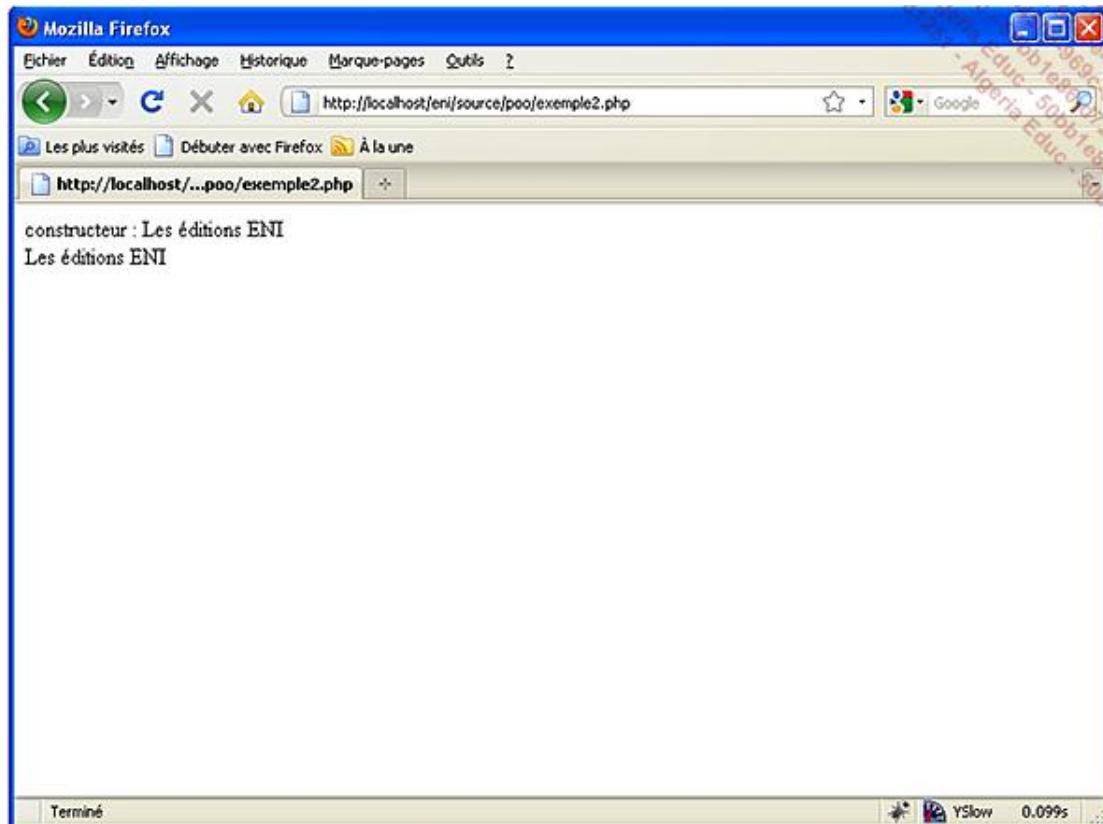
Nous pouvons écrire :

Poo/exemple2.php

```
<?php
class MaClasse
{
    var $nom;
    function __construct($nom)
    {
        $this->nom=$nom;
        echo "constructeur : ".$this->nom."<br>";
    }
}

$MonObject=new MaClasse('Les éditions ENI');
echo $MonObject->nom."<br />";
?>
```

pour obtenir le résultat suivant :



## b. Destructeur

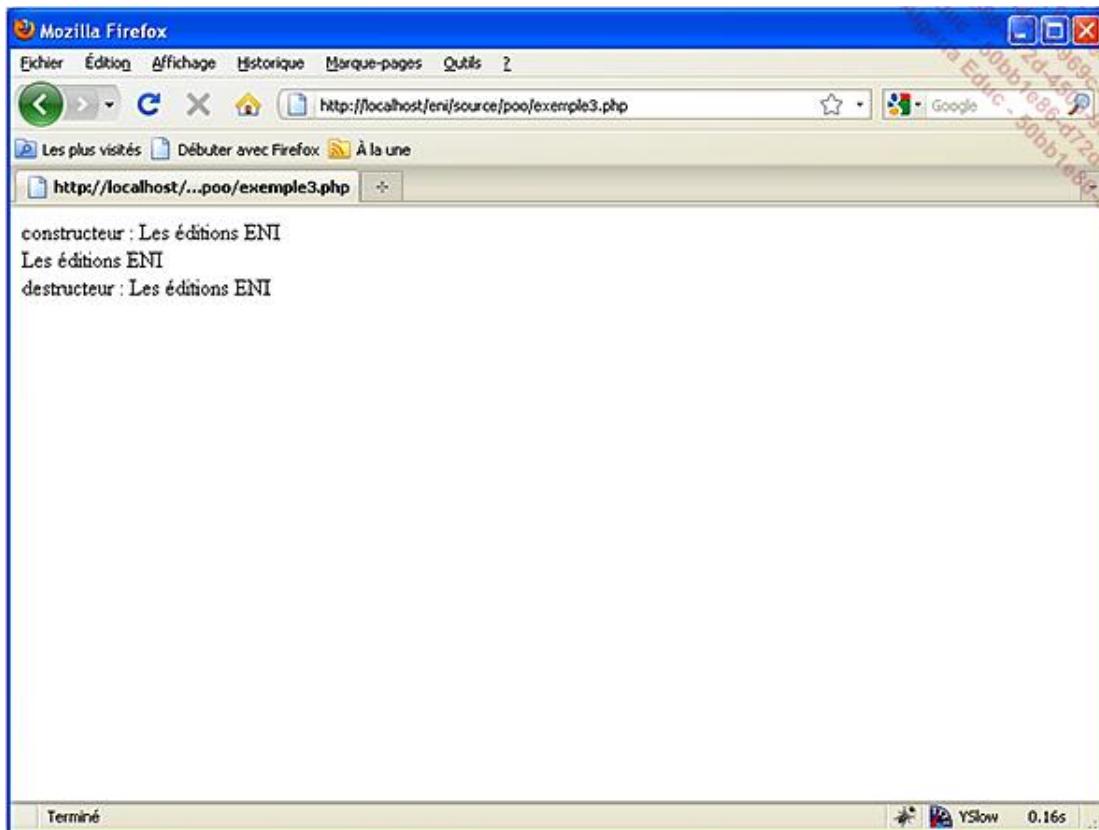
Le destructeur permet de détruire l'instance d'une classe. Il possède les mêmes caractéristiques qu'un constructeur car cette fonction est automatiquement exécutée.

Elle se présente de la façon suivante :

Poo/exemple3.php

```
<?php
function __destruct()
{
    echo "destructeur : ".$this->nom."  
>";
    unset ($this->nom);
}
?>
```

pour obtenir le résultat suivant :



### 3. Base de données

Dans une classe, nous pouvons aussi communiquer avec une base de données.

Nous allons effectuer l'opération en communiquant avec notre base de données au format MySQLi. Pour cela, nous choisissons le multifichier, permettant une meilleure maintenance de la programmation.

Nous créons notre fichier comprenant la classe avec l'ensemble des fonctions de connexion.

---

 La description des fonctions de connexions avec une base de données est identique à ce qui a été vu dans les chapitres précédents.

---

Tout d'abord, nous déclarons nos variables de connexion et notre constructeur.

Poo/maclasse.class.php

```
<?php
class CnxBDD
{
    var $host;
    var $login;
    var $password;
    var $base;

    public function __construct($serveur='$serveur', $user='$user',
        $password='$password', $bdd='$bdd')
    {
        $this->host=$serveur;
        $this->login=$user;
        $this->password=$password;
        $this->base=$bdd;
    }
}
```

Nous préparons la fonction de connexion comme une utilisation classique avec retour du bon déroulement dans \$this.

```

<?php
function connect()
{
$connect = mysqli_connect ($this->host, $this->login,
$this->password,$this->base);

if (mysqli_connect_errno())
    die ("Echec de la connexion : ". mysqli_connect_error());

$this->connect = $connect;
}
?>

```

Nous créons une nouvelle fonction permettant de sélectionner une table. Nous ne mettons pas à cet endroit la requête en dur car une classe est avant tout une suite de fonctions qui peuvent être utilisées à différents endroits d'un projet.

```

<?php
function requete ($sql)
{
$result = mysqli_query($this->connect,$sql);
if (!$result)
    die ("Probleme requete : ".$sql);

while ($Row = mysqli_fetch_object( $result ))
{
    $Rows[ ] = $Row;
}

return $Rows;
}
?>

```

Lorsque la requête s'exécute, nous enregistrons les valeurs dans une variable qui sera renvoyée directement à la page web pour traiter les données.

Il est important de penser à fermer la connexion :

```

<?php
function disconnect()
{
    mysqli_close($this->connect);
}

?>

```

Maintenant que notre classe a été créée, nous réalisons la page web. Nous chargeons les critères de connexion dans notre classe de la façon suivante :

Poo/exemple4.php

```

<?php
$serveur='localhost';
$login= 'root';
$password= '';
$bdd='ouvrage';

require_once ( 'maclasse.php' );
?>

```

Nous appelons une nouvelle classe et nous nous connectons :

```

<?php
try
{
    $CnxBDD = new CnxBDD($serveur,$login,$password,$bdd);
}
?>

```

```
$CnxBDD->connect();
?>
```

Nous envoyons notre requête SQL à notre fonction qui se trouve dans la classe. Lors de la procédure d'exécution, nous recevons le résultat de la requête. Nous affichons le contenu brut pour voir les données que nous recevons de notre classe.

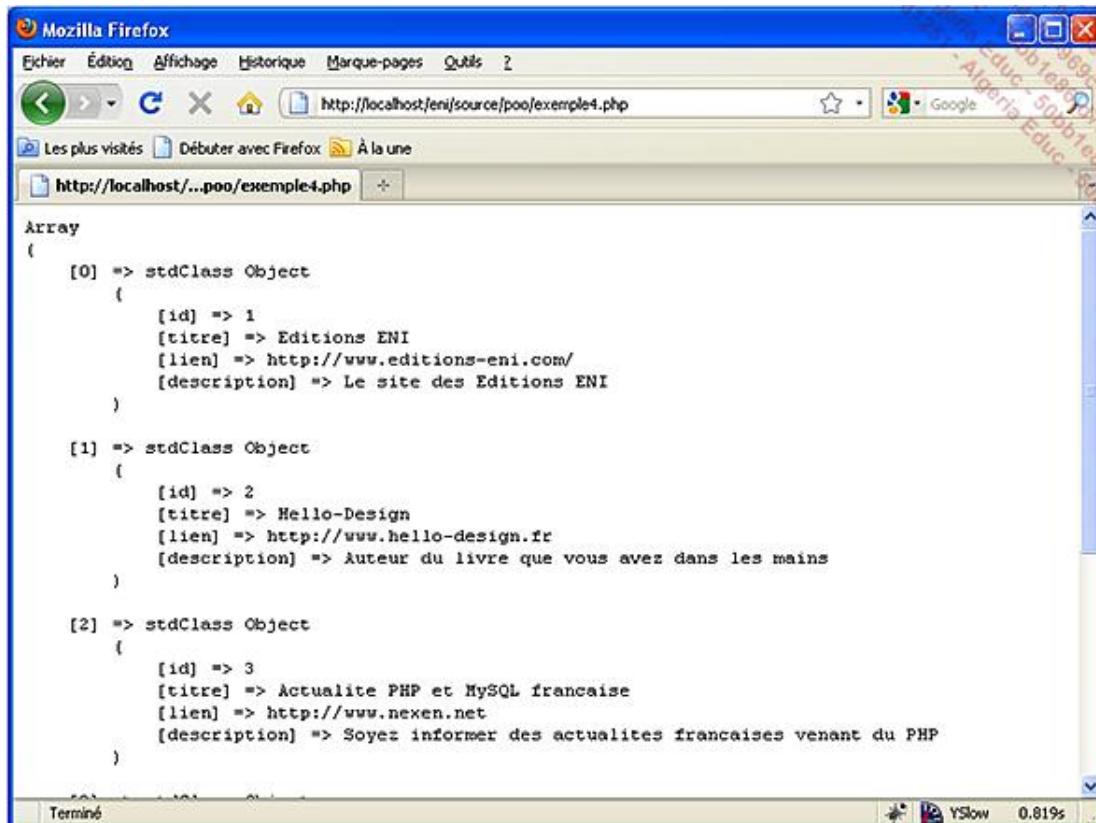
```
<?php
$row=$CnxBDD->requete ("SELECT * FROM actualite");

echo" <pre>";
print_r($row);
echo "</pre>";
?>
```

Puis nous nous déconnectons et libérons la classe :

```
<?php
$CnxBDD->deconnect();
unset ($CnxBDD);
}
catch (MySQLException $e)
{
die ('Erreur : '.$e->getMessage().'<br />');
}
?>
```

Le déroulement de cette procédure montre la connexion vers une base de données. Le résultat obtenu se présente de la façon suivante :



# Espaces de noms

## 1. Introduction

Les espaces de noms, appelés « Namespace », sont disponibles depuis PHP 5.3.

L'utilisation des espaces de noms permet d'organiser logiquement les classes en modules. L'énorme avantage est qu'ils permettent d'éviter des conflits entre les noms des classes et surtout les noms à rallonge ou les doublons de classes.

Le signe utilisé pour les espaces de noms en PHP est \ (prononcez "backslash") et se traduit de la façon suivante :

```
<?php
namespace nom\nom
?>
```

## 2. Généralités

L'utilisation des espaces de noms sous cette forme permet de résoudre les problèmes d'alias et offre la possibilité d'encapsuler l'ensemble des classes d'une bibliothèque dans un espace de noms.

### a. Simple combinaison

Pour illustrer ces quelques lignes théoriques, nous devons créer deux fichiers :

- un fichier contenant les namespaces,
- un fichier PHP d'exécution.

Tout d'abord, nous déclarons dans un premier fichier une balise namespace que nous appellerons « ESPACEdeNOM ». Cette balise est très importante car toutes les nouvelles déclarations seront placées dans le namespace « ESPACEdeNOM ».

Namespace/lib-namespace.php

```
<?php
namespace ESPACEdeNOM;
?>
```

Nous déclarons deux nouvelles fonctions qui nous serviront d'exemple :

```
<?php
function nom()
{
    echo 'ESPACEdeNOM\nom';
}

function prenom()
{
    echo 'ESPACEdeNOM\prenom';
}
?>
```

Pour concevoir le deuxième fichier, qui appellera le fichier namespace, nous effectuons l'opération suivante :

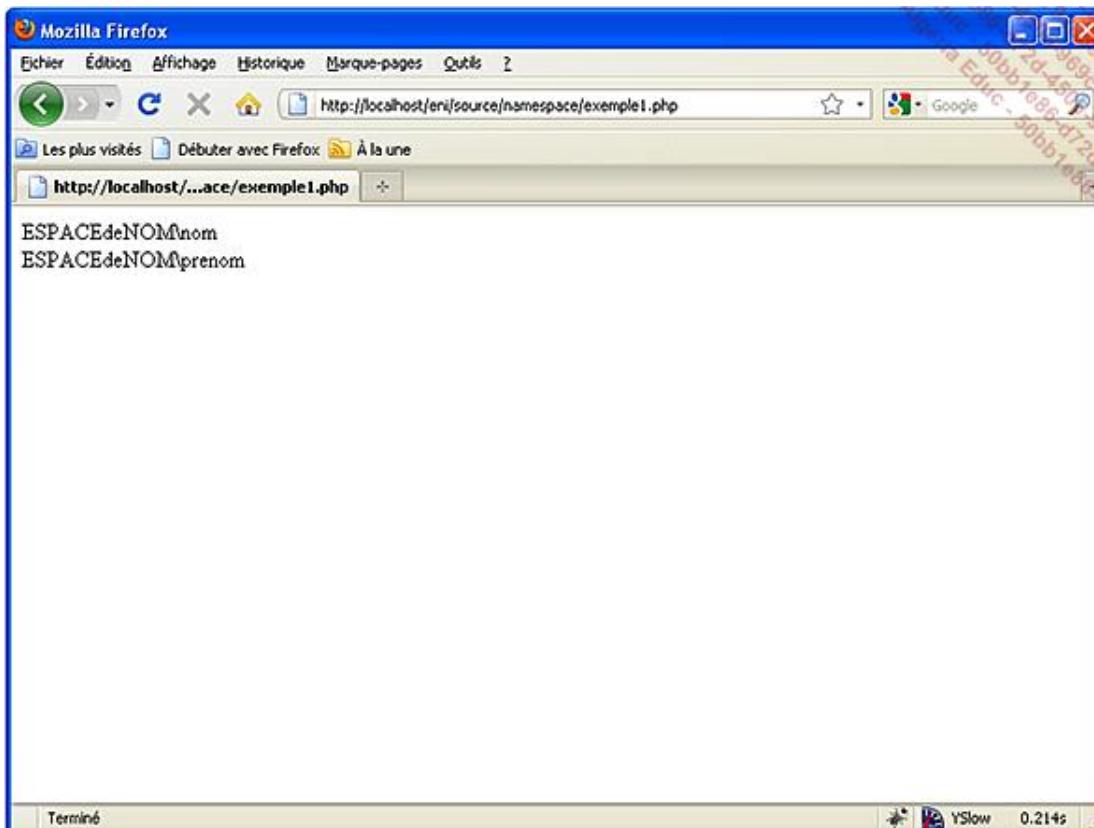
Namespace/exemple1.php

```
<?php
require_once('lib-namespace.php');
```

```
ESPACEdeNOM\nom();
echo "<br />";
ESPACEdeNOM\prenom();
?>
```

Ce script charge la déclaration des « namespaces » et affiche le contenu des deux fonctions.

Nous obtenons le résultat suivant :



En résumé, notre espace de nom agit comme un nouveau type de conteneur et l'opérateur \ est utilisé pour séparer le nom de l'espace de nom des noms de fonctions qu'il contient.

## b. Multicombinaisons

Nous pouvons déclarer plusieurs espaces de noms différents. Chaque fichier ne peut contenir qu'un seul espace de nom, mais nous pouvons avoir plusieurs fichiers pour un espace de noms.

Tout ce qui suit après l'espace de noms est associé dans cet espace jusqu'à la fin du fichier.

Nous créons un nouveau fichier contenu le nouvel espace de noms. La représentation se présente de la façon suivante :

Namespace/lib-namespace2.php

```
<?php
namespace PRODUIT;

function nom()
{
    echo 'PRODUIT\nom';
}
?>
```

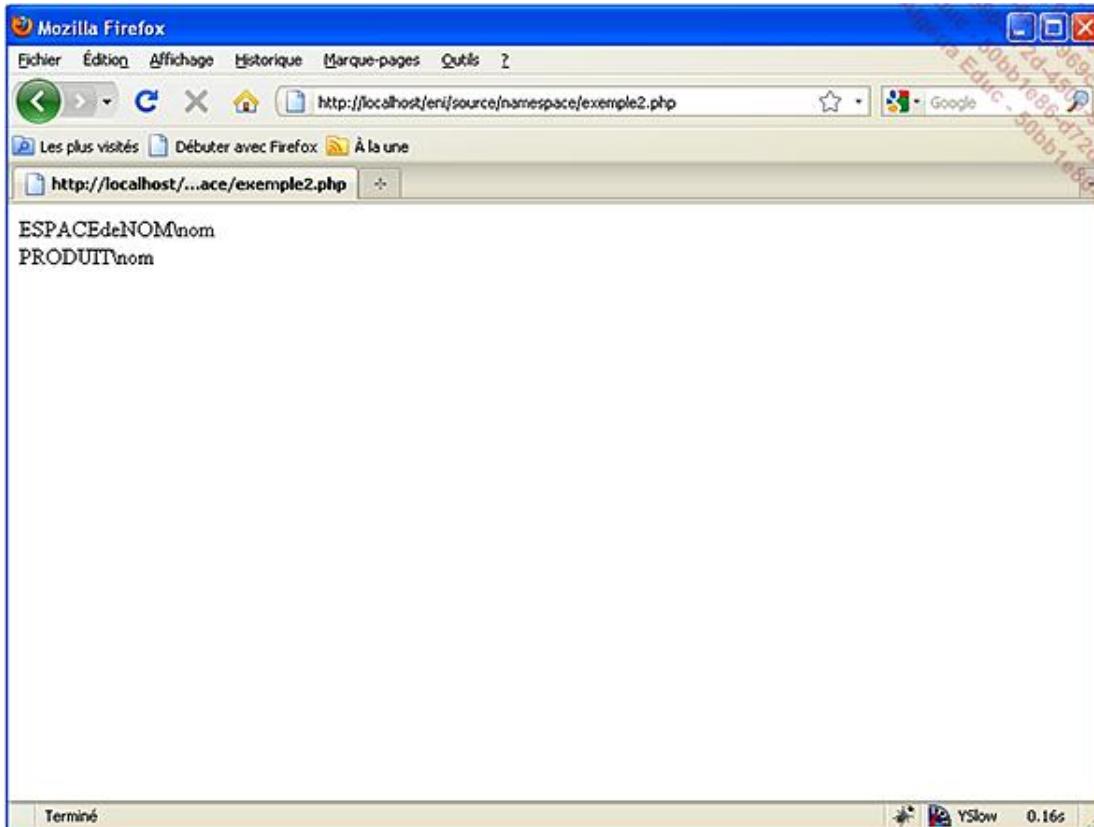
Nous créons notre fichier qui appellera les deux fichiers contenant les espaces de noms pour demander les fonctions.

Namespace/exemple2.php

```
<?php
require_once('lib-namespace.php');
require_once('lib-namespace2.php');

ESPACEdeNOM\nom();
echo "<br />";
PRODUIT\nom();
?>
```

Nous obtenons le résultat suivant :



### 3. Les classes

L'utilisation des espaces de noms va permettre d'utiliser la programmation objet en employant les classes.

Pour cela, nous plaçons avant une classe la déclaration des espaces de noms.

Nous réutilisons le script `poo/exemple1.php` pour obtenir le nouveau script PHP suivant :

`Namespace/lib-namespace3.php`

```
<?php
namespace ESPACEdeNOM;

class MaClasse
{
    public $nom = 'Les éditions ENI';

    public function afficheNom()
    {
        echo $this->nom;
    }
}?
?>
```

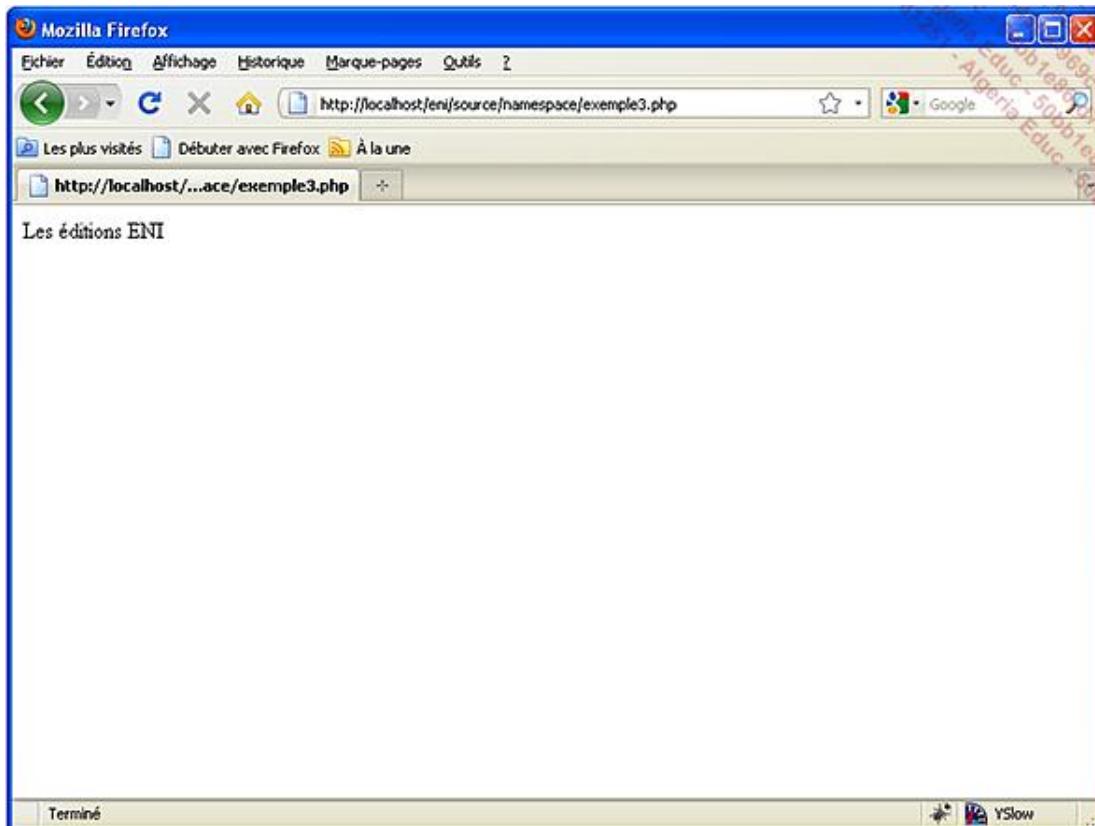
Pour appeler cette classe, notre script se présente désormais de la façon suivante :

Namespace/exemple3.php

```
<?php
require_once('lib-namespace3.php');
$row = new ESPACEdeNOM\MaClasse;
$row->afficheNom();
unset ($row);
?>
```

Nous retrouvons dans ce script exactement les mêmes fonctionnalités que dans les pages précédentes.

Nous obtenons le résultat suivant :



Cependant, lorsque nous manipulons les espaces de noms, les classes, les fonctions, il est parfois difficile de se situer.

Aussi le langage PHP propose des mots-clefs qui retournent des informations :

Namespace/lib-namespace4.php

```
<?php
namespace ESPACEdeNOM;

class MaClasse
{
    var $retour;           //retour de données

    function position()
    {
        $retour = "";
        $retour .= "Classe : ".__CLASS__."
        $retour .= "Namespace : ".__NAMESPACE__."
        $retour .= "Function : ".__FUNCTION__."
        return $retour;
    }
}
```

?>

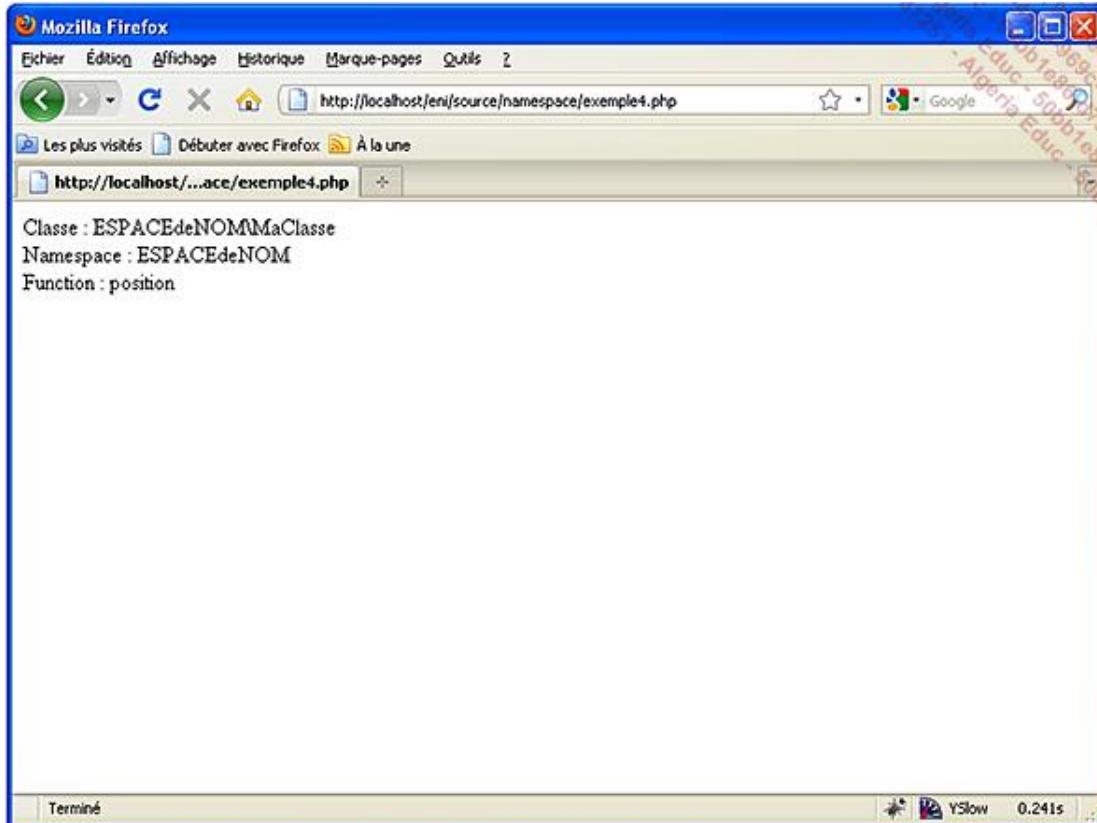
La fonction ci-dessus doit retourner le nom de la classe, puis le nom des espaces de noms et pour finir le nom de la fonction utilisée.

Nous exécutons cette classe en gardant le même principe que précédemment, de la façon suivante :

Namespace/exemple4.php

```
<?php
require_once('lib-namespace4.php');
$row = new ESPACEdeNOM\MaClasse;
echo $row->position();
unset ($row);
?>
```

Pour obtenir le résultat suivant :



# Cas pratique

## 1. Sujet

Notre cas pratique reprend le même thème que le reste du livre, c'est-à-dire la gestion d'un carnet d'adresses avec une seule base de données.

Nous allons voir comment utiliser la programmation objet pour deux fonctionnalités du carnet d'adresses, c'est-à-dire :

- l'identification
- la gestion de l'administration

Les autres points étudiés comme la gestion du carnet d'adresses, les différentes exportations, restent sur le même principe que ce qui sera étudié juste après.

Nous allons utiliser la programmation objet avec une base de données MySQLi et en employant les « espaces de noms ». Par conséquent, nous devons posséder PHP 5.3.

 Il est possible d'utiliser l'objet avec une autre version de PHP 5 en supprimant la notion de namespace.

## 2. La classe

La création de la classe est la partie majeure de l'application, car elle va contenir l'ensemble des fonctions que nous allons utiliser pour notre application.

Nous déclarons la classe de la façon suivante :

Fichier : include/class.class.php

```
<?php
namespace ESPACEdeNOM;

class cnxBDD
{
    var $host;
    var $login;
    var $password;
    var $base;

    public function __construct($serveur='$serveur', $user='$user',
    $password='$passwd', $bdd='$bdd')
    {
        $this->host=$serveur;
        $this->login=$user;
        $this->password=$password;
        $this->base=$bdd;
    }

    function __destruct()
    {
        unset ($this->connect);
    }
}
?>
```

Notre classe contient la déclaration des espaces de noms, du conteneur et du destructeur, différents points qui ont été étudiés dans les pages précédentes.

Nous préparons la fonction qui sera utile pour la connexion à la base de données. Nous utilisons les fonctions try/catch permettant de récupérer les erreurs éventuelles.

Lorsque la connexion avec la base de données est effectuée, nous renvoyons l'information pour exécuter la requête.

Le script PHP se présente de la façon suivante :

```
<?php
function connect()
{
try
{
$connect = mysqli_connect ($this->host, $this->login,
$this->password,$this->base);

if (mysqli_connect_errno())
die ("Echec de la connexion : ". mysqli_connect_error());

$this->connect = $connect;
}

catch (PDOException $error)
{
echo 'Nº : '.$error->getCode(). '<br />';
die ('Erreur : '.$error->getMessage(). '<br />');
}
}
?>
```

Nous devons ensuite préparer deux requêtes :

- une requête pour la sélection de données,
- une requête pour la manipulation de données.

Une requête pour la sélection de données est utile pour connaître le nombre de lignes d'enregistrements, mais aussi pour afficher le contenu provenant de la base de données.

Des explications complémentaires seront apportées dans les pages qui vont suivre.

La fonction requête de sélection utilise l'attribut de la connexion qui a été ouverte précédemment et exécute la requête SQL qui a été envoyée.

Pour vérifier si celle-ci possède au moins une ligne de résultat, nous utilisons la fonction num\_rows.

Si aucun résultat n'est présent, la fonction nous renvoie une information fausse.

Si la requête retourne des données, celles-ci seront stockées dans un tableau.

La boucle pour mémoriser la fonction est utile car si le résultat contient plusieurs lignes de données, nous pouvons les traiter. Sans cette opération, nous serions obligés de créer une fonction supplémentaire.

Le script PHP sera le suivant :

```
<?php
function requeteSelect ($sql)
{
try
{
$result = $this->connect->query($sql);

if (!$result)
die ("Probleme requete : ".$sql);

if ($result->num_rows==0 )
{
    $rows=0;
}
else
{
    while ($row = $result->fetch_object())
```

```

    {
        $rows[] = $row;
    }
}
return $rows;
}
catch (PDOException $error)
{
echo 'Nº : '.$error->getCode().'  
>';
die ('Erreur : '.$error->getMessage().'  
>');
}
}
?>

```

L'autre requête va permettre d'insérer, de mettre à jour, de supprimer des données. Ces opérations utilisent des fonctions différentes.

La fonction requête utilise l'attribut de la connexion qui a été ouverte précédemment et exécute la requête SQL qui a été envoyée.

L'exécution de la requête se déroule en deux temps : nous préparons d'abord la requête pour ensuite l'exécuter. Cette opération a été décrite dans les chapitres précédents.

Le script PHP est le suivant :

```

<?php
function requeteOther ($sql)
{
try
{
    $qid=$this->connect->prepare($sql);
    $qid->execute();
}
catch (PDOException $error)
{
    echo 'Nº : '.$error->getCode().'  
>';
    die ('Erreur : '.$error->getMessage().'  
>');
}
}
?>

```

La fonction try/catch est utilisée pour permettre de récupérer les problèmes éventuels.

Pour finir notre classe, nous nous déconnectons :

```

<?php
function disconnect()
{
    mysqli_close($this->connect);
}
?>

```

### 3. L'identification

L'identification est le premier écran vers la connexion et la gestion de l'administration.

L'écran de saisie d'identification reste identique à celui du chapitre La saisie et l'affichage - Connexion à un compte, car il s'agit d'une partie HTML. Nous vous conseillons de relire celle-ci pour connaître les différents tests des champs de saisie.

Lorsque les champs ont été correctement renseignés, nous exécutons le fichier login.inc.php qui va nous permettre, si la saisie est correcte, l'accès à notre compte.

La partie validation a été étudiée dans le chapitre La saisie et l'affichage - La connexion. Nous utilisons seulement la classe puisqu'elle a déjà été déclarée un peu plus haut.

Nous déclarons l'espace de noms et la nouvelle classe en envoyant les paramètres d'identification vers la base de données, pour ensuite nous connecter.

Le script PHP se présente de la façon suivante :

Fichier login.inc.php

```
<?php
$cnx = new ESPACEdeNOM\cnxBDD($serveur,$user,$passwd,$bdd);
$cnx->connect();
?>
```

Tout d'abord, nous contrôlons si les identifiants sont bien uniques. Pour cela, nous utilisons la connexion qui a été validée et exécutons la requête de sélection.

La valeur qui sera retournée nous permettra de savoir si le compte existe bien dans notre base de données.

Le script PHP se présente de la façon suivante :

```
<?php
$sql="SELECT COUNT(*) as nbre FROM user WHERE login='$login' AND
password='$password' ";
$nbre=$cnx->requeteSelect($sql);
if ($nbre==0)    header("Location:identification.php?erreur=login");
?>
```

Tout en gardant notre connexion ouverte, nous pouvons demander à notre classe de mettre à jour la clef de notre compte et la date de notre passage.

Pour cela, nous utilisons l'autre requête. Cette fonction va exécuter la requête SQL qui sera envoyée comme ceci :

```
<?php
$idclef=recup_clef();
$date=DATE("Y-m-d");
$sql="UPDATE user SET idclef='$idclef',date_lastpass='$date' WHERE
login='$login' AND password='$password' ";
$cnx->requeteOther($sql);
?>
```

La dernière étape concerne la récupération des données. Cette opération utilise la même fonction de sélection.

Étant donné que le résultat renvoyé comportera une seule ligne, nous utilisons [0] pour être certain de prendre les bonnes valeurs :

```
<?php
$sql="SELECT * FROM user WHERE login='$login' AND password='$password' AND
idclef='$idclef' ";
$row=$cnx->requeteSelect($sql);
if ($row[0]->niveau)
{
    session_start();
    $_SESSION['login'] = $row[0]->login;
    $_SESSION['idclef'] = $row[0]->idclef;
    $_SESSION['niveau'] = $row[0]->niveau;
    $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
    $_SESSION['iduser'] = $row[0]->id;
    $destination=$row[0]->page;
    header("Location:$destination");
}
?>
```

## 4. La gestion de l'administration

La gestion de l'administration est accessible pour les personnes qui possèdent les droits d'administration.

Cette partie a été détaillée dans le chapitre La saisie et l'affichage - La gestion de l'administration.

## a. Ajouter un compte

Le formulaire permettant l'ajout d'un nouveau compte reste identique à ce qui a été déjà vu dans le chapitre La saisie et l'affichage.

Par contre, lors de la confirmation de la saisie, nous nous connectons à la classe pour insérer la nouvelle personne dans la base de données.

Nous déclarons notre classe et nous nous connectons :

Fichier : admin-add.php

```
<?php
$cnx = new ESPACEdeNOM\cnxBDD($serveur,$user,$passwd,$bdd);
$cnx->connect();
?>
```

L'opération est identique à celle de la partie précédente.

Concernant l'insertion des données saisies à partir du formulaire, nous envoyons simplement la requête SQL à la fonction se trouvant dans notre classe.

```
<?php
function insere_compte(&$frm)
{
global $cnx;

    $creat_compte=date("Y-m-j");

    $sql = "
        INSERT INTO user (
            'niveau'
            , 'login'
            , 'password'
            , 'email'
            , 'date_creation'
            , 'page'
        ) VALUES (
            '$frm[niveau]'
            , '".$htmlentities($frm['nlogin'])."'
            , '".$md5($frm['npassword'])."'
            , '".$htmlentities($frm['email'])."'
            , '$creat_compte'
            , 'compte.php'
        )";
$cnx->requeteOther($sql);

}
?>
```

Le script ci-dessus ne possède aucun paramètre car c'est la classe qui effectue le travail et les différents tests.

## b. Afficher un compte

Afficher un compte reste identique du côté HTML. La partie La gestion de l'administration du chapitre La saisie et l'affichage explique ces points en détail.

La connexion à la classe est obligatoire pour pouvoir récupérer les données de la base.

Pour afficher ces données, nous devons envoyer une requête SQL à notre fonction de sélection comme ceci :

Fichier admin-view.php

```
<?php
$sql="select * from user";
$res=$cnx->requeteSelect($sql);
?>
```

La requête retourne le résultat avec l'ensemble des valeurs que nous pouvons afficher avec la fonction FOREACH de la façon suivante :

```
<?php  
  
foreach($res as $row)  
{  
    echo "<td>$row->login</td>";  
}  
?>
```

### c. Éditer un compte

L'édition d'un compte se base sur le même principe que les parties précédentes.

La manipulation des données est réalisée à partir de la classe qui possède l'ensemble des fonctions.

L'édition du compte consiste à réaliser une requête SQL correcte et à l'envoyer auprès de la fonction de la classe.

Le script PHP est disponible en téléchargement car les différentes fonctions PHP ont déjà été étudiées.

# Liens utiles

L'internet est une source importante de documentation pour le langage PHP. Il existe de nombreuses aides, solutions et forums couvrant l'ensemble des problèmes que nous pouvons rencontrer.

Vous trouverez ci-après une liste de sites, n'hésitez pas à les consulter régulièrement car ce sont des sites très actifs. Cette liste est non exhaustive mais propose les sites les plus importants et les plus actifs.

## 1. Liens français

### Incontournables

PHP : <http://fr.php.net>

El Roubio : <http://www.elroubio.net/>

### Association & rendez-vous

AFUP : <http://www.afup.org>

PHP Quebec : <http://www.phpquebec.org>

PHP Apero : <http://www.aperophp.net/>

### Actualité, agrégateur de nouvelles et articles

Nexen : <http://www.nexen.net>

PHP Index : <http://www.phpindex.com>

PHP Solutions : <http://phpsolmag.org/fr>

Planete PHP : <http://www.planete-php.fr>

### Tutoriaux, Forum

PHP Team : <http://www.phpteam.net>

PHP France : <http://www.phpfrance.com>

PHP Debutant : <http://www.phpdebutant.org>

Developpez : <http://php.developpez.com/>

ASP-PHP : <http://www.asp-php.net>

PHP Portail : <http://www.phportail.net>

### Sources, scripts

PHP Scripts : <http://www.phpscripts-fr.net>

Comscripts : <http://www.comscripts.com/>

### Police de caractères

Font temple : <http://www.fonttemple.com/>

Dafont : <http://www.dafont.com/fr/>

## 2. Liens Anglophones

### Actualité, agrégateur de nouvelles

PHP Developpeur : <http://www.phpdeveloper.org/>

Planet PHP : <http://www.planet-php.org>

### Tutoriaux

PHP Coding pratices : <http://php-coding-practices.com/>

### Sources, scripts, extensions

Pear : <http://pear.php.net>

Pecl : <http://pecl.php.net>

PHP Builder : <http://www.phpbuilder.com>

HotScripts : <http://www.hotscripts.com>