

# PHP 5 avancé

## 4<sup>e</sup> édition

**Éric Daspét**

**Cyril Pierre de Geyer**

Préface de Damien Seguy



# Préface

---

L'une des grandes forces de PHP est sa facilité de prise en main. Il se révèle pratique à mettre en œuvre grâce à sa documentation de référence et aux nombreuses applications disponibles.

PHP est un langage didactique car il ne masque pas la complexité et pousse l'utilisateur à comprendre et à appliquer les standards. À l'inverse, d'autres technologies encadrent beaucoup l'utilisateur et l'abandonnent lorsque les problèmes deviennent plus complexes.

PHP distille progressivement les technologies qu'il exploite, et donne toujours au programmeur la possibilité d'aller plus loin.

La maîtrise du langage (on pourrait même parler de plate-forme) requiert donc un apprentissage permanent, qui va de pair avec l'utilisation de PHP. Pour aller plus vite avec PHP, il faut expérimenter, ou profiter de l'expérience des autres. C'est dans ce sens que *PHP 5 avancé* a été pensé : il est fait pour ceux qui veulent aller plus loin, et plus vite.

*PHP 5 avancé* est un livre à garder à côté de son clavier. Contrairement aux autres livres pédagogiques, il propose un panorama très large du langage. Il fournit des méthodes pour chaque aspect de la programmation.

Bien sûr, on y retrouve tout ce qui a fait le succès de PHP 4, mais le livre se concentre surtout sur les nouveautés introduites en PHP 5. Cette nouvelle évolution du langage introduit la programmation objet moderne, simplifie le XML, met sur le devant de la scène SQLite et élève le niveau de programmation en général. PHP 5 est aussi la preuve que les projets Open Source peuvent rivaliser avec les éditeurs propriétaires, en termes de niveau de fonctionnalités, de sécurité, de fiabilité et au niveau théorique.

Ce livre est résolument tourné vers les informaticiens qui veulent aller plus loin avec PHP et ne jamais manquer de ressources pour toutes leurs applications Web. Il sert de référence à tous ceux qui veulent intelligemment tirer le meilleur de la technologie.

Damien SEGUY  
*Responsable de la documentation PHP française*  
*Vice-président AFUP*  
*Webmestre Nexen.net*



# Table des matières

---

<b>Avant-propos</b> .....	XXXI
<b>Pourquoi ce livre ?</b> .....	XXXI
<b>Structure de l'ouvrage</b> .....	XXXII
<b>Remerciements</b> .....	XXXIV
CHAPITRE 1	
<b>Qu'est-ce que PHP ?</b> .....	1
<b>Introduction à PHP</b> .....	1
Un langage Open Source .....	1
Que faire avec PHP ? .....	3
Particularités de PHP .....	4
Historique .....	6
Mode de développement du projet PHP .....	8
<b>Nouveautés de PHP 5</b> .....	10
La programmation orientée objet .....	10
Refonte et simplification de XML .....	10
Intégration de la base SQLite .....	10
Simplification des tâches courantes .....	11
PDO : socle commun aux SGBD .....	11
<b>Architecture et fonctionnement</b> .....	11
Architecture technique .....	11
Fonctionnement de PHP .....	12

<b>PHP en France et dans le monde</b> .....	14
Les chiffres d'utilisation en France .....	14
La communauté française .....	15
Les ressources d'aide francophones .....	16
Les ressources d'aide anglophones .....	23
CHAPITRE 2	
<b>Installer et configurer PHP 5</b> .....	27
<b>Migration vers PHP 5</b> .....	27
Incompatibilités .....	28
PHP en ligne de commande et en CGI .....	28
<b>Modes d'installation</b> .....	28
CGI .....	28
Modules .....	29
<b>Installer PHP 5 sous MS-Windows</b> .....	29
Installation automatique .....	29
Installation manuelle .....	30
<b>Installer PHP 5 sous Unix</b> .....	38
Utilisation automatisée .....	38
Installation manuelle d'Apache .....	39
Installation manuelle de MySQL .....	40
Installation manuelle de PHP .....	41
Gestion des droits d'accès .....	44
Modules additionnels PECL .....	44
<b>Configuration de PHP avec php.ini</b> .....	45
Utilisation des modules et des extensions .....	45
Les directives de configuration .....	46
Gestion de la configuration .....	50
CHAPITRE 3	
<b>Les structures de base</b> .....	53
<b>Insertion de PHP dans HTML</b> .....	53
Balises d'ouverture et de fermeture .....	54
Les commentaires .....	54

Enchaînement des instructions . . . . .	55
Structure du document . . . . .	56
Exécuter du code PHP . . . . .	57
<b>Constantes et variables</b> . . . . .	59
Variables . . . . .	59
Constantes . . . . .	63
<b>Types de données</b> . . . . .	64
Booléens (boolean). . . . .	65
Les nombres entiers (integer). . . . .	65
Les nombres flottants (double, float) . . . . .	66
Les chaînes de caractères (string). . . . .	66
Les tableaux (array) . . . . .	71
Transtypage . . . . .	76
CHAPITRE 4	
<b>Traitements de base</b> . . . . .	79
<b>Les opérateurs</b> . . . . .	79
Opérateurs d'affectation . . . . .	79
Opérateurs arithmétiques . . . . .	81
Opérateurs combinés . . . . .	83
La concaténation. . . . .	84
Opérateurs de comparaison . . . . .	84
Opérateurs logiques . . . . .	86
Opérateurs sur les bits . . . . .	86
Priorités entre opérateurs . . . . .	87
<b>Structures de contrôle</b> . . . . .	88
Les conditions. . . . .	88
Les boucles . . . . .	93
Les instructions d'arrêt. . . . .	98
<b>Les fonctions utilisateurs</b> . . . . .	99
Déclaration d'une fonction. . . . .	99
Appel de fonction. . . . .	100
Visibilité des variables . . . . .	101
Retourner plusieurs valeurs . . . . .	103
Nombre de paramètres indéfini . . . . .	104

<b>Inclure des bibliothèques ou des fichiers</b> .....	105
Différence entre require() et include() .....	106
require_once() et include_once() .....	106
CHAPITRE 5	
<b>Traitements de chaînes</b> .....	107
<b>Fonctions d’affichage</b> .....	107
Affichages simples .....	107
Affichages avec masques .....	108
<b>Informations sur une chaîne</b> .....	111
Accéder à un caractère précis .....	111
Valeur ASCII d’un caractère .....	111
Taille d’une chaîne .....	111
Position d’une sous-chaîne .....	113
Présence de certains caractères .....	114
<b>Conversions et formatages</b> .....	114
Protections et échappements .....	114
Conventions d’affichage locales .....	118
Jeux de caractères .....	119
<b>Manipulations sur les chaînes</b> .....	122
Recherche d’une sous-chaîne .....	122
Récupérer une sous-chaîne .....	122
Remplacer un motif .....	122
Fonctions d’élégage .....	123
Remplissage .....	124
Changement de casse .....	124
Coupure de paragraphes .....	125
CHAPITRE 6	
<b>Utilisation des tableaux</b> .....	127
<b>Taille d’un tableau</b> .....	127
<b>Recherche d’un élément</b> .....	128
Présence dans le tableau .....	128
Recherche de la clé correspondante .....	129
Nombre d’occurrences d’un élément .....	130
Récupération aléatoire d’éléments .....	130

<b>Trier les tableaux</b> .....	131
Tri par valeur .....	131
Tri en ordre inverse .....	131
Garder les associations clé-valeur .....	132
Tri par clé .....	132
Tri naturel .....	132
Trier avec une fonction utilisateur .....	133
Tri multicritère .....	134
<b>Extractions et remplacement</b> .....	134
Affecter des variables .....	134
Sérialisation de tableaux .....	135
Extraction d'un sous-tableau .....	135
Remplacement d'un sous-tableau .....	136
<b>Gestion des clés et des valeurs</b> .....	137
Liste des clés utilisées .....	137
Liste des valeurs utilisées .....	137
Échanger les clés et les valeurs .....	137
<b>Fusions et séparations</b> .....	138
Fusion de plusieurs tableaux .....	138
Séparation d'un tableau en plusieurs .....	139
<b>Différences et intersections</b> .....	140
Différences entre tableaux .....	140
Intersections entre deux tableaux .....	140
Gestion des doublons .....	141
<b>Gestion des piles et des files</b> .....	141
<b>Navigation dans les tableaux</b> .....	142

## CHAPITRE 7

<b>Fonctions usuelles</b> .....	143
<b>Fonction d'affichage</b> .....	143
Informations de configuration .....	143
Affichage de débogage .....	146
Coloration syntaxique de code .....	147
<b>Fonctions mathématiques</b> .....	148
Connaître les extrémités .....	148
Arrondir des valeurs .....	148



Créer des valeurs aléatoires .....	149
Travailler sur différentes bases .....	150
<b>Fonctions de date</b> .....	151
Formater une date/heure locale .....	151
<b>Fonctions réseau</b> .....	155
Résolution DNS d'une adresse IP .....	155
Corrélation IP/DNS .....	156
<b>Fonctions de chiffrement</b> .....	157
Quelques définitions : chiffrement, hachage, codage/décodage .....	157
Fonctions de hachage .....	158
Fonctions de codage et décodage .....	162
<b>Exécution de code</b> .....	163
Fonction à l'arrêt du script .....	163
Exécution d'une chaîne de code PHP .....	164
Login/mot de passe sécurisés .....	164

## CHAPITRE 8

<b>Formulaires et superglobales</b> .....	167
<b>Formulaires HTML</b> .....	167
Nouveautés depuis PHP 4.0 .....	168
<b>Caractères spéciaux et HTML</b> .....	168
<b>Création du formulaire</b> .....	169
Déclaration d'un formulaire .....	169
Méthode d'envoi du formulaire .....	170
Champ de texte .....	171
Zone de texte .....	173
Cases à cocher .....	174
Bouton radio .....	175
Liste de sélections et liste déroulante .....	176
Champs cachés .....	178
Les champs pour mot de passe .....	179
Image cliquable .....	179
Envoi d'images et de fichiers .....	179
<b>Réception des données en PHP</b> .....	180
Utilisation des superglobales .....	180
Récupération d'une donnée simple .....	181

Retours à la ligne et zones de texte . . . . .	182
Utilisation des cases à cocher . . . . .	184
Validation de données avec l'extension Filter . . . . .	184
Listes à sélections multiples . . . . .	191
Gestion des images cliquables . . . . .	192
Téléchargements d'images et de fichiers . . . . .	192
Formulaire dynamique et tableaux . . . . .	194
<b>Autres problématiques . . . . .</b>	<b>195</b>
Gestion du temps . . . . .	195
Gestion de la taille des données . . . . .	196
Stockage des fichiers temporaires . . . . .	196
Sécurité et données reçues . . . . .	196
Procédure de gestion des formulaires . . . . .	197

## CHAPITRE 9

<b>Environnement web et superglobales . . . . .</b>	<b>199</b>
<b>Descriptif du contexte Web . . . . .</b>	<b>199</b>
Client-serveur . . . . .	199
En-tête et contenu . . . . .	200
Variables superglobales . . . . .	201
<b>Informations sur le serveur . . . . .</b>	<b>201</b>
Nom du serveur . . . . .	202
Racine du serveur . . . . .	202
Autres informations sur le serveur . . . . .	203
<b>Authentification HTTP . . . . .</b>	<b>203</b>
Principes du protocole HTTP . . . . .	204
Gestion avec PHP . . . . .	204
Authentification par le serveur web . . . . .	206
<b>Paramètres de la connexion . . . . .</b>	<b>206</b>
Adresse IP et port du client . . . . .	206
Adresse IP et port du serveur . . . . .	208
<b>Description de la requête HTTP . . . . .</b>	<b>208</b>
Paramètres de la requête . . . . .	209
L'adresse demandée (URL) . . . . .	209
Informations fournies par le client . . . . .	210

<b>Environnement système</b> .....	212
Nom du script exécuté .....	212
<b>Interactions PHP/JavaScript</b> .....	213
<b>Ligne de commande</b> .....	213
Lecture des arguments .....	213
Nombre d'arguments .....	214
CHAPITRE 10	
<b>Les cookies</b> .....	215
<b>Présentation</b> .....	215
Forme du cookie sur votre ordinateur .....	216
<b>Lecture et écriture d'un cookie</b> .....	217
Envoi d'un cookie .....	217
Lecture d'un cookie .....	218
Suppression d'un cookie .....	220
Modifier les valeurs d'un cookie .....	220
<b>Validité et date d'expiration</b> .....	221
<b>Tableaux et types complexes</b> .....	222
Restriction de portée du cookie .....	223
<b>Limitations et sécurité</b> .....	225
Limitations dues aux navigateurs .....	225
Les cookies n'ont aucune sécurité .....	225
<b>Cas d'application</b> .....	226
Outil de personnalisation d'affichage .....	226
CHAPITRE 11	
<b>Les sessions</b> .....	231
<b>Qu'est-ce qu'une session ?</b> .....	231
<b>Lecture et écriture</b> .....	232
<b>Utilisation avancée</b> .....	233
Fonctionnement interne des sessions .....	234
Suppression d'une session .....	235
Définition manuelle de l'initialisation .....	235

Stockage des données de session . . . . .	236
Paramètres du cookie . . . . .	237
Accès concurrents aux sessions . . . . .	237
<b>Configuration de PHP . . . . .</b>	<b>238</b>
Initialisation des sessions . . . . .	238
Stockage des données de session . . . . .	238
Paramètres du cookie . . . . .	239
Expiration des sessions. . . . .	239
Gestion du cache . . . . .	239
Transmission de l'identifiant . . . . .	240
<b>Gestionnaires de sessions . . . . .</b>	<b>241</b>
Définir un gestionnaire personnalisé . . . . .	242
<b>Limitations et sécurité . . . . .</b>	<b>245</b>
Cachez les sessions. . . . .	245
N'utilisez pas la réécriture des liens. . . . .	245
Les identifiants par défaut suffisent . . . . .	245
Attaque par fixation de session. . . . .	246
Vérifiez l'identité de l'utilisateur . . . . .	246
N'ayez pas confiance . . . . .	246
<b>Cas d'application . . . . .</b>	<b>247</b>
Authentification par formulaire . . . . .	247
 CHAPITRE 12	
<b>Gestion des objets . . . . .</b>	<b>253</b>
<b>Introduction aux objets . . . . .</b>	<b>253</b>
Pourquoi programmer en objet ? . . . . .	253
Qu'est-ce qu'un objet ? . . . . .	254
Qu'est-ce qu'une classe ? . . . . .	254
Qu'est-ce qu'une instance ? . . . . .	255
<b>Utilisation simple des objets . . . . .</b>	<b>255</b>
Déclarer une classe . . . . .	255
Utilisation des objets . . . . .	257
Vérifier le type d'un objet . . . . .	262
<b>Copie et référence . . . . .</b>	<b>264</b>
Le comportement PHP 4 . . . . .	264
PHP 5, le passage par référence . . . . .	265

Garder la compatibilité avec PHP 4 . . . . .	266
La copie explicite d'objet, ou clonage . . . . .	266
Égalité et identité . . . . .	269
<b>Constructeurs et destructeurs . . . . .</b>	<b>269</b>
Constructeur . . . . .	269
Destructeur . . . . .	271
<b>La notion d'héritage . . . . .</b>	<b>272</b>
Définition de la notion d'héritage . . . . .	272
Définition d'une classe héritée . . . . .	272
Redéfinition d'attribut ou de méthode . . . . .	274
Accès aux méthodes parentes . . . . .	275
<b>Sûreté de programmation . . . . .</b>	<b>276</b>
Typage . . . . .	279
Classes abstraites et interfaces . . . . .	280
<b>Accès statiques . . . . .</b>	<b>283</b>
Accès à une classe arbitraire . . . . .	283
Définition en vue d'un accès statique . . . . .	283
Accès à la classe en cours . . . . .	284
Accès à la classe parente . . . . .	284
<b>Chargement automatique . . . . .</b>	<b>285</b>
<b>Utilisation via les sessions . . . . .</b>	<b>285</b>
Utilisation de <code>__sleep()</code> et <code>__wakeup()</code> . . . . .	285
<b>Surcharge . . . . .</b>	<b>286</b>
Affectations des attributs . . . . .	287
Lecture d'attribut (Mutator) . . . . .	287
Appel d'une méthode (Accessor) . . . . .	288
<b>Itérateurs . . . . .</b>	<b>288</b>
Utilisation simple . . . . .	288
Utilisation complète . . . . .	291
<b>Notations d'index . . . . .</b>	<b>292</b>
Auto-incrémentation . . . . .	293
<b>Coupler PHP et UML . . . . .</b>	<b>294</b>
<b>Introspection . . . . .</b>	<b>295</b>
Principes pour démarrer . . . . .	295
Les fonctions . . . . .	297

Les objets, classes et interfaces .....	300
Les attributs .....	302
CHAPITRE 13	
<b>Gestion de fichiers</b> .....	303
<b>Lecture et écriture</b> .....	303
Fonctions d'accès rapide .....	304
Ouverture d'un fichier .....	310
Lecture d'un fichier .....	313
Écriture dans un fichier .....	314
Positions dans le fichier .....	315
Fermeture d'un fichier .....	317
Gestion du tampon .....	317
Accès concurrents .....	318
<b>Manipulation de fichiers</b> .....	320
Copie et déplacement .....	320
Création et effacement .....	321
Liens .....	322
<b>Gestion des répertoires</b> .....	323
Parcourir un répertoire .....	323
Position dans l'arborescence .....	325
Créations et effacements .....	326
<b>Informations sur les fichiers</b> .....	326
Existence d'un fichier .....	327
Dates de fichiers .....	327
Taille de fichier .....	328
Espace disque disponible .....	328
Nom et adresse d'un fichier .....	329
Nature des fichiers .....	329
Liens symboliques .....	330
<b>Permissions et droits d'accès</b> .....	330
Changement de propriétaire .....	332
Modifier les permissions .....	332
Masque par défaut .....	332

<b>Sécurité et fichiers</b> .....	333
Permissions et droits d'accès .....	333
Arguments utilisateur .....	333
Safe_mode et open_basedir .....	333
<b>Cas d'application</b> .....	334
Outil de gestion documentaire simple .....	334
CHAPITRE 14	
<b>Gestion des flux</b> .....	337
<b>Exécution de programmes</b> .....	337
Lancement sans interactions .....	337
Lancement interactif .....	341
Sécurité et programmes externes .....	347
<b>Gestion des sockets réseau</b> .....	347
Ouverture .....	348
Lecture et écriture .....	349
Fermeture .....	349
Fonctions de contrôle .....	349
<b>Gestion unifiée des flux</b> .....	351
Types de flux gérés .....	352
Utilisation simple .....	355
Contextes .....	359
Filtres .....	361
Types personnalisés .....	364
<b>Cas d'application</b> .....	366
Système de paiement en ligne .....	366
Sauvegardes automatiques pour interface réseau .....	369
Conversion entre jeux de caractères .....	371
CHAPITRE 15	
<b>Flux de sortie PHP</b> .....	375
<b>Principes et utilisations</b> .....	375
Principe de fonctionnement .....	375
Exemples d'utilisation .....	376
<b>Gestion du tampon de sortie</b> .....	377
Début et arrêt de la mise en tampon .....	377

Récupération du contenu .....	377
Imbrication de tampons .....	379
Informations sur le tampon. ....	379
<b>Filtres automatiques</b> .....	380
Compression des pages avec zlib .....	380
Conversion entre jeux de caractères .....	381
Filtres utilisateur. ....	382
Automatisation .....	383
<b>Tampon interne de PHP</b> .....	384
Délai avant affichage .....	384
Vider le tampon .....	384
Autres tampons en jeu .....	384
CHAPITRE 16	
<b>Envoyer et recevoir des e-mails</b> .....	385
<b>De l'utilité de gérer des e-mails</b> .....	385
<b>Webmail Open Source</b> .....	386
<b>Mise en œuvre</b> .....	388
Prérequis techniques .....	388
Anatomie d'un e-mail. ....	389
Envoyer des e-mails .....	390
Courrier électronique multimédia .....	393
Envoyer des e-mails au format HTML. ....	395
Envoyer des pièces jointes .....	397
Recevoir des e-mails. ....	402
<b>Astuces et sécurité</b> .....	406
Lancer un script à la réception .....	406
Vérification d'une adresse e-mail. ....	407
Espacer vos envois en masse .....	408
<b>Bibliothèques Open Source</b> .....	408
HTML Mime mail par phpguru.org .....	408
<b>Cas d'application</b> .....	411
Gestion d'une lettre d'information .....	411



## CHAPITRE 17

**Travailler avec**

<b>une base de données</b> .....	413
<b>Utilisation d'un SGBD</b> .....	413
Qu'est-ce qu'un SGBD ? .....	413
Travailler avec un SGBD relationnel .....	414
<b>Présentation de MySQL</b> .....	415
Points forts/points faibles .....	415
Fonctionnalités .....	417
Types de tables MySQL .....	418
<b>Outils d'administration Open Source</b> .....	421
phpMyAdmin .....	422
<b>Les commandes SQL</b> .....	423
Créer une base de données .....	424
Créer des tables .....	425
Modifier des tables .....	431
Supprimer des tables .....	432
Insérer des données (INSERT) .....	432
Modifier des données (UPDATE) .....	434
Effacer des données (DELETE) .....	435
Remplacer des données (REPLACE) .....	436
Filtrer avec la clause WHERE .....	437
Sélectionner des données (SELECT) .....	438
Gérer les transactions .....	441

## CHAPITRE 18

<b>Utiliser une base de données avec PHP</b> .....	443
<b>Approche classique PHP 4</b> .....	443
<b>PDO, PHP Data Object</b> .....	445
Particularités .....	445
<b>Utiliser votre base de données</b> .....	446
<b>Structure des classes de PDO</b> .....	447
Prise en main rapide .....	448
<b>Connexion au serveur de données</b> .....	450
Structure du DSN .....	450

Utiliser des connexions persistantes . . . . .	451
Gérer les erreurs de connexion . . . . .	452
Fermer une connexion . . . . .	453
Se connecter à plusieurs bases de données . . . . .	454
Créer un fichier de configuration . . . . .	455
<b>Effectuer une requête . . . . .</b>	<b>456</b>
Requêtes invalides . . . . .	457
Requête de sélection . . . . .	457
Requête d'insertion / modification . . . . .	461
Sécurité et échappements . . . . .	463
<b>Gestion des erreurs . . . . .</b>	<b>465</b>
Utiliser les exceptions . . . . .	466
<b>Gestion des transactions . . . . .</b>	<b>466</b>
<b>Les requêtes préparées . . . . .</b>	<b>468</b>
Construction de la requête . . . . .	469
Préparer une requête . . . . .	470
Lier des données à des paramètres et exécution . . . . .	470
Exploitation d'une requête de sélection . . . . .	472
Fermeture de la requête préparée . . . . .	473
<b>Cas d'application . . . . .</b>	<b>473</b>
Gestion de publication . . . . .	473
 CHAPITRE 19	
<b>Erreurs et exceptions . . . . .</b>	<b>483</b>
<b>Explications sur les erreurs . . . . .</b>	<b>483</b>
Qu'est-ce qu'une erreur ? . . . . .	483
Pourquoi gérer les erreurs ? . . . . .	484
Que faire avec les erreurs ? . . . . .	484
<b>Les erreurs PHP . . . . .</b>	<b>485</b>
Description d'une erreur PHP . . . . .	485
Les bases d'une gestion d'erreur . . . . .	486
Niveaux d'erreurs et filtres . . . . .	488
Créer une erreur manuellement . . . . .	491
Affichage des erreurs . . . . .	492
Journalisation des erreurs (log) . . . . .	493
Personnaliser le gestionnaire d'erreurs . . . . .	498

<b>Les assertions</b> . . . . .	499
Description d'une assertion . . . . .	499
Utilisation d'une assertion . . . . .	499
Désactivation des assertions . . . . .	500
Configuration des assertions . . . . .	500
Personnalisation de la gestion . . . . .	502
<b>Les exceptions</b> . . . . .	503
Description d'une exception . . . . .	503
Lancement d'une exception . . . . .	504
Réception d'une exception . . . . .	504
Filtrage des exceptions reçues . . . . .	505
Propagation des exceptions . . . . .	506
Utilisation des exceptions . . . . .	508
<b>Politiques de gestion d'erreur</b> . . . . .	509
Le développement . . . . .	509
Être averti lors d'un problème . . . . .	509
Toujours agir lors d'une erreur . . . . .	510
Externaliser les alertes de sécurité . . . . .	510
Gardez des traces sur le contexte . . . . .	510
CHAPITRE 20	
<b>XML : concepts et SimpleXML</b> . . . . .	513
<b>De l'utilité du XML</b> . . . . .	513
Gains apportés par XML . . . . .	514
Exemples d'utilisation . . . . .	514
<b>Présentation et prérequis</b> . . . . .	514
Structure du XML . . . . .	515
Principaux formats . . . . .	519
<b>Gérer le XML à la main</b> . . . . .	521
Création d'un nouveau fichier . . . . .	521
Relecture et manipulations . . . . .	523
<b>Ecrire du XML avec XMLWriter</b> . . . . .	523
Prise en main rapide . . . . .	524
Fonctionnalités avancées . . . . .	525
<b>Utilisation de SimpleXML</b> . . . . .	526
Import et export d'un document . . . . .	527

Manipulation des éléments . . . . .	528
Manipulation des attributs . . . . .	532
Recherche Xpath . . . . .	533
Extension des objets SimpleXML . . . . .	535
<b>Cas d'application</b> . . . . .	535
Lecture d'un fichier RSS . . . . .	535
CHAPITRE 21	
<b>XML avancé</b> . . . . .	539
<b>Relecture d'un XML avec SAX</b> . . . . .	539
Fonctionnement des événements . . . . .	540
Initialisation . . . . .	541
Réagir à des événements . . . . .	543
Envoi des données et analyse . . . . .	547
<b>Manipulation avec DOM</b> . . . . .	549
Structure générale . . . . .	550
L'objet document . . . . .	551
Description d'un nœud . . . . .	553
Navigation dans l'arbre . . . . .	555
Gestion des attributs . . . . .	560
Création de nœuds . . . . .	562
Modification de l'arbre XML . . . . .	564
Création d'un document complet . . . . .	566
Recherche Xpath . . . . .	567
Extension des classes DOM . . . . .	568
Utilisation de Xinclude . . . . .	569
Validation et conformité . . . . .	569
<b>Transformation XML par XSLT</b> . . . . .	570
Utilisation du module XSL . . . . .	570
Initialisation . . . . .	571
Chargement de la feuille de style . . . . .	571
Transformation . . . . .	571
Paramètres de transformation . . . . .	572
Extensions et interactions avec PHP . . . . .	572

## CHAPITRE 22

<b>Les services web</b> .....	575
<b>Introduction aux services web</b> .....	575
Protocoles et technologies .....	575
Principe d'un appel à un service .....	579
<b>Utilisation simple (avec WSDL)</b> .....	583
Créer un client SOAP .....	584
Créer un serveur SOAP .....	585
Persistance .....	589
Cache WSDL .....	590
<b>Utiliser SOAP sans WSDL</b> .....	591
Créer un client SOAP sans WSDL .....	591
Serveur SOAP sans WSDL .....	592
Gestion des types et des structures .....	593
<b>Compatibilité .Net et formats</b> .....	595
Différents formats de message .....	595
Compatibilité avec un service .Net .....	596
<b>Autres détails et possibilités</b> .....	596
Codage caractères .....	596
Définir des en-têtes SOAP .....	597
Utiliser un autre transport que HTTP .....	598
<b>Gestion des erreurs</b> .....	598
Erreurs reçues par un client SOAP .....	599
Utilisation des traces .....	599
Renvoyer une erreur dans un serveur .....	600

## CHAPITRE 23

<b>Les templates</b> .....	601
<b>De l'utilité des templates</b> .....	601
<b>Moteurs de templates Open Source</b> .....	602
Une solution légère : PHPLib .....	602
Le couteau suisse : smarty .....	602
Un système original : Templet .....	603
<b>Différentes approches</b> .....	603
L'approche PHP natif .....	604

L'approche search&replace .....	606
L'approche par composants .....	608
Utilisation de XML et XSLT .....	609
<b>Analyse et choix</b> .....	610
Pérennité de la solution retenue .....	611
Simplicité pour les graphistes .....	611
Simplicité pour les développeurs .....	612
Les performances du moteur .....	612
<b>Bibliothèques Open Source</b> .....	613
PHPLib .....	613
Smarty .....	616
Templeet .....	622
CHAPITRE 24	
<b>Les systèmes de cache</b> .....	627
<b>De l'utilité des caches</b> .....	627
<b>Outils de cache Open Source</b> .....	628
<b>Mise en œuvre</b> .....	628
<b>Les caches globaux</b> .....	628
Cache d'une page HTML .....	629
Cache de fichiers de différents types .....	631
Cache de configuration .....	632
<b>Cache des données utilisateur</b> .....	632
Cache par session .....	633
<b>Les caches HTTP</b> .....	633
Dates de mises à jour des fichiers .....	634
Utilisation des serveurs proxies .....	635
Utiliser la date d'expiration .....	637
<b>Mise à jour du cache</b> .....	637
Détection de la modification .....	638
Temps de validité .....	638
Sites semi-statiques .....	639
<b>Pear::Cache</b> .....	639
La classe générique .....	640

Classe pour le Cache HTML .....	642
Autres caches .....	643
<b>Pear::Cache_Lite</b> .....	644
Utilisation .....	644
Spécialisations .....	645
<b>Étude de cas</b> .....	646
Cache pour un site d'actualité .....	646

## CHAPITRE 25

<b>Gestion des images</b> .....	649
<b>Utilité de la gestion d'images</b> .....	649
<b>Prérequis techniques</b> .....	650
<b>Initialisation et utilisation</b> .....	650
La création du modèle de l'image .....	650
Libérer les ressources mémoire .....	653
Affichage de l'image sur le navigateur .....	654
Enregistrer l'image dans un fichier .....	655
<b>Travail sur une image</b> .....	656
Le référentiel .....	656
Tracer des formes .....	656
Écrire du texte .....	659
Copie d'une zone d'image .....	663
Gestion de la palette de couleurs .....	664
Connaître la taille d'une image .....	665
<b>Astuces et remarques</b> .....	666
Éviter les fausses couleurs .....	666
Limite de temps .....	666
Malvoyants et référencement .....	666
<b>L'outil Open Source de gestion d'albums photos : Gallery</b> .....	667
<b>La bibliothèque Open Source JpGraph</b> .....	668
Installation et configuration .....	669
Architecture de la JpGraph .....	670
Création d'un graphique .....	671
Envoi et enregistrement de l'image .....	672
Gérer les polices de caractères .....	672
Propriétés et méthodes communes .....	673

Les graphiques à base de lignes . . . . .	673
Les graphiques en camembert . . . . .	676
D'autres types de graphiques . . . . .	679
<b>Étude de cas</b> . . . . .	680
Redimensionner des images . . . . .	680
Superposer des images . . . . .	683
CHAPITRE 26	
<b>Expressions régulières</b> . . . . .	685
<b>Syntaxe</b> . . . . .	685
Protections et échappements . . . . .	686
Délimitation et présentation . . . . .	686
Chaîne de recherche simple . . . . .	687
Construction d'expression . . . . .	688
Gestion des occurrences multiples . . . . .	691
Assertions . . . . .	693
Captures . . . . .	695
Modificateurs . . . . .	696
<b>Les fonctions</b> . . . . .	697
Chercher une correspondance . . . . .	697
Faire des remplacements . . . . .	699
Échappement et protections . . . . .	702
<b>Performances</b> . . . . .	703
Fonctionnement du moteur . . . . .	703
Stratégies . . . . .	704
Boucles infinies . . . . .	704
CHAPITRE 27	
<b>Sécurité</b> . . . . .	705
<b>Qu'est-ce que la sécurité ?</b> . . . . .	705
Préoccupations du gestionnaire . . . . .	706
Préoccupations de l'utilisateur . . . . .	706
Pourquoi parler de l'utilisateur ? . . . . .	707
<b>Configuration et sécurité</b> . . . . .	708
Interface avec le serveur web . . . . .	709
Safe_mode et restrictions . . . . .	711



Échappement automatique . . . . .	712
Variables globales . . . . .	713
Sessions et identifiants . . . . .	713
Mises à jour du logiciel . . . . .	715
Stockage des données et fichiers. . . . .	715
<b>Sécurité de l'application</b> . . . . .	717
Vérification des entrées utilisateur . . . . .	717
Éviter les principales attaques . . . . .	721
Emplacement des contrôles . . . . .	725
Gérer les erreurs . . . . .	728
Sécuriser les sessions . . . . .	728
Chiffrement et sécurité . . . . .	729
<b>Bonnes habitudes</b> . . . . .	731
Vérifiez vos résultats. . . . .	732
Ne croyez pas l'utilisateur . . . . .	733
N'exagérez pas . . . . .	734
Faites faire un audit externe . . . . .	734
CHAPITRE 28	
<b>Outils de développement PHP</b> . . . . .	735
<b>Éditeurs de texte &amp; IDE</b> . . . . .	735
UltraEdit . . . . .	736
PHPEdit . . . . .	738
Eclipse . . . . .	744
Le Zend Studio . . . . .	746
<b>Les outils de modélisation/RAD</b> . . . . .	749
Macromedia Dreamweaver MX . . . . .	749
WaterProof ::UML . . . . .	754
UML2PHP5 . . . . .	759
CHAPITRE 29	
<b>Les frameworks</b> . . . . .	763
<b>Ce qu'est un framework</b> . . . . .	763
Un cadre de travail . . . . .	763
La séparation MVC . . . . .	764
Les avantages d'un framework. . . . .	764

<b>Quelques frameworks disponibles en Open Source</b> .....	765
Copix et Jelix .....	765
Symfony .....	766
Zend Framework .....	766
Les autres .....	767
<b>Courte introduction à Symfony</b> .....	768
Installation .....	768
Initialisation de l'application .....	768
Génération du modèle .....	769
Premier contrôleur .....	769
Lien avec la vue .....	771
Le test .....	772
Quelques points non abordés .....	772
Documentation .....	773
ANNEXE	
<b>Ressources en ligne</b> .....	775
Bibliothèques .....	775
Applications PHP .....	779
ERP .....	781
<b>Index</b> .....	785



# Avant-propos

---

## Pourquoi ce livre ?

Pourquoi écrire un livre si son sujet n'est pas une affaire de passion ? En effet, pour nous, PHP est une affaire de cœur. Nous allons vous transmettre non seulement un savoir mais aussi une expérience et une passion.

PHP peut être considéré comme un des fers de lance du monde Open Source. Toute l'image de cette philosophie de partage et d'entraide s'exprime à travers lui. Et si à une belle idée, on associe un produit fiable, stable, complet et étendu, pourquoi hésiter ?

En dépit de ses atouts, PHP a été longtemps perçu par les professionnels comme un outil pour pages personnelles ou petits projets. Certes, il est adapté à ce type de missions, mais son spectre d'action est nettement plus vaste. Heureusement grâce à ses qualités intrinsèques et à sa communauté qui a réussi à se faire entendre et à séduire, les mentalités ont fini par évoluer et PHP a été élevé à sa juste valeur.

Ce livre, nous l'avons pensé et écrit pour des développeurs pointilleux désirant exploiter au mieux les capacités de PHP. Sans le rendre inaccessible aux débutants, nous souhaitons qu'il soit utile à des développeurs professionnels ou d'un niveau avancé. L'arrivée de PHP 5 n'a finalement été qu'un prétexte pour nous pencher sur cet ouvrage. Nous avons tous deux des profils différents, l'un très technique et puriste, l'autre orienté vers le fonctionnel, la vulgarisation et la pédagogie. Le résultat se veut donc très pointu et très vaste tout en adoptant une approche pédagogique.

La nouvelle version de PHP rompt avec l'ancien modèle objet : celui-ci, limité, a été remplacé par un modèle objet complet, et de nombreuses fonctionnalités ayant pour but de faciliter la vie du développeur ont été introduites. Nous avons désormais un langage mature, adapté à des projets web professionnels et qui n'a pas à rougir d'une comparaison avec d'autres langages ou architectures.

Ces pages ont été conçues de façon à souligner ces nouveaux ajouts et à fournir une référence utile au jour le jour pour les développeurs PHP. Contrairement à d'autres ouvrages qui se fondent massivement sur l'excellente documentation de PHP (visible en ligne et à jour sur [fr.php.net](http://fr.php.net)), nous avons souhaité réaliser un livre qui lui apporte une réelle valeur ajoutée, dépassant le simple étalage des fonctions et des paramètres.

## Structure de l'ouvrage

Cet ouvrage s'articule autour des thèmes abordés lors du développement d'une application web. Chaque chapitre est centré sur un de ces thèmes. Il décrit les différentes fonctionnalités PHP qui s'y rapportent mais aussi tout ce qui les entoure et qui permettra de les mettre en œuvre. Des exemples concrets, des cas d'applications pratiques et des retours d'expériences seront régulièrement présentés.

La première partie du livre fait office d'entrée en la matière :

- Le **chapitre 1** donne toutes les informations sur la plate-forme PHP, sa diffusion et les ressources d'aide que vous pourrez trouver, francophones et internationales.
- Le **chapitre 2** détaille les options de configuration les plus importantes et les procédures d'installation, sous Unix et Microsoft Windows. Il y est également présenté les différents points à prendre en compte pour une migration de PHP4 vers PHP5.

La partie suivante concerne les fonctionnalités de base du langage. On y trouve les rappels sur la syntaxe et les structures, puis l'interface avec les pages web via les formulaires ou cookies. Cette partie permet aux débutants d'apprendre les bonnes bases de PHP. Les développeurs confirmés pourront, eux, y trouver une référence avec quelques astuces et détails utiles :

- Le **chapitre 3** fait un rappel des syntaxes de base du langage PHP : types de données, affectation, organisation du code, etc.
- Le **chapitre 4** montre les structures de bases de PHP : les différentes boucles et conditions.
- Le **chapitre 5** détaille les différentes fonctions de gestion des chaînes de caractères.
- Le **chapitre 6** se focalise sur la gestion des tableaux et les fonctions afférentes.
- Le **chapitre 7** présente les quelques fonctions usuelles qui ne se rapportent pas à un sujet particulier et qui sont souvent utiles lors de développements.
- Le **chapitre 8** décrit l'interaction entre PHP et les formulaires HTML (variables, fichiers), ainsi que les superglobales PHP permettant leur manipulation.
- Le **chapitre 9** est le dernier de cette première partie très orientée vers la référence, il complète le précédent en s'intéressant à l'environnement autour de PHP : principalement la communication avec le serveur web, le système et le réseau.

La troisième partie entre dans le cœur du sujet en se focalisant sur différents thèmes rencontrés dans le cadre du développement d'applications poussées. Le développeur confirmé y trouvera matière à progresser :

- Le **chapitre 10** commence cette section avec une description avancée des cookies, de leur utilisation et de leur environnement. On y retrouvera aussi quelques informations liées à la sécurité.
- Le **chapitre 11** prend la suite du chapitre sur les cookies pour évoquer les sessions. Outre la description simple de leur utilisation, nous abordons une réflexion globale sur

les sessions, leur utilité et leur sécurité. Les développeurs confirmés y trouveront les informations pour mettre en œuvre leur propre gestionnaire de session.

- Le **chapitre 12** présente la plus grosse avancée de PHP 5 : la programmation orientée objet. Une description complète des fonctionnalités y est faite, mettant en exergue les différences fondamentales par rapport à PHP 4.
- Le **chapitre 13** décrit en détail la gestion des fichiers : lecture, écriture, manipulations, fichiers distants, etc.
- Le **chapitre 14** étend les notions abordées avec les fichiers pour manipuler tous types de flux de données : sockets réseaux, exécution de programmes externes et flux personnalisés.
- Le **chapitre 15** s'intéresse à la gestion du tampon de sortie de PHP : pouvoir appliquer un filtre sur les données envoyées au navigateur, pouvoir manipuler le flux de sortie pour compresser les pages web, etc.
- Le **chapitre 16** détaille tout ce que vous devez savoir concernant l'envoi et la réception d'e-mails : de l'utilisation pour envoyer un simple message texte jusqu'à la description des e-mails HTML ou avec pièces jointes.
- Le **chapitre 17** est dédié au langage SQL et aux SGBD en général. Une approche poussée du cas de MySQL est réalisée.
- Le **chapitre 18** présente en détail comment communiquer entre PHP et une base de données en utilisant PDO (PHP Data Object).
- Le **chapitre 19** est dédié à la gestion des erreurs avec PHP. La première partie décrit la gestion des erreurs classiques telles qu'on peut les voir dans PHP 4 et des assertions. La seconde partie décrit une nouveauté de PHP 5 : les exceptions. D'autres points comme la configuration, les journaux d'erreur ou la politique de gestion des erreurs sont aussi abordés.
- Le **chapitre 20** présente une autre nouveauté de PHP 5 : la gestion XML avec SimpleXML. Les notions basiques de gestion XML y seront abordées, ainsi que tout ce dont vous avez besoin pour lire et manipuler rapidement du XML.
- Le **chapitre 21** complète le précédent en donnant les méthodes pour les manipulations avancées que vous pourriez avoir à faire avec XML : SAX, DOM, XSLT, etc.
- Le **chapitre 22** traite des services web et particulièrement de SOAP.
- Le **chapitre 23** traite de la dissociation de la logique métier et du visuel : les templates.
- Le **chapitre 24** aborde toutes les problématiques de la gestion des caches. Il vous donne toutes les clés pour trouver ou créer le système adapté à vos besoins.
- Le **chapitre 25** détaille l'utilisation de l'extension GD. Elle vous permettra de produire ou manipuler facilement des images, des photos diagrammes ou des graphiques avec PHP.
- Le **chapitre 26** se focalise sur l'utilisation des expressions régulières. La syntaxe et l'utilisation des expressions compatibles Perl supportées par PHP seront décrites en détail.

La quatrième et dernière partie traite des sujets annexes lors de vos développements, la sécurité et les outils :

- Le **chapitre 27** fait un tour des aspects de la sécurité à prendre en compte lors du développement d'une application. Vous y trouverez des exemples de failles ou de problèmes fréquents ainsi que les bonnes habitudes pour les éviter.
- Les **chapitres 28** et **29** achèvent ce livre avec une description des différents outils de développement pour PHP et des frameworks intéressants.

## Remerciements

Nous tenons à remercier tous ceux qui nous ont aidés à rédiger ce livre.

Aux familles, proches et amis pour leur soutien et leur patience pendant ces longs mois de rédaction et de réflexion,

à Eyrolles pour avoir cru en notre projet et l'avoir soutenu dès le départ,

à Sarah Gedon, Romain Bourdon, Guillaume Ponçon, Sarah Haim, Grégoire Cachet, Valérie Poinotte et Stéphane Deschamps pour leurs multiples aides pour le développement des divers chapitres,

à, dans le désordre, Christophe Gesché (Moosh), Paul Bardinon, Jérôme Renard, , Alain Gazalet, Eudes Robichon, Frédéric Bordage, Guillaume Bouchard, Julien Jackubowski, Yoan Blanc, Laurent Jouanneau, Damien et Ghislain Seguy, Quentin Sinagra, Remi Pauchet, KDO, Xavier Langlet, Jean-Eudes Amrein, Raphaël Rousseau et Stéphane Raviart pour les diverses relectures qu'ils ont pu faire,

à tous les lecteurs des précédentes éditions, qui par leurs retours nous ont permis d'améliorer cet ouvrage,

... et tous les autres dont nous n'avons pas le nom complet, que nous n'avons pas pu recontacter ou que nous avons simplement oubliés dans la précipitation juste avant l'impression de cette page.

Merci à tous, car sans vous ce livre n'aurait peut-être pas vu le jour.

Éric Daspet et Cyril Pierre de Geyer

# 1

## Qu'est-ce que PHP ?

---

PHP (*PHP Hypertext PreProcessor*) est un langage de programmation. Sa principale application se situe au niveau de la gestion des sites web dynamiques. On peut par exemple lui faire créer le contenu de pages HTML suivant différents paramètres : l'âge d'un visiteur, sa catégorie socioprofessionnelle, des mots-clés qu'il aura indiqués dans un moteur de recherche, des actualités du jour, etc.

Les capacités de PHP ne s'arrêtent pas à la création de pages web. Il est aussi possible de manipuler des images, de créer des fichiers PDF, de se connecter à des bases de données ou des serveurs LDAP, et même d'instancier des objets Java. Un module annexe lui permet également de fournir des interfaces graphiques classiques (client lourd, sans navigateur ou serveur web), via GTK.

Les fonctionnalités de PHP permettant de sortir de l'ordinaire des sites web sont très nombreuses. Dans ce chapitre, nous allons vous montrer que PHP est non seulement un langage mais aussi une plate-forme globale. Nous vous présenterons ses possibilités, ses caractéristiques et son historique. Enfin, nous aborderons PHP du côté français, c'est-à-dire en mettant en avant les ressources et chiffres mis à disposition par la communauté francophone.

### Introduction à PHP

#### *Un langage Open Source*

PHP est à l'origine un langage de script conçu spécifiquement pour agir sur les serveurs web. En ajoutant quelques lignes de PHP à une page HTML, le serveur exécute les instructions correspondantes pour écrire du code HTML à la place. Le résultat (le code HTML initial ajouté à celui produit par PHP) est envoyé au navigateur. Cela permet par



exemple d'afficher la date du jour à un endroit bien précis du visuel. On parle alors de page dynamique.

Dans l'exemple suivant, PHP ajoute une chaîne de caractères au milieu du code HTML :

```
<html>
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <p>
      <?php
        echo "Ceci est une syntaxe PHP";
      ?>
    </p>
  </body>
</html>
```

PHP dispose de près de 3 000 fonctions utilisables dans des applications très variées et couvre pratiquement tous les domaines en rapport avec les applications web. Par exemple, presque tous les SGBD du marché (Systèmes de gestion de bases de données) peuvent s'interfacer avec PHP, qu'ils soient commerciaux ou qu'ils viennent du monde du logiciel libre.

PHP 5 et ses nouveautés propulsent PHP dans le monde des plates-formes d'entreprises comme .Net ou J2EE.

### Licence et téléchargement

PHP est distribué via une licence propre qui permet sa rediffusion, son utilisation et sa modification librement et gratuitement. Il peut être téléchargé depuis le site web officiel sur <http://www.php.net/> ou un de ses miroirs tel que <http://fr.php.net/>.

### Exécution

L'exécution de PHP est similaire à celle de Java ou des langages .NET, c'est-à-dire que les scripts sont convertis en un langage intermédiaire (*byte code*) avant d'être exécutés. Toutefois, à la différence de ces langages, le code intermédiaire de PHP est recréé à chaque exécution et ne peut pas être diffusé. Du point de vue utilisateur, on exploite directement le code source : il n'y a pas d'étape de compilation.

### Courbe d'apprentissage

Reprenant une syntaxe claire et familière puisque très proche de celle du langage C, PHP est un langage dont la prise en main est généralement très rapide. Il est facile d'en apprendre les bases mais il est difficile de le maîtriser pleinement. Effectivement, connaître et utiliser toutes les fonctionnalités et concepts de PHP nécessite un apprentissage poussé.

## Que faire avec PHP ?

La principale utilisation que l'on peut avoir de PHP est l'utilisation d'un langage de script traité côté serveur pour la création de pages web. Cette utilisation sur serveur web est la principale mais PHP peut également être utilisé pour deux autres types de développement.

### Fonctionnement couplé à un serveur web

Le fonctionnement sur un serveur web est l'application la plus répandue. Trois composants entrent en jeu : un serveur web (le plus souvent Apache ou IIS), le module PHP et un navigateur web. Lorsque le serveur web reçoit une demande de page, PHP en élabore le contenu avant de l'envoyer au navigateur. Ce mode de fonctionnement permet de créer des sites Internet dynamiques ou de s'interfacer avec des progiciels pour gérer la logique métier de l'entreprise.

### Applications en ligne de commande

Vous pouvez utiliser PHP de façon autonome, sans serveur web, en ligne de commande. Pour cela, il vous suffit de faire appel à l'exécutable `php`. Cela peut parfois être utile pour réaliser des actions simples sur votre ordinateur (par exemple, changer automatiquement le nom de plusieurs centaines de fichiers) sans nécessiter la présence de tout un contexte web. Pour automatiser des actions, vous pouvez coupler son utilisation au gestionnaire des tâches (serveur `cron` sous Linux). Le fonctionnement est le même : vous appelez un fichier contenant le script via PHP : `php -q rename.php`.

### Services web

PHP permet de créer et d'utiliser des services web. Ce type d'application permet de mettre votre contenu à disposition d'autres personnes. Ainsi, tels Amazon, Google ou Yahoo!, vous pourrez créer vos propres applications que d'autres utiliseront. On parle alors d'applications en « marque blanche ». Amazon, par exemple, vous permet de reprendre son catalogue, de le mettre à vos couleurs et de vendre ses produits comme s'il s'agissait des vôtres. PHP vous permet autant de gérer et de produire des services web que d'en utiliser.

### Applications graphiques

PHP dispose d'une extension permettant de produire des applications graphiques traditionnelles. Il n'y a alors ni serveur web ni navigateur, et l'application s'exécute entièrement sur le poste client. L'extension nécessaire n'est pas incluse par défaut, mais vous pouvez la récupérer sur un site dédié : <http://gtk.php.net/>. L'ajout récent de la prise en charge des bases de données SQLite va donner une toute nouvelle ampleur à ce type de développement. PHP peut alors piloter toute l'application de façon autonome, des fenêtres à la gestion des données sans nécessiter de serveurs ou logiciels annexes. Vous pourrez retrouver au chapitre 18 toutes les informations pour vous connecter à SQLite via PDO.

## Particularités de PHP

Les principaux « concurrents » de PHP sont Perl, .NET et ses différents langages, JSP (*Java Server Pages*), voire ColdFusion.

Globalement, il faut garder en tête qu'à chaque problème correspond sa solution et qu'il est difficile de dire que tel langage ou tel autre est meilleur de façon générale. Cependant, PHP 5 dispose par rapport à ses concurrents de quelques particularités et avantages significatifs.

### De nombreux connecteurs techniques

PHP intègre des possibilités de connexion à la majorité des bases de données (Oracle, SQL Server, MySQL, dBase, ODBC, etc.), annuaires (LDAP, etc.) et systèmes de paiement en ligne (VeriSign, Cybercash, Crédit Mutuel, etc.).

C'est particulièrement intéressant quand on sait que près de 40 % de la charge de développement d'une application est liée à l'intégration d'applications ou de sources de données existantes (selon IDC, cabinet de conseil et d'études sur les marchés des nouvelles technologies de l'information).

L'essentiel des protocoles et des formats qu'on peut rencontrer sur Internet ou intranet sont aussi pris en charge : TCP, HTTP, SMTP, LDAP, IMAP, POP, SSL, Soap, XSLT, XML, PDF, etc.

### Peu de connecteurs applicatifs

Bien que pouvant s'interfacer avec SAP, Lotus Notes, IBM iseries et d'autres progiciels, PHP ne dispose pas d'un grand nombre de connecteurs applicatifs. On peut regretter par exemple l'absence de connecteurs vers les principaux MOM du marché (*Message Oriented Middleware*) tels que Tibco, MQseries ou Microsoft MSMQ. On trouve toutefois un connecteur pour SAP qui permet d'exécuter les différentes fonctions du progiciel.

La possibilité pour PHP de se connecter directement au *backend* (interfaces internes des logiciels) et aux bases de données permet de compenser en partie ce manque.

### Les performances de PHP

PHP est extrêmement performant et fiable, même selon les critères d'application critiques. Avec un seul serveur standard, on peut répondre à des millions de requêtes par jour. Pour des sites à très fort trafic, il existe diverses solutions permettant d'optimiser et d'améliorer les performances globales de PHP.

Des sites ou des applications importantes utilisent PHP (*Le Monde, Le Figaro, TV5, Yahoo, TF1, Canal +...*). Il s'agit maintenant d'une solution reconnue comme viable autant du côté stabilité et fiabilité que du côté des performances. Des chiffres détaillés sur l'utilisation de PHP seront donnés plus loin dans ce chapitre.

## Les bases de données reconnues par PHP

PHP 5 contient des connexions natives vers la plupart des systèmes de gestion de bases de données (SGBD). Avec la version 5, PHP dispose même d'une mini-base de données directement intégrée : SQLite. Voici une liste non exhaustive des bases de données reconnues par PHP : Microsoft SQL server, Oracle, PostgreSQL, MySQL, Sybase, SQLite, FilePro, Informix, Interbase, mSQL, dBase, Empress, et bien d'autres.

De plus, le standard ODBC (*Open DataBase Connectivity*) et les fonctions ODBC de PHP permettent de se connecter à n'importe quelle base de données possédant un pilote ODBC.

## Services web et interopérabilité

PHP est le champion de l'intégration bas niveau. Il est capable d'instancier des objets COM, des classes Java, Python ou .NET. L'intégration de bibliothèques C via des modules PHP est elle aussi aisée.

PHP dispose également d'une couche Soap (avec, entre autres, PEAR::SOAP) et d'une couche XML-RPC. Elles permettent de créer ou de consommer des services web très simplement. Vous pouvez par exemple vous connecter au moteur de recherche Google ou au système d'Amazon pour y effectuer des recherches.

Les flux XML associés aux parseurs XSL/XSLT vous permettent de travailler avec d'autres systèmes d'information. Des connectivités SNMP, LDAP sont aussi disponibles. Les différents modules de PHP couvrent une base extrêmement large sur tout ce qui peut être en interaction avec un script web. Il serait surprenant que vous n'y trouviez pas de quoi couvrir vos besoins.

## Bibliothèques intégrées

PHP a été conçu pour le web et, par conséquent, il dispose de nombreuses fonctions permettant d'effectuer la majorité des actions en rapport avec le web.

On peut par exemple trouver la création de fichiers PDF, la création d'images à la volée, la connexion et la communication avec d'autres serveurs web ou FTP, ainsi que l'envoi et la réception de courrier électronique. Toutes ces bibliothèques bénéficient de fonctions de haut niveau permettant au programmeur de se concentrer sur son application au lieu de gérer les détails de chaque composant.

## La portabilité

PHP est disponible pour plusieurs systèmes d'exploitation. Il fonctionne sous MS Windows (toutes versions supérieures à Windows 95) et l'essentiel des versions d'Unix ou associés (par exemple Solaris, Linux, OpenBSD, FreeBSD, MacOS X, etc.). Votre code pourra être utilisé sur toutes ces plates-formes de la même façon et sans modification de code.

### Coûts de licence

PHP est gratuit. Vous pouvez, à tout moment, vous procurer la dernière version sur le site : <http://www.php.net>, sans payer quoi que ce soit. Cependant le prix du logiciel PHP n'est pas le seul à entrer en compte. Il faut aussi prévoir le prix du système d'exploitation, d'une éventuelle base de données, du serveur web, etc. L'avantage de PHP est qu'il peut, comme indiqué précédemment, être utilisé dans la majorité des cas. Ainsi, vous pourriez autant l'utiliser avec une plate-forme sous Linux qu'avec une plate-forme sous Windows, voire sur AS400. Dans cette optique, vous pouvez utiliser PHP couplé à un serveur Linux et une base de données MySQL sans déboursier un centime d'euro.

### Coûts de développement

Un développement fait en PHP est généralement plus rapide qu'un développement effectué sous J2EE ou .Net, le code étant plus court et moins complexe. De plus, actuellement, le coût journalier d'un bon développeur PHP est moins élevé que celui d'un bon développeur Java.

Ainsi, globalement, les coûts de développement PHP sont généralement moins importants que les coûts induits par l'utilisation des alternatives.

### Le code source

Le code source de PHP est disponible gratuitement. À l'inverse des produits commerciaux dont les sources ne sont pas distribuées, vous avez la possibilité de modifier tout ou partie des sources pour adapter PHP à vos besoins spécifiques. Le produit modifié peut être vendu et redistribué librement suivant vos propres conditions.

### Dynamisme de la communauté PHP

La communauté PHP est estimée par la société Zend à près de 4 500 000 développeurs courant 2007. Elle est très organisée et très réactive. L'annonce d'une faille de sécurité implique généralement un correctif dans la journée. De plus, de nombreuses personnes développent des outils Open Source de très bonne facture et les proposent au public.

## Historique

Contrairement à d'autres langages comme le C, le C++, voire le Perl, PHP est un langage assez jeune. Son évolution sur quelques années en a fait l'un des langages les plus importants du Web.

### PHP/FI

PHP/FI a été créé en 1995 par Rasmus Lerdorf. À l'origine, il s'agissait d'une bibliothèque de scripts fonctionnant sous Perl, dont l'objectif était, entre autres, de permettre à son auteur de savoir qui venait consulter son CV sur son site personnel. Rasmus donna donc à cette bibliothèque son premier nom : *Personal Home Page Tools*.

Petit à petit, la bibliothèque Perl s'est muée en une implémentation directement en C, l'objectif étant des gains de performances et des possibilités plus poussées : communiquer avec les bases de données, créer des applications dynamiques pour le Web, etc.

À ce stade, Rasmus décida de proposer son code à la communauté afin que tout le monde puisse l'utiliser et en profiter, voire contribuer à son développement.

PHP/FI signifiait à cette époque *Personal Home Page / Forms Interpreter* pour indiquer, chose rare à l'époque, que PHP/FI gérait les formulaires (FI pour Interpréteur de formulaire). Ses principales caractéristiques étaient la simplicité d'insertion dans du HTML, une syntaxe proche du Perl et un système d'interprétation des variables de formulaires.

Bien que très jeune, le langage disposait déjà de nombreux adeptes. En 1997, on estimait l'audience à plusieurs milliers d'utilisateurs. Près de 50 000 domaines avaient installé PHP (soit 1 % des noms de domaines).

PHP/FI 2.0 fut publié officiellement en novembre 1997, après avoir passé l'essentiel de sa vie en version bêta. Peu de temps après, une version alpha de PHP 3.0 était publiée.

### PHP 3

PHP 3.0 n'est pas réellement une suite à PHP/FI mais plutôt une refonte. En 1997, Andi Gutschman et Zeev Suraski (fondateurs de Zend : combinaison des prénoms Zeev et Andi) essayèrent d'utiliser PHP/FI dans le cadre du développement d'une application de e-commerce, mais les performances n'étaient pas suffisantes. Ils décidèrent de réécrire PHP/FI de façon complète.

PHP 3.0 a été la première version de PHP assez fonctionnelle et stable pour être mise en production sur de véritables projets. Afin d'assurer une continuité avec PHP/FI, Rasmus rejoignit le projet PHP 3.0, qui devint le successeur officiel de PHP/FI 2.0.

Parmi les nouvelles fonctionnalités de PHP 3.0, la possibilité d'y intégrer des extensions fut sûrement celle qui lui permit de connaître un tel succès. En effet, une API modulaire donna la possibilité à n'importe quel développeur de créer ses propres modules et de les partager avec l'ensemble de la communauté. Des modules permettant de créer des images dynamiquement ou de travailler sur des fichiers PDF sont ainsi apparus.

Avec cette nouvelle mouture, PHP devenait un langage de programmation à part entière et se devait de prendre un nom plus professionnel. C'est ainsi que PHP devint *PHP Hypertext Preprocessor*.

Au bout d'une dizaine de mois de test et de débogage, la première version officielle de PHP 3.0 fut lancée en juin 1998. À la fin de cette même année, PHP était déjà utilisé sur des centaines de milliers de sites. On estime que PHP 3.0, à son apogée, était installé sur 10 % du parc mondial des serveurs web.

## PHP 4

Juste après la publication de PHP 3.0, Andi et Zeev se remirent au travail pour réécrire totalement le moteur de PHP car, malgré ses fonctionnalités et sa stabilité, ils n'étaient pas satisfaits de ses performances.

Ils commencèrent donc à travailler sur ce qu'on appellera par la suite le *Zend Engine*. Une première version de ce moteur fut publiée dans le courant de l'année 1999, mais ce n'est qu'en mai 2000 qu'il fut officiellement intégré à PHP dans sa nouvelle version : PHP 4.0.

En plus de ce nouveau moteur apportant des performances beaucoup plus élevées, les principales évolutions de PHP 4.0 par rapport à son prédécesseur tenaient à sa prise en charge des sessions HTTP et de nombreux serveurs web, ainsi qu'à la mise en tampon des sorties et à une sécurité accrue des informations visiteurs.

## PHP 5

Le développement de PHP 5 a été entamé en 2002, mais c'est l'année 2003 qui a été la plus active. L'objectif était double : d'une part, rendre PHP plus professionnel, mais également le simplifier.

La première version stable de PHP 5 a fait son apparition en 2004. Les versions 5.1 et 5.2, quant à elles, sont respectivement sorties en 2005 et 2006. Par rapport à la version 4, les principales nouveautés sont :

- l'intégration du Zend Engine 2, qui amène une prise en charge complète de la programmation orientée objet ;
- la refonte de la prise en charge de XML ;
- l'intégration de la base de données SQLite ;
- la simplification des principales tâches courantes.
- l'apparition d'un socle commun pour la gestion des appels aux bases de données : PHP Data Object (PDO).

Vous trouverez plus loin dans ce chapitre un paragraphe dédié aux nouveautés de PHP 5.

## ***Mode de développement du projet PHP***

Le mode de développement de PHP, fondé sur le travail collaboratif, impressionne. Très souvent, durant des sessions de formation, les gens s'étonnent qu'un tel outil ait pu être développé bénévolement.

C'est pourtant le cas ; cependant, pour qu'un tel système fonctionne, une hiérarchie se doit d'être définie et suivie tout en restant souple.

### **Les différentes équipes**

Plusieurs équipes travaillent au développement de PHP :

- équipe de développement (500 personnes) ;

- équipe qualité (250 personnes) ;
- équipe de documentation (120 personnes) ;
- équipe de traduction (120 personnes).

Étant donné que de nombreux contributeurs participent à plusieurs équipes, on estime leur nombre total à 700 contributeurs. Une illustration de l'organisation est donnée à la figure 1-1.



Figure 1-1

*Déroulement du développement*

#### Note

On notera cependant que ces contributeurs ne travaillent pas en permanence ni toujours ensemble, mais à leur rythme et en alternance. Ainsi, on peut estimer qu'environ 10 % des inscrits travaillent à un moment donné.

#### L'équipe de développement

Les sorties (*releases*) sont généralement gérées par un RM (*Release Master*) qui joue le rôle de l'organisateur. Il est éventuellement aidé par un RMB (*Release Master Bitche*), dont le rôle est de gérer les tâches ingrates : servir d'avocat du diable, recueillir les critiques et les bogues, etc.

La désignation d'un RM se fait sur une base de volontariat et par approbation de ses pairs.

Les développeurs utilisent l'outil CVS pour gérer les différentes versions. Chacun d'entre eux propose ses sources à son rythme via cet outil CVS.

#### CVS : Current Version System

CVS est un système libre permettant à plusieurs agents de travailler simultanément sur un même projet, tout en gardant la trace de toutes les modifications survenues.

Conçu pour faciliter le travail de développement en équipe, il conserve les révisions successives. Il facilite ainsi la collaboration de multiples intervenants sur un même projet.

Qu'il travaille localement (sur la même machine) où à distance (en réseau), chacun n'accède jamais qu'à une copie des fichiers. Les originaux demeurent sur le « référentiel » et ne sont modifiés qu'à travers les mécanismes sécurisés et « journalisés » de CVS.



#### L'équipe de gestion qualité

Une fois une version candidate à la mise en ligne prête, l'équipe de qualité entre en jeu. Son travail consiste à effectuer des batteries de tests sur l'ensemble de la version candidate. Une version candidate n'est jamais proposée sans qu'elle ait passé l'ensemble des tests.

#### L'équipe de documentation

L'équipe de documentation travaille à la mise en place de documentation pour les utilisateurs. La première version étalon se fait en anglais.

#### L'équipe de traduction

Pour que chacun puisse accéder facilement à l'information dans sa propre langue, des équipes internationales œuvrent à traduire la documentation dans leur langue maternelle. On remarquera d'ailleurs que le site <http://php.net> met automatiquement à disposition la documentation dans votre langue.

## Nouveautés de PHP 5

### *La programmation orientée objet*

PHP 5 a fait son apparition en 2004. Sa principale nouveauté réside dans la nouvelle mouture de son moteur : le *Zend Engine 2*. Ce nouveau moteur permet de gérer dans leur ensemble les aspects de la programmation objet, remédiant ainsi à ce que certains considéraient comme un défaut.

### *Refonte et simplification de XML*

Les autres nouveautés concernent la gestion de XML. La version 4 de PHP impliquait une utilisation relativement lourde pour qui souhaitait manipuler des flux XML. Avec la version 5, deux nouveautés révolutionnent sa manipulation :

- l'intégration d'un nouveau gestionnaire XML : la bibliothèque libxml2, qui amène une implémentation DOM standard complète ;
- l'extension SimpleXML.

La première permet de traiter tous les aspects de la manipulation XML, avec la complexité que cela implique.

La seconde s'adresse à tous les traitements XML simples. Il n'est plus obligatoire de passer des opérations compliquées pour récupérer les données de fichiers XML.

### *Intégration de la base SQLite*

Enfin, concernant les bases de données, le PHPGroup a souhaité mettre en avant une nouvelle solution en proposant une base de données intégrée directement dans PHP : il

s'agit de SQLite. Celle-ci dispose de nombreuses fonctionnalités et ne nécessite pas l'installation de serveur de bases de données. Outre toutes les applications web qui pourront profiter de cette nouveauté, on peut imaginer que le couple PHP/GTK permettant de créer des applications locales fenêtrées devrait prendre son envol.

## ***Simplification des tâches courantes***

Les autres ajouts sont liés à l'objectif de simplifier les tâches les plus courantes. Ainsi, de nombreuses fonctions ont vu le jour et la gestion des erreurs a été repensée. Enfin, la compatibilité avec PHP 4 a été au cœur des préoccupations et seules certaines spécificités dans la programmation objet nécessiteront d'être reformulées.

## ***PDO : socle commun aux SGBD***

PDO (PHP Data Object) est la principale nouveauté de PHP 5.1. Cette extension vous apportera un confort d'utilisation et une abstraction plus importante que les anciennes fonctions natives propres à chaque SGBD. L'approche objet de PDO vous permettra, de plus, d'étendre facilement les fonctions d'accès à votre base de manière transparente.

En interne, PDO permet à l'équipe de développement de PHP de développer beaucoup plus rapidement de nouveaux connecteurs vers de nouvelles bases de données. Au lieu de tout réécrire du début comme auparavant, ils peuvent se baser sur une architecture complète et ne rajouter que ce qui est spécifique.

PDO est un socle commun pour les connecteurs vers les SGBD. Il fournit des fonctions de base et unifie les interfaces utilisateur. Il ne constitue pas à proprement parler un système d'abstraction aux bases de données, bien qu'il puisse servir en ce sens.

## **Architecture et fonctionnement**

### ***Architecture technique***

Dans la plupart des déploiements, PHP est utilisé conjointement avec :

- généralement Apache comme serveur HTTP ou, plus rarement, Microsoft IIS ;
- MySQL et Oracle comme SGBD/R ; on peut aussi rencontrer PostgreSQL ou Microsoft SQL Server ;
- Linux ou BSD comme système d'exploitation ; Windows ou MacOS sont aussi des possibilités fonctionnelles.

Les plates-formes en production reposent en majorité sur le quatuor Linux, Apache, MySQL et PHP (LAMP).

Grâce à ses nombreux connecteurs et à la prise en charge de Java, COM et .Net, PHP est capable de se connecter à la plupart des applications existantes de l'entreprise.

Cette plate-forme peut ensuite exposer l'existant de l'entreprise et les nouveaux développements au travers de différents types d'interfaces :

- web (HTML, WML, etc.) ;
- services web reposant sur Soap ;
- applications graphiques ;
- client riche ;
- Ajax ;
- ligne de commande (CLI) ;
- et même Microsoft Office (Word, Excel), Adobe PDF, Macromedia Flash (via Ming), etc.

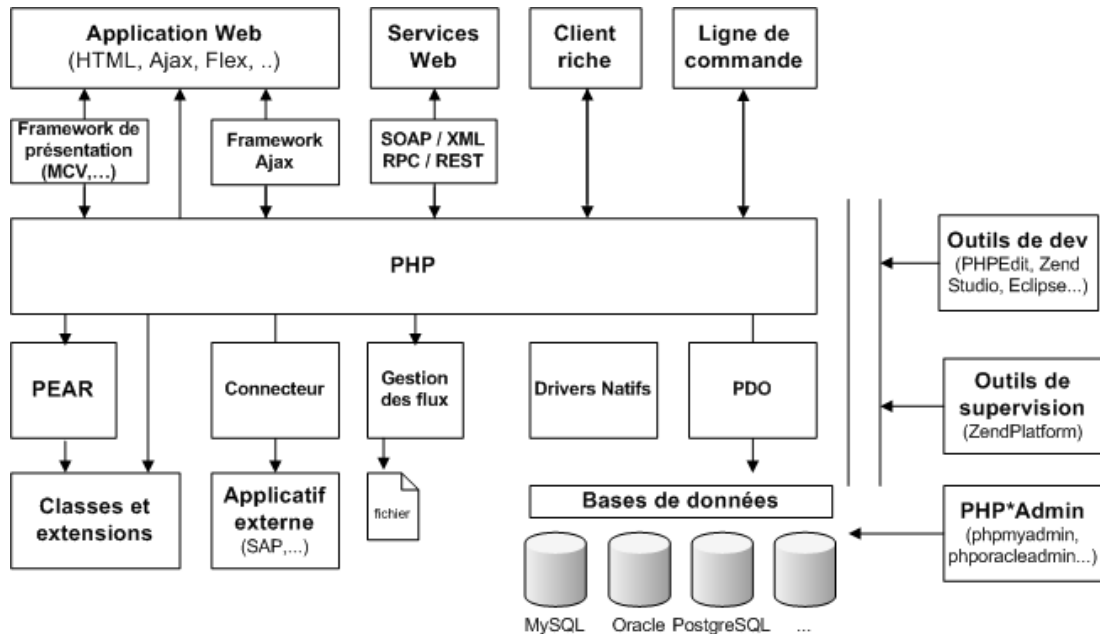


Figure 1-2

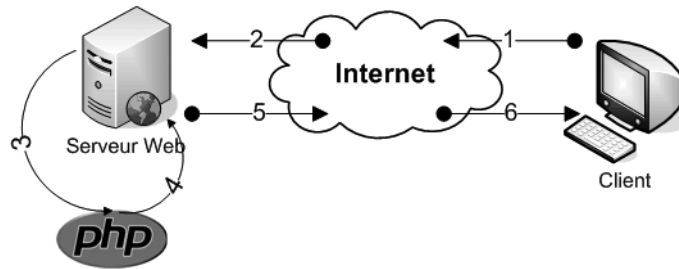
Architecture technique de PHP

## Fonctionnement de PHP

L'utilisateur qui appelle une page PHP ignore tout du code sous-jacent. Effectivement, ce code est interprété par le serveur avant d'être traduit dans le format de sortie (généralement en HTML, mais aussi en XML, fichier PDF, etc.). Pour ce faire, le serveur web lance l'interpréteur PHP exécutant ainsi le script PHP.

Les commandes figurant dans la page sont interprétées et le résultat prend la forme d'un document publié à la place du code source. À l'issue de cette phase de traduction, la page modifiée est envoyée au client pour y être affichée par le navigateur.

Figure 1-3  
Fonctionnement  
de PHP



Le serveur web reconnaît à l'extension des fichiers, différente de celle des pages HTML simples, si le document appelé par le client comporte du code PHP. L'extension utilisée par les pages PHP peut être définie individuellement dans le fichier de configuration du serveur web. Les extensions courantes pour les pages PHP sont .php et .php5 ; nous utiliserons l'extension .php afin d'assurer une compatibilité avec toutes les versions.

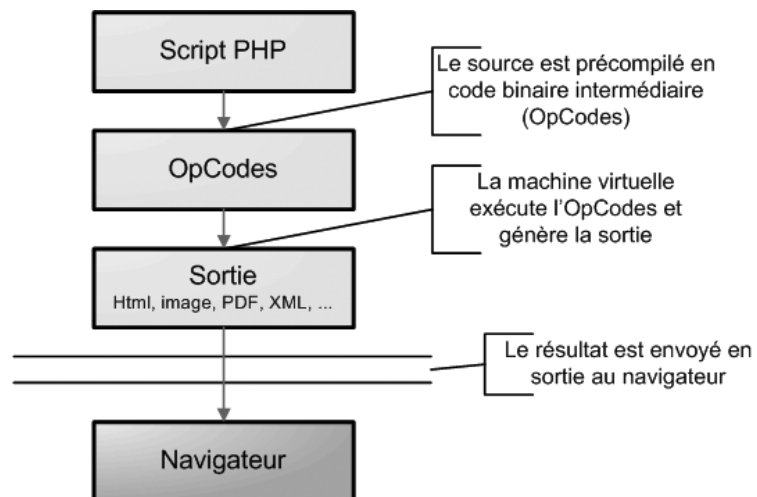
### La machine virtuelle de PHP

Le cœur de PHP 5 est basé sur une machine virtuelle. Les concepts sont les mêmes que pour Java et .Net. Un précompilateur compile le code source en *byte code* (code intermédiaire), puis l'envoie à la machine virtuelle pour exécution.

Cette architecture permet d'ajouter des outils d'optimisation à l'exécution (cache de code), qui divisent souvent par trois le temps d'affichage d'une page.

PHP 5 propose enfin une API qui permet d'étendre ses fonctionnalités au travers de modules additionnels. Ces modules permettent par exemple de se connecter à une base de données ou à un annuaire LDAP, d'exécuter des composants COM ou Java, de dialoguer en Soap avec des services web, etc.

Figure 1-4  
Fonctionnement de  
la machine virtuelle



## PHP en France et dans le monde

LAMP (*Linux Apache MySQL PHP*) est la première plate-forme web dans le monde.

Apache est le serveur le plus utilisé sur Internet avec plus de 60 % de parts de marché, suivi de loin par le serveur IIS de Microsoft, qui totalise aux environs de 30 % de parts de marché (chiffres de mars 2007, source Netcraft).

On trouve sur le site de PHP des statistiques d'utilisation à l'adresse suivante :

<http://www.php.net/usage.php>

En mars 2007, on comptait plus de 20 millions de domaines utilisant PHP. La figure 1-5 nous présente l'évolution de l'utilisation de PHP dans le temps..

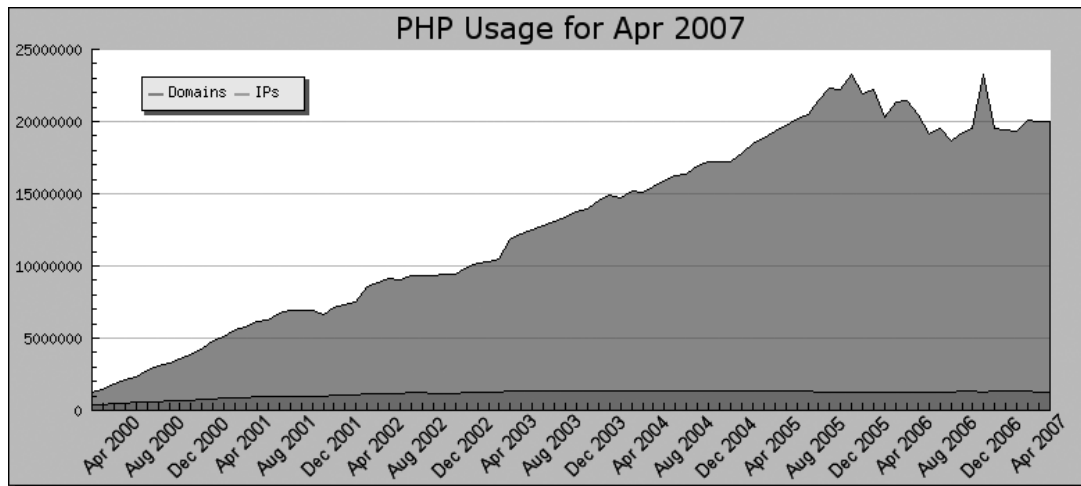


Figure 1-5

Statistiques d'utilisation de PHP

### Les chiffres d'utilisation en France

L'Afup (Association française des utilisateurs de PHP) nous présente un tableau récapitulant les technologies utilisées sur les dix sites provoquant le plus de trafic en France, selon un classement de Jupiter MMXI (voir tableau 1-1).

Bien entendu, compte tenu du trafic et des nombreux services proposés par ces sites, il n'est pas rare que plusieurs serveurs et logiciels gèrent les différents services. Cependant, cette étude nous montre que les principaux sites français en termes de volumétrie utilisent PHP.

Tableau 1-1 Chiffres d'utilisation du PHP en France

	Site Internet	Technologie utilisée
1	<i>Wanadoo.fr</i>	PHP
2	<i>Lycos</i>	PHP
3	<i>Free.fr</i>	PHP
4	<i>Msn.fr</i>	Microsoft/ASP
5	<i>Tiscali.fr</i>	PHP
6	<i>Yahoo.fr</i>	PHP
7	<i>Microsoft.fr</i>	Microsoft/ASP
8	AOL	Confidentiel
9	<i>Google.fr</i>	Propriétaire
10	<i>Voila.fr</i>	PHP

Vous trouverez sur le site de l'Afup ce classement réactualisé, ainsi que la méthode employée pour connaître les technologies utilisées par le serveur.

## La communauté française

La France est l'un des acteurs les plus prolifiques sur la scène internationale concernant PHP. Parmi les fers de lance, on compte **Wampserver**, développé par Romain Bourdon, qui permet en quelques clics de souris d'installer Apache, PHP et MySQL sur Windows. Wampserver dispose d'un système d'add-on qui permet, entre autres, de switcher de PHP 4 à PHP 5 en un clic de souris (idéal pour tester vos applications PHP 4 en PHP 5). Le logiciel **SPIP** développé par arno, l'excellente bibliothèque **FPDF**, permettant de créer du PDF, développée par Olivier Plathey, et **PHPedit**, géré par Sébastien Hordeaux, font aussi partie des références. L'un des frameworks référence « **symfony** » est également issu du travail du français Fabien Potencier. Emmanuel Faivre, Laurent Abbal et Thierry Murail sont les créateurs d'**EasyPHP**, un auto-installeur célèbre. N'oublions pas également Vincent Pontier qui est le créateur de la **mascotte de PHP : l'éléphant**.



Figure 1-6

Les principaux outils français

Outre ces excellents produits, libres d'utilisation, les Français sont très actifs dans de nombreux projets de développement. Ainsi, la France, tout comme l'Allemagne, fait partie des principaux pays impliqués dans le développement de PHP. Les États-Unis, plus axés vers les technologies propriétaires, commencent à s'y mettre mais restent encore peu présents.

Il en résulte de très nombreuses ressources disponibles gracieusement sur Internet. De nombreux bénévoles mettent à disposition des informations sur tous les aspects de PHP. Nous vous proposons de découvrir au travers des pages suivantes les différents sites français composant la communauté PHP en notre pays.

## Les ressources d'aide francophones

Il existe de nombreux sites traitant de PHP. Nous avons ici essayé de sélectionner les plus représentatifs malgré la difficulté, tant les sites de qualité sont nombreux.

### L'Afup

L'Afup (Association française des utilisateurs de PHP) est une association dont le principal objectif est de promouvoir le langage PHP auprès des professionnels. C'est l'Afup qui organise depuis 2001 le Forum PHP en France (site Internet : <http://www.afup.org>).



The screenshot shows a Mozilla Firefox browser window displaying the website of the Association Française des Utilisateurs de PHP (Afup). The browser's address bar shows the URL [http://afup.org/article.php?id\\_article=264](http://afup.org/article.php?id_article=264). The website features a navigation menu on the left with items like 'Accueil', 'PHP', 'PHP en entreprise', 'Annuaire prestataires', 'Forum PHP', 'Actualités', 'L'AFUP', and 'Espace membres'. The main content area is titled 'Livres Blanc "PHP en entreprise"' and contains the following text:

**L'Association Française des Utilisateurs de PHP ([www.afup.org](http://www.afup.org)) publie la quatrième édition de son livre blanc « PHP en entreprise ». Rédigé par des experts de PHP, ce document fournit aux entreprises une information synthétique sur PHP 5 et son écosystème.**

Ce livre blanc s'adresse aux développeurs, chefs de projets, décideurs et architectes qui souhaitent répondre aux questions suivantes :

- ▶ la plate-forme PHP rivalise-t-elle avec .NET et J2EE ?
- ▶ Quelle est son architecture technique ?
- ▶ Combien d'entreprises l'utilisent-elle ?
- ▶ Peut-on développer des services web et des applications client serveur avec PHP ?
- ▶ Est-il possible d'interfacer SAP et Lotus Notes avec PHP ?
- ▶ Quels sont les projets critiques qui recourent à cette technologie ?
- ▶ etc.

Chiffres clés (25 études Forrester, Gartner, etc. compilées), schémas techniques (2), captures d'écrans (8), exemples de code (8), témoignages d'entreprises (15) : tous les éléments sont réunis pour faire de ce livre blanc un véritable outil de travail.

Sommaire :

- ▶ Fiche d'identité de PHP

On the right side of the page, there is a section titled 'Ile font confiance à PHP' featuring logos for Philips, IBM, and the French Ministry of Budget and Reform of the State. At the bottom right, there are logos for SUGARCRM and flickr.

Figure 1-7

L'Association française des utilisateurs de PHP

## Utilité du site

Vous trouverez de nombreux retours d'expérience, chiffres et conseils sur l'utilisation de PHP. L'objectif est de vous donner les outils pour vendre PHP à vos clients.

## Conseil

Téléchargez le livre blanc de l'Afup. Il contient de nombreuses informations sur PHP, son utilisation, des retours d'expérience, etc.

Inscrivez-vous comme membre et participez au développement et à la promotion du PHP en France.

## PHPFrance.com

PHPFrance est l'espace avec lequel de nombreux développeurs PHP d'aujourd'hui se sont formés il y a quelques années. De nos jours, le forum est très actif et peu de questions demeurent longtemps sans réponse. Un espace contenant des cours est extrêmement pratique (site Internet : <http://www.phpfrance.com>).

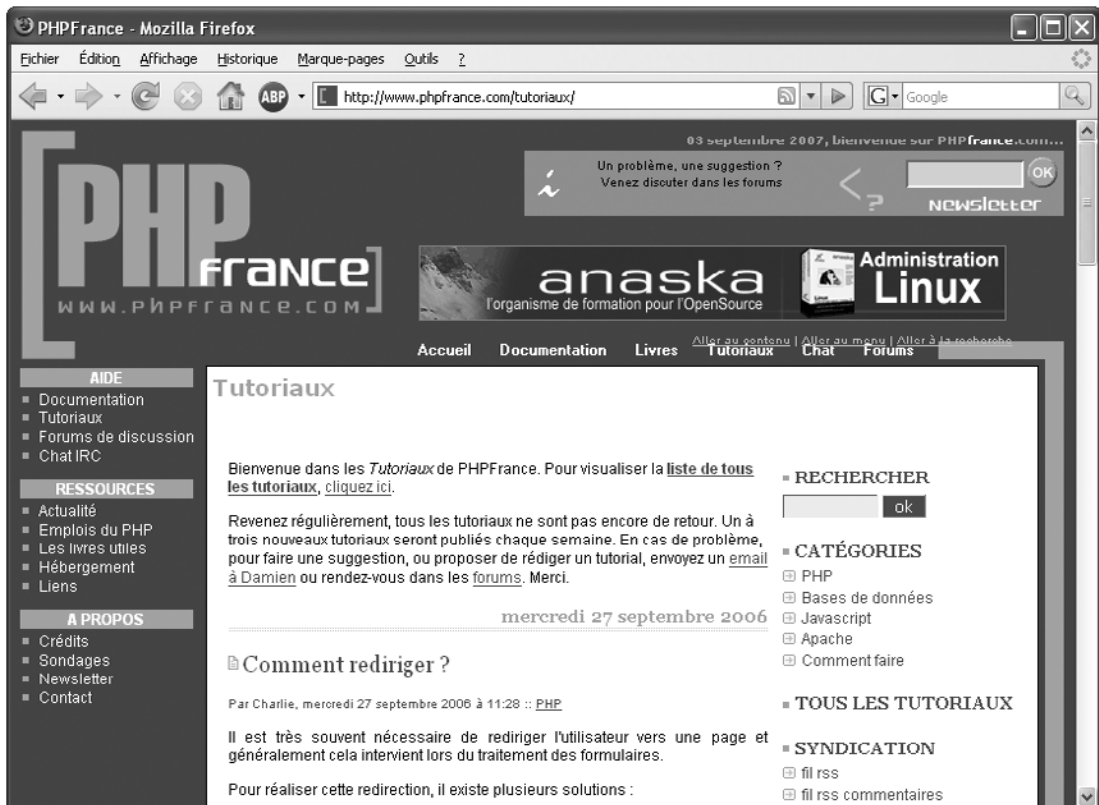


Figure 1-8  
PHPFrance



## Utilité du site

PHPFrance propose de nombreux articles sur l'utilisation de PHP. Vous trouverez également un forum à l'activité débordante où peu de questions restent sans réponse. Accessoirement, un salon IRC (*Internet Relay Chat*) est associé au site : #phpfrance sur le réseau Undernet.

## Conseil

Si vous cherchez un développeur PHP ou un emploi sur PHP, allez sur la rubrique nommée « emplois du PHP », vous y trouverez des informations intéressantes.

Consultez le salon IRC #phpfrance sur le réseau Undernet pour retrouver en direct des passionnés de PHP.

## PHPDebutant.org

Apprendre le PHP vous semble difficile ? Venez sur PHPDebutant.org pour lire les articles sur l'apprentissage de PHP. Ce site extrêmement bien fait comblera les débutants en leur permettant de faire leurs premières passes d'armes en PHP (site Internet : <http://www.phpdebutant.org>).

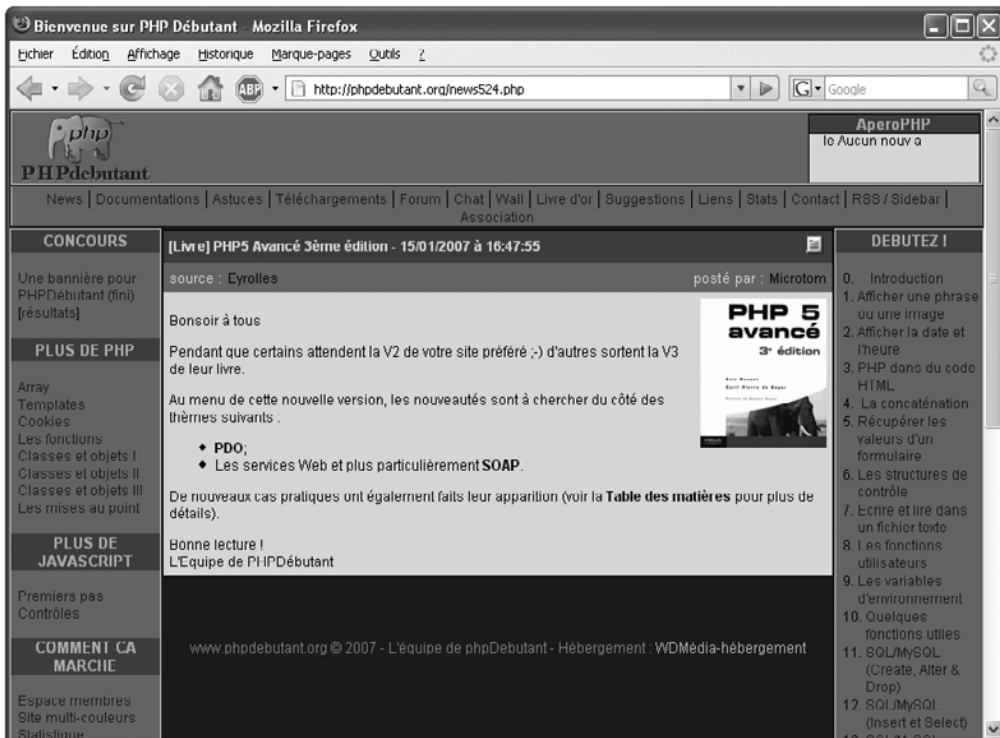


Figure 1-9

PHPDebutant

## Utilité du site

Apprendre PHP vous semblera beaucoup plus facile avec cette aide.

## PHPIndex.com

PHPIndex est l'un des sites pionniers français sur le PHP. Lancé en novembre 1999, ce portail propose de nombreuses ressources et informations sur le PHP. Cet espace s'adresse aux développeurs confirmés qui souhaitent se tenir au courant sur des sujets pointus (site Internet : <http://www.phpindex.com>).

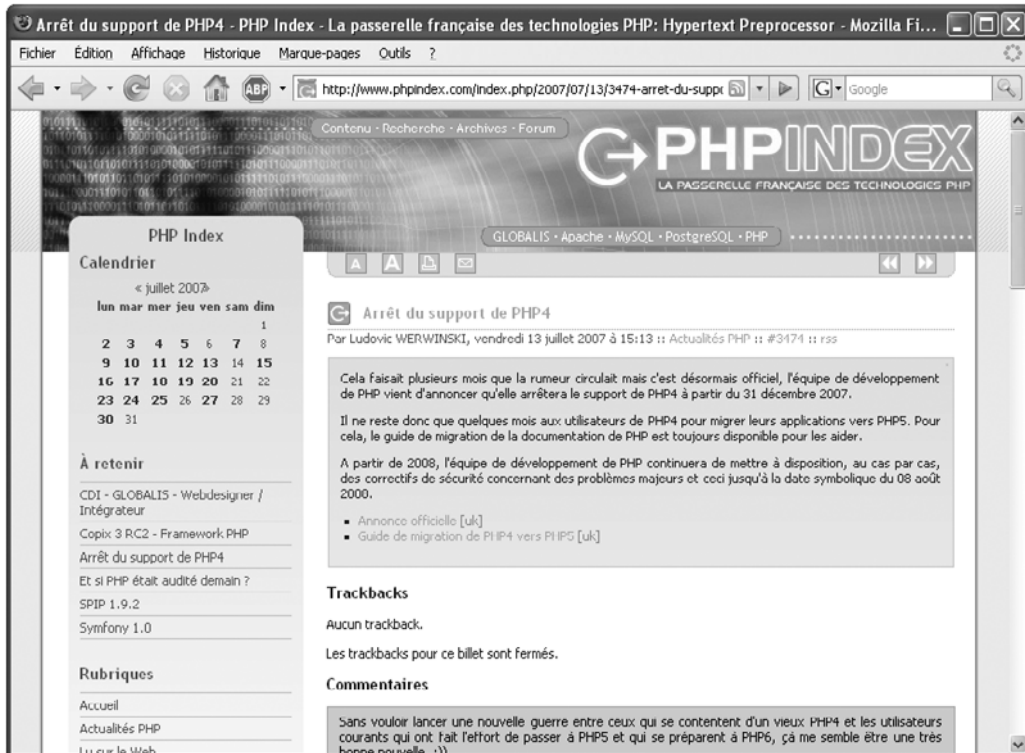


Figure 1-10  
PHPIndex

## Utilité du site

Vous trouverez de nombreux liens vers des articles et des cours sur PHP. Les actualités sont intéressantes et généralement orientées professionnels.

## Conseil

Si vous cherchez un développeur PHP ou un emploi sur PHP, allez sur la rubrique « jobs », vous y trouverez des informations intéressantes.

## PHPTeam.net

PHPTeam.net est un site de contenu. Il propose des articles à destination des développeurs expérimentés, plutôt que pour les débutants. On notera par exemple une introduction à PHP/GTK, un article sur la production de documents PDF ou un article sur les interactions entre PHP et JAVA (site Internet : <http://www.phpteam.net>).

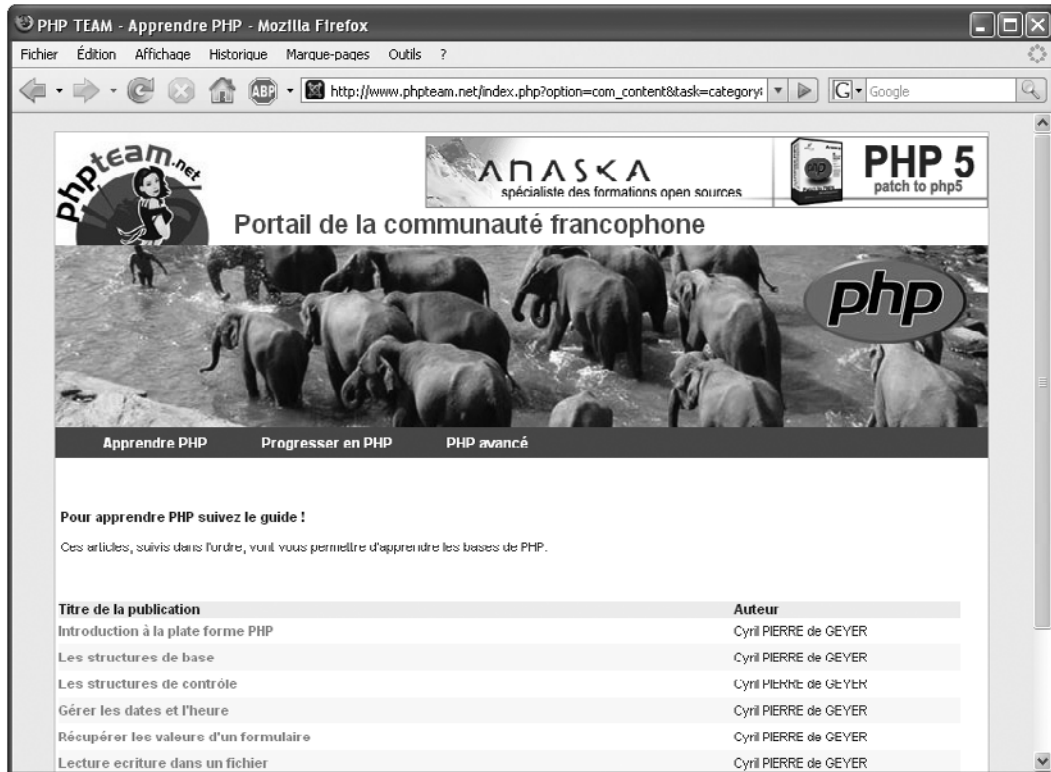


Figure 1-11  
PHPTeam

### Utilité du site

Ce site vous permettra principalement de progresser en PHP. Il constitue un apport supplémentaire aux sites anglophones dédiés au PHP avancé.

## Nexen.net

Nexen.net est l'un des plus anciens sites français consacré au PHP. Depuis l'origine, Nexen participe à la réalisation des documentations PHP et MySQL en français : elles sont disponibles en téléchargement, fréquemment remises à jour, et disposent d'un moteur de recherche perfectionné. Nexen.net est un service édité par Nexenservices, la

société d'hébergement spécialisée sur la plate-forme PHP/MySQL (site Internet : <http://www.nexen.net>).

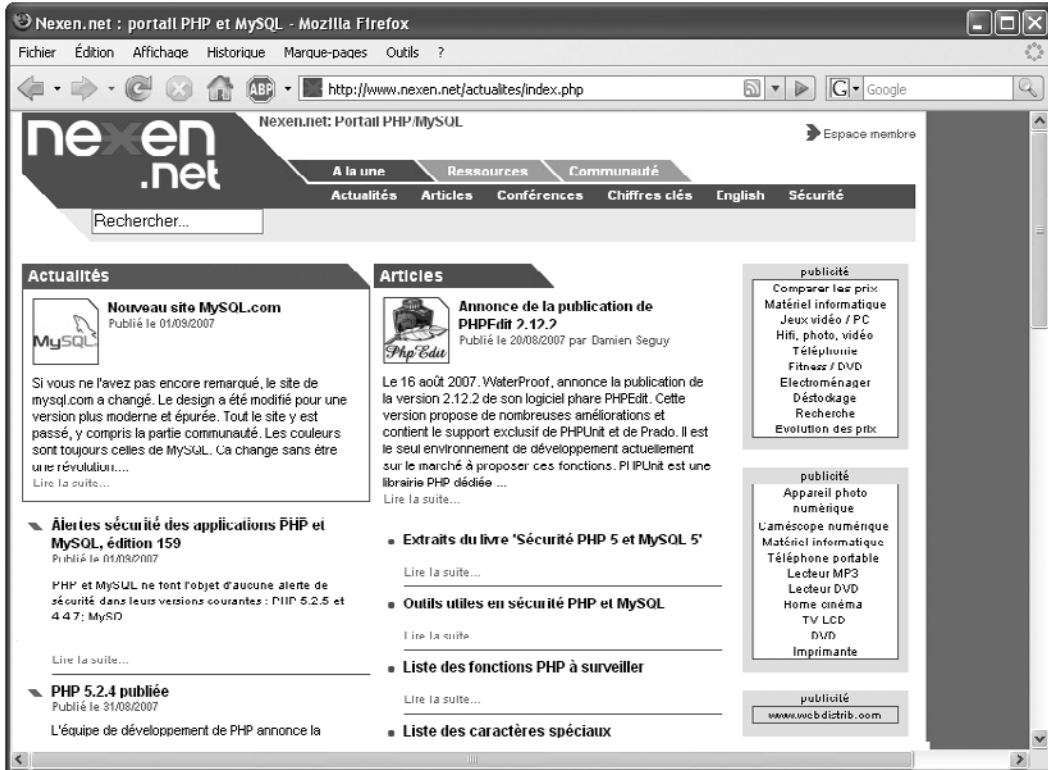


Figure 1-12

Nexen

## Utilité du site

Les nouvelles vous permettent de suivre les actualités mondiales sur PHP et MySQL. Ces nouvelles sont aussi disponibles sous forme de lettre hebdomadaire. Le système est clair et souvent mis à jour. Une bibliothèque de scripts vous permet également de gagner beaucoup de temps dans la réalisation de vos projets.

## Conseil

Inscrivez-vous à la lettre hebdomadaire pour être informé des principales actualités de PHP.

## PHPScripts-fr.net

PHPScripts offre de nombreuses ressources sur le PHP. Son principal atout est son annuaire de scripts. Vous y trouverez des ressources dans quasi tous les domaines : administration de base de données, agenda, annuaire, authentification, bannières, etc. Vous trouverez également une base de données d'articles et des portions de code (site Internet : <http://www.phpscripts-fr.net>).



Figure 1-13

PHPScripts

### Utilité du site

Inutile de réinventer la roue à chaque besoin. Commencez par consulter les scripts Open Source existants. Les scripts présents sur PHPScripts sont notés et commentés par les utilisateurs, ce qui vous permet de choisir parmi la multitude de ressources disponibles.

### PHP Solutions

*PHP Solutions* est un magazine papier dédié à PHP et MySQL. Il rassemble de nombreux articles intéressants en français. D'origine polonaise, le magazine est traduit dans de nombreuses langues (site Internet : <http://www.phpsolmag.org>).

Figure 1-14  
PHP Solutions



Utilité du magazine

PHP Solutions a l'avantage d'être édité sur papier. Les articles sont dans l'air du temps et bien mis en page.

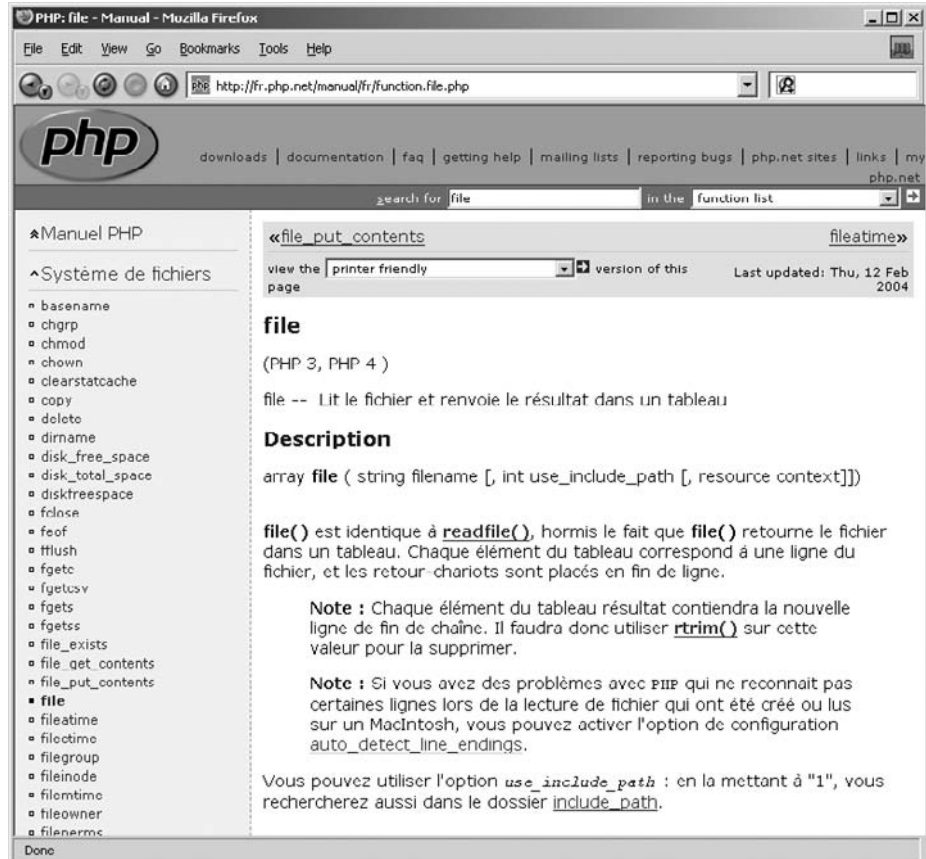
## Les ressources d'aide anglophones

### Le site référence PHP

Le site le plus important est le site de PHP lui-même, car il contient la documentation et de nombreuses informations. On notera qu'il existe des miroirs français permettant de disposer d'une bonne rapidité. Le site vous propose automatiquement le plus d'informations

possible en français grâce à la détection automatique de votre langue (site Internet : <http://www.php.net>, miroir français : <http://fr.php.net>).

Figure 1-15  
Le site PHP.net



### Utilité du site

Le site propose un accès à la documentation en ligne. On note également le moteur de recherche des fonctions très utile.

### Conseil

Utilisez le moteur de recherche des fonctions. Si vous connaissez le C, indiquez le nom en C de la fonction que vous recherchez. En PHP, son nom est souvent assez proche. Quand vous avez trouvé votre fonction et sa définition comme sur la figure 1-17, consultez les fonctions dans l'espace de gauche, elles concernent toutes le même sujet et peuvent vous permettre de progresser.

## MySQL.com

Sur le site de MySQL existe une section dédiée aux développeurs. On y trouve de nombreuses ressources dont des programmes, des articles et des conseils pour optimiser vos applications (site Internet : <http://www.mysql.com>).

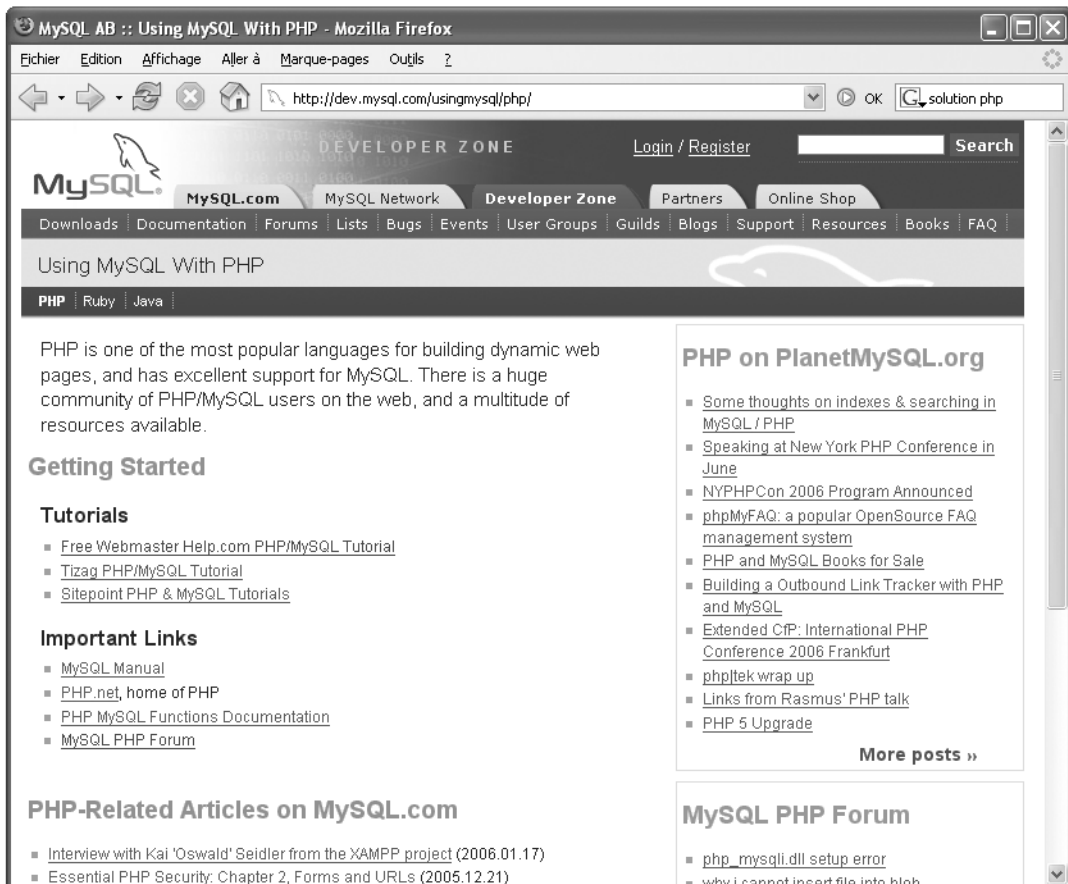


Figure 1-16

*MySQL.com*

## Utilité du site

MySQL.com vous propose de nombreuses ressources pour améliorer vos bases de données. Regardez du côté des programmes proposés pour manipuler vos bases et même pour vous aider à migrer vers MySQL.





# 2

## Installer et configurer PHP 5

---

PHP peut servir de différentes manières. La plus commune est le mode client/serveur, c'est-à-dire associé à un serveur web. Nous allons détailler dans ce chapitre l'installation d'une telle configuration. PHP peut être couplé avec de nombreux serveurs web dont les deux principaux sont Apache et IIS. Apache étant largement prédominant sur ce marché, nous allons nous intéresser à son cas en particulier. On notera que PHP est souvent associé à Linux, à Apache et à MySQL : la dénomination LAMP caractérise cette association. Nous verrons donc dans un premier temps la procédure à suivre pour installer PHP 5 sur une station MS Windows (a priori pour la période de développement), puis sur une plate-forme Linux (pour la production).

Il existe des outils permettant d'installer une plate-forme PHP complète, mais nous vous conseillons d'effectuer au moins une fois une installation manuelle de l'ensemble afin d'en comprendre le fonctionnement et de bien en saisir les subtilités.

### Migration vers PHP 5

La compatibilité avec la version précédente a été une des préoccupations majeures durant le développement de PHP 5. Une grande majorité des applications devraient pouvoir être exécutées sur PHP 5 sans dommages, ou ne nécessiter que des modifications mineures.

Il existe cependant quelques différences et nous allons essayer de les résumer ici pour vous permettre une migration simple.

## Incompatibilités

Voici la liste des principales incompatibilités avec les versions antérieures de PHP :

- Les fonctions `strpos()` et `stripos()` utilisent maintenant tous les caractères fournis dans le paramètre de recherche et plus uniquement le premier. Consultez le chapitre 5 sur les chaînes de caractères pour plus d'informations.
- Les classes doivent être déclarées avant d'être utilisées. Vous ne pouvez plus faire vos déclarations à la fin des fichiers.
- La fonction `array_merge()` a été modifiée pour n'accepter que des tableaux. Pour chaque variable passée en paramètre autre qu'un tableau, un message `E_WARNING` sera envoyé. Consultez le chapitre 6 sur les fonctions de tableaux pour plus d'informations.
- La superglobale `$_SERVER` est maintenant renseignée avec `argc` and `argv` si la directive de configuration `variables_order` de votre `php.ini` inclut « S ».
- Les objets sont affectés et transmis par référence et non plus par valeur. Vous trouverez les détails sur ce fonctionnement au chapitre 12.

## PHP en ligne de commande et en CGI

Les exécutables de PHP 5 ont été réorganisés. Sous Microsoft Windows la version CGI s'appelle maintenant `php-cgi.exe` et non plus `php.exe`. La version CLI (PHP en ligne de commande) se trouve maintenant dans le répertoire principal de PHP, sous le nom `php.exe` (elle était auparavant dans un répertoire dédié). Un nouvel exécutable a également été introduit avec PHP 5 : `php-win.exe`, qui permet d'utiliser PHP en ligne de commande sans faire apparaître une fenêtre en déclenchant l'affichage.

Sur les plates-formes de type Unix, l'exécutable nommé `php` sera soit la version CGI, soit la version CLI, selon ce que vous utiliserez comme paramètres de configuration.

### Directive de configuration pour compatibilité

Certains changements comme le passage des objets par référence peuvent être inversés *via* une directive de configuration. En activant `zend.zel_compatibility_mode` dans votre `php.ini`, vous demanderez à PHP de se comporter le plus possible comme PHP 4. Seules les fonctionnalités du langage seront impactées, non les fonctions.

Cette option est faite pour pouvoir exécuter une application PHP 4 : il est déconseillé de l'utiliser si vous avez du code PHP 5.

## Modes d'installation

L'intégration de PHP dans le serveur Apache peut se faire de deux façons : en tant que module Apache ou via l'interface CGI (*Common Gateway Interface*).

### CGI

Dans une installation de type CGI, PHP sera installé de façon autonome. Lorsque notre serveur Apache aura besoin de traiter un document PHP, il lancera le programme PHP de

façon externe et en récupérera le résultat. À chaque page interprétée, vous aurez donc le lancement et l'arrêt d'un processus (celui de PHP) sur votre machine.

## Modules

Dans une installation en module Apache, PHP sera directement intégré dans le serveur web. Le code PHP sera alors exécuté directement par le processus du serveur web. Cela signifie aussi que PHP ne sera chargé qu'une seule fois au lancement du serveur web et que, pour modifier une option de configuration ou pour ajouter un module à PHP, il faudra redémarrer entièrement le serveur web.

L'installation en module Apache est celle que nous vous conseillons si vous n'avez pas de besoins contraires. Elle amène performances (pas de processus lancé à chaque requête) et souplesse (comme nous le verrons, elle rend possible des modifications de configuration uniquement pour certains scripts).

## Installer PHP 5 sous MS-Windows

Vous pouvez utiliser PHP 5 sur de nombreuses plates-formes. Le système d'exploitation le plus utilisé sur les postes de travail restant MS Windows, il est bien entendu possible d'y installer une plate-forme WAMP (*Windows Apache MySQL PHP*).

### Installation automatique

Il existe plusieurs distributions permettant d'installer en quelques clics une plate-forme de développement WAMP. La plus intéressante est WampServer (<http://www.wampserver.com>). Celle-ci permet en plus de disposer d'un environnement Windows Apache MySQL PHP, et de passer de PHP 4 à PHP 5 en un clic de souris.

**Figure 2-1**  
*Menu de  
WampServer*



## Installation manuelle

### Installation de PHP 5

Tout d'abord, il convient de télécharger la dernière version stable de PHP 5. Vous pourrez la trouver sur le site <http://www.php.net>.

PHP 5 pour Windows est disponible sous deux formes : un fichier ZIP ou une installation automatisée Win32. Nous allons utiliser la version ZIP, car la version automatisée ne peut s'installer qu'en CGI. De plus, l'installation automatisée va essayer de paramétrer PHP 5 pour fonctionner avec notre serveur, tâche que l'on souhaite effectuer manuellement afin de bien en comprendre le fonctionnement.

Une fois l'archive téléchargée, il ne vous reste plus qu'à la décompresser. Vous obtenez alors un répertoire nommé `php5.XX-win32`. Renommez ce répertoire en tant que `php` et copiez-le à la racine de votre disque (C:\). Vous devez alors avoir un répertoire `C:\php` contenant le fichier `php.exe` ainsi que d'autres fichiers et répertoires.

### Copie du fichier de configuration

Pour fonctionner, PHP utilise un fichier de configuration, le fichier `php.ini`. Celui-ci permet de préciser la plupart des aspects du mode de fonctionnement de PHP. Il permettra, entre autres, de paramétrer certaines fonctionnalités telles que les sessions ou l'envoi de fichiers, d'ajouter des modules à PHP, etc.

Pour être utilisé, ce fichier doit être lu par PHP. Il est donc nécessaire de le positionner dans un répertoire faisant partie du chemin de recherche de notre système (variable d'environnement `PATH`). La solution la plus simple consiste donc à positionner notre fichier `php.ini` dans notre répertoire `c:\windows` (ou `c:\winnt`).

Deux fichiers de configuration sont fournis par défaut avec `php` : `php.ini-dist` et `php.ini-recommended`.

Le premier est un modèle par défaut ; c'est celui que PHP utilise quand il ne trouve pas le fichier de configuration. Ces valeurs sont dites conservatives car elles encouragent la compatibilité avec les anciennes configurations. Le second est, quant à lui, une configuration plus stricte, recommandée par l'équipe de développement de PHP. Nous allons utiliser cette dernière pour faire nos premiers pas.

#### Note sur le fichier `php.ini`

Longtemps, les développements ont été réalisés avec le fichier `php.ini-dist`. De ce fait, il peut arriver qu'à l'installation de logiciels ils ne fonctionnent pas si vous utilisez le fichier `php.ini-recommended`. Utilisez le `-recommended` en priorité, et n'envisagez le `-dist` que pour compatibilité avec les anciens scripts qui le nécessitent.

Nous allons donc copier le fichier `php.ini-recommended` (ou `php.ini-dist` pour compatibilité) dans notre répertoire `C:\windows` et le renommer en `php.ini`. Toutes les fois où l'on voudra modifier la configuration de PHP, il suffira de venir éditer ce fichier. Faites attention cependant, car dans cette configuration la directive `display_errors` est à `0ff`, ce qui veut dire que vous ne verrez pas les erreurs engendrées par vos scripts à l'écran (elles

sont stockées dans un fichier de *log*). Pour la phase de développement, vous pouvez modifier cette valeur dans le fichier et la mettre à 0n.

### Utilisation de ligne de commande

L'installation de PHP 5 est maintenant terminée, il ne nous reste plus qu'à vérifier que PHP fonctionne bien. Pour cela, nous allons exécuter un script en mode de commande et vérifier que PHP renvoie bien un résultat.

Nous allons donc commencer par créer un fichier `test.php` dans le répertoire `C:\test`. Éditez ce fichier avec un éditeur de texte quelconque (le Bloc-notes Windows par exemple) et copiez-y les lignes ci-dessous :

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

#### Explication

Cette simple commande permet d'afficher la configuration de PHP. Elle renvoie une page au format HTML contenant toutes les informations de configuration de notre installation de PHP.

Nous allons maintenant exécuter ce script avec PHP. Le résultat retourné sera affiché directement dans notre fenêtre DOS. Il ne sera donc pas lisible (suite de balises HTML et de textes) mais nous permettra de vérifier simplement que le script a bien fonctionné.

Pour cela, commencez par ouvrir une fenêtre DOS (menu Démarrer>Exécuter, puis tapez `cmd`).

Il ne vous reste plus qu'à lancer `php-cgi.exe` en lui passant en paramètre le script qu'il doit exécuter :

```
c:\php\php-cgi.exe c:\test\test.php
```

Si vous avez bien suivi les instructions ci-dessus, des lignes contenant du code HTML devraient se mettre à défiler rapidement, la dernière balise affichée devant être `</html>`.

Figure 2-2

Utilisation de PHP  
en ligne de commande

```
C:\WINDOWS\system32\cmd.exe
<tr><td class="e">_ENV["USERDOMAIN"]</td><td class="v">ANASKA</td></tr>
<tr><td class="e">_ENV["USERNAME"]</td><td class="v">Administrateur</td></tr>
<tr><td class="e">_ENV["USERPROFILE"]</td><td class="v">C:\Documents and Settings\Administrateur</td></tr>
</table><hr />
<tr><td class="e">_ENV["windir"]</td><td class="v">C:\WINDOWS</td></tr>
</table><hr />
<h2>PHP License</h2>
<table border="0" cellpadding="3" width="600">
<tr class="v"><td>
<p>
This program is free software; you can redistribute it and/or modify it under the terms of the PHP License as published by the PHP Group and included in the distribution in the file: LICENSE
</p>
<p>
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
</p>
<p>
IF you did not receive a copy of the PHP license, or have any questions about PHP licensing, please contact license@php.net.
</p>
</td></tr>
</table><hr />
</div></body></html>
C:\php5>
```

## Installation d'Apache

Maintenant que PHP est installé sur notre serveur, nous allons pouvoir installer notre serveur web, en l'occurrence Apache, et le configurer pour fonctionner avec PHP.

### Note sur les versions d'Apache

Vous pourrez trouver dans la documentation de PHP et sur les sites des développeurs que la version 2 d'Apache n'est pas supportée et qu'elle est déconseillée. En réalité seules certaines configurations bien spécifiques d'Apache 2 posent problèmes.

La version 2 d'Apache permet d'utiliser plusieurs types de moteurs internes. Plusieurs de ces moteurs sont dits « multi-thread » et gèrent plusieurs fils d'exécution en parallèle dans un même processus système. Certaines extensions PHP et certaines bibliothèques utilisées par PHP ne sont pas compatibles avec ce fonctionnement et risquent de provoquer des bogues importants. Ces configurations sont donc à éviter et déconseillées par l'équipe de PHP.

Le moteur nommé « prefork » peut être utilisé sans risque. Il s'agit du moteur historique d'Apache, qui était déjà utilisé sur la version 1.3. Sous Linux, la plupart des configurations Apache par défaut utilisent le modèle prefork. Dans cette configuration, Apache 2 ne pose aucun problèmes vis-à-vis de PHP.

La première tâche consiste à télécharger les fichiers d'installation de notre serveur. Pour cela, rendez-vous sur le site <http://httpd.apache.org>.

Le fichier d'installation se présente sous la forme d'un exécutable d'installation automatisée. Il suffit de double-cliquer dessus et de suivre les instructions pour avoir une installation par défaut d'Apache. Les seules informations à fournir concerneront le (ou les) nom(s) de domaine(s) désignant votre serveur. Si vous n'en avez pas, il vous suffit de mettre `localhost`. Si vous utilisez Windows XP, 2000 ou NT, le programme d'installation vous demandera également si vous souhaitez qu'Apache soit installé en tant que service. En choisissant d'installer Apache en tant que service, celui-ci sera automatiquement lancé au démarrage de l'ordinateur et son exécution sera surveillée par le système. Pour le gérer (arrêter, démarrer), il vous faudra atteindre la fenêtre de gestion de service de Windows se trouvant dans le panneau de configuration.

Si vous utilisez Windows 98 ou Millenium (Windows 95 n'étant officiellement plus reconnu par PHP 5) ou si vous décidez de ne pas installer Apache en service, votre serveur sera installé en tant que simple programme. Des liens dans le menu Démarrer vous permettront de lancer et d'arrêter votre serveur.

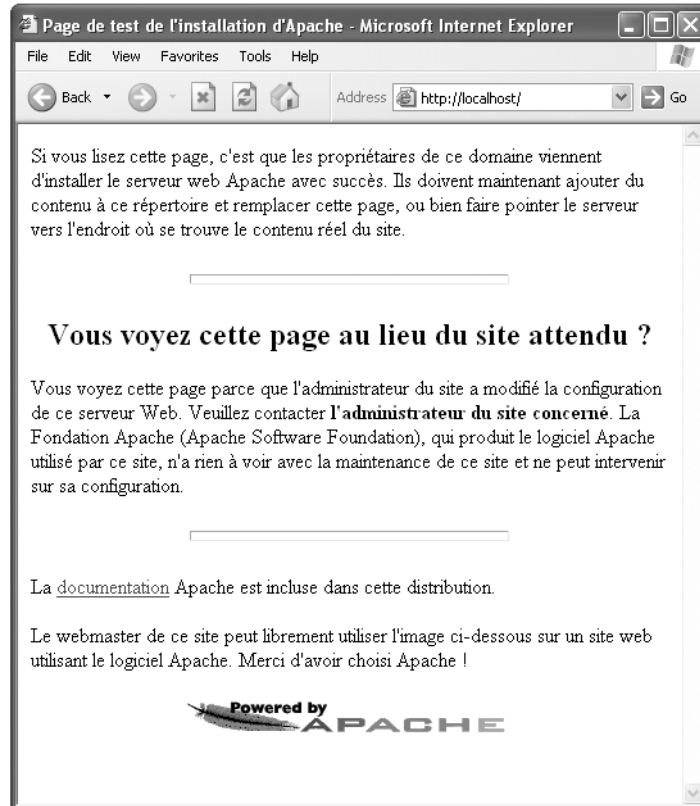
Votre serveur est maintenant installé, il ne vous reste plus qu'à vérifier qu'il fonctionne. Pour cela, lancez votre serveur et ouvrez une fenêtre de navigateur (Microsoft Internet Explorer, Mozilla ou Opera par exemple) sur l'adresse ou <http://localhost/>.

### Note

L'adresse `localhost` correspond à la boucle locale. Pour tout ordinateur, cette adresse pointe sur lui-même.

Si vous avez bien suivi les instructions ci-dessus, la page de bienvenue de votre serveur Apache doit s'afficher dans votre navigateur (voir figure 2-3).

**Figure 2-3**  
*Page d'accueil  
d'Apache*



Notre serveur est maintenant installé. Par défaut, Apache ne reconnaît pas les fichiers PHP et ne sait pas comment les traiter. Il nous reste donc à lui fournir les directives nécessaires. Celles-ci devront être ajoutées dans le fichier de configuration d'Apache, `httpd.conf`, qui se trouve dans le répertoire `conf` de votre installation d'Apache.

### Installer PHP en CGI

L'installation de PHP en tant que CGI est la plus simple. Trois directives sont nécessaires dans le fichier `httpd.conf` :

```
AddType application/x-httpd-php .php
```

#### Note

Cette directive doit être mise après les déclarations de modules.



Cette première directive permet de lier l'application `x-httpd-php` (autrement dit notre programme `php`) aux fichiers ayant l'extension `.php`. Apache saura ainsi comment traiter ce type de fichiers.

De la même façon, il est possible de définir nos propres extensions pour nos fichiers `php`. Si vous redéfinissez l'extension des fichiers HTML, vous pourrez y insérer du code PHP sans avoir à les renommer (vous utiliserez cependant inutilement des ressources processeur en analysant des fichiers qui ne contiennent pas de PHP). Vous pourriez également créer des fichiers avec une extension au nom de votre société, qui seraient traités comme des fichiers PHP :

```
AddType application/x-httpd-php .html
AddType application/x-httpd-php .myext
AddType application/x-httpd-php .anaska
```

Les deux directives qui suivent permettent à Apache de connaître la localisation du CGI PHP :

```
# PHP a été installé dans le répertoire c:\php\
ScriptAlias /php/ "c:\php\"
Action application/x-httpd-php "/php/php-cgi.exe"
```

Dans l'absolu, seule la troisième ligne fournit une information pertinente à Apache. La directive `Action` permet de lier l'application `x-httpd-php` à notre exécutable `c:\php\php.exe`. Le `ScriptAlias` permet quant à lui simplement de définir un alias `/php/` pour le chemin `c:\php\`.

Une fois ces quelques lignes ajoutées, il ne nous reste plus qu'à enregistrer le fichier `httpd.conf` et à redémarrer notre serveur. Apache sait maintenant que les fichiers PHP doivent être traités par l'exécutable CGI PHP ; il sait également où celui-ci se trouve.

### Installer PHP en module

Le deuxième type d'installation est l'installation en module. Cette installation est un peu plus complexe à mettre en œuvre mais offre des performances plus intéressantes.

Tout d'abord, nous allons devoir fournir à Apache la DLL lui permettant d'accéder et d'utiliser le module PHP. Ce fichier est fourni avec PHP et s'appelle `php5ts.dll`. Repérez ce fichier dans la racine de votre répertoire PHP et copiez-le dans la racine de votre répertoire Apache (à côté du fichier `Apache.exe`). Vérifiez également que la DLL `php5apache.dll` est bien présente dans le répertoire de votre installation de PHP.

Il faut maintenant ajouter les directives nécessaires au fonctionnement de PHP en tant que module. De la même façon que pour l'installation en CGI, nous allons éditer le fichier `httpd.conf` et y ajouter la directive permettant d'associer l'extension `.php` à l'utilisation du module PHP 5 :

```
AddType application/x-httpd-php .php
```

Nous ajoutons ensuite deux directives permettant de charger le module PHP5 au démarrage et de l'exécuter :

```
LoadModule php5_module c:\php\php5apache.dll
AddModule mod_php5.c
```

Une fois ces deux lignes ajoutées, refermez votre fichier `httpd.conf` et relancez votre serveur.

Vérifier que l'installation s'est bien déroulée

Pour vérifier qu'une installation s'est bien passée, on affiche généralement un `phpinfo()`. Pour cela, on copie le fichier `c:\test\test.php` que nous avons créé précédemment, dans l'arborescence de notre serveur web. Par défaut, la racine de notre serveur est le répertoire `htdocs` se trouvant dans l'installation d'Apache. C'est dans ce répertoire que nous allons copier notre fichier. Ouvrez un navigateur Internet à l'adresse `http://localhost/test.php`. Si vous avez bien suivi nos instructions, une page donnant des informations sur PHP et votre configuration doit s'afficher. La figure 2-4 montre un exemple de page que vous devriez obtenir.

Figure 2-4  
*phpinfo()*

PHP API	20031224
PHP Extension	20020429
Zend Extension	90021012
Debug Build	no
Thread Safety	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp, compress.zlib

This program makes use of the Zend Scripting Language Engine:  
Zend Engine v2.0.0-dev, Copyright (c) 1998-2004 Zend Technologies

Powered By

## PHP Credits

## Configuration

### PHP Core

Directive	Local Value	Master Value
<code>allow_call_time_pass_reference</code>	On	On
<code>allow_url_fopen</code>	On	On
<code>always_populate_raw_post_data</code>	Off	Off

## Configuration du serveur Apache

Enfin, quel que soit votre mode d'installation, vous pouvez modifier la directive `DirectoryIndex` afin que le fichier `index.php` soit affiché par défaut lorsqu'un utilisateur accède à un répertoire de votre arborescence web.

```
DirectoryIndex index.html index.php
```

De même, il sera intéressant de modifier la directive `DocumentRoot`. Celle-ci permet de définir la racine de votre serveur dans votre arborescence de fichiers. Créez, par exemple, un répertoire `c:\www` et modifiez la directive ainsi :

```
DocumentRoot "C:\www"
```

La balise `<Directory>` permet d'associer à une arborescence une série d'options influant sur le comportement du serveur web. Par exemple, vous pouvez autoriser ou non la visualisation des fichiers du répertoire dans le cas où le fichier `index` n'est pas présent.

Pour en revenir à notre installation, il faut donc indiquer les options correspondant à notre racine.

```
#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "C:\www">
    Options Indexes FollowSymLinks MultiViews
```

En général, le plus simple est de reprendre la configuration par défaut qui était appliquée à l'ancienne racine web (`C:\Program Files\Apache Group\Apache\htdocs` pour une installation sous Windows).

### Attention

Il existe plusieurs sections `<Directory>`. Ne modifiez pas celle qui fait référence à la racine de votre système (`<Directory />`).

Le serveur Apache va maintenant distribuer les fichiers se trouvant dans ce répertoire ; c'est donc là qu'il faudra déposer vos sites web.

## Installation de MySQL

Le SGBD MySQL a pris son essor dans le monde du Web en même temps que (et principalement grâce à) PHP. Ces deux produits se sont retrouvés sur le marché au même moment et avec le même objectif principal : offrir une solution alternative gratuite et simple d'accès aux produits commerciaux proposés sur le marché.

PHP et MySQL se sont rapidement trouvés liés, pour devenir le couple incontournable de la création de sites web. C'est sûrement ce qui a poussé l'équipe de développement PHP à intégrer de façon native la prise en charge de MySQL dans la version 4 de PHP.

Ainsi, sur toutes les différentes versions 4 de PHP, il était possible de se connecter à une base de données MySQL sans avoir à ajouter de module spécifique. Ce comportement de

PHP a permis d'établir la plate-forme LAMP (Linux, Apache, MySQL, PHP) comme la solution la plus utilisée sur le Web.

La version 5 de PHP apporte de nombreux changements, dont la suppression de la reconnaissance automatique de MySQL. Cela est principalement dû à deux raisons : des conflits de licences lors de la sortie de PHP 5.0, une volonté affichée du PHPGroup de tendre la main aux autres SGBD et l'intégration native d'une nouvelle base de données par défaut, SQLite. Cela devrait, à terme, pousser les gens à utiliser SQLite pour des sites à charge faible ou moyenne, et installer un module pour le choix d'une base de données spécifique (MySQL, PostgreSQL, Oracle ou autre) pour des besoins plus importants.

La version 5.1 a de plus ajouté une interface unifiée permettant d'utiliser toutes les bases de données de la même façon : PDO. C'est cette interface que nous privilégierons dans l'essentiel de ce livre. Vous retrouverez plus d'informations sur PDO au chapitre 18.

### Activation du module PHP PDO

Ainsi, pour utiliser les connecteurs de bases de données PDO, il est nécessaire d'ajouter un module PHP spécifique. Ce module s'appelle `php_pdo.dll` et est normalement fourni avec PHP dans le sous-répertoire `ext/`.

Si le fichier DLL est bien présent, il vous reste à l'activer en modifiant le fichier de configuration `php.ini`.

Si le répertoire des extensions n'est pas renseigné, précisez-le. Il désigne l'emplacement des DLL d'extension :

```
extension_dir = "c:\php\ext\  
extension=php_pdo.dll
```

### Activer le driver MySQL de PDO

Une fois PDO chargé, il faut charger le driver PDO spécifique à MySQL. Il s'appelle `php_pdo_mysql.dll` et se charge de la même manière, via une directive `extension` du fichier `php.ini` :

```
extension=php_pdo_mysql.dll
```

Il vous faudra aussi copier le fichier `libmysql.dll` du répertoire `dlls/` de votre installation PHP dans un des répertoires système de votre serveur (par exemple `c:\windows\system32\` ou `c:\winnt\system32\`).

### Activer les anciennes extensions mysql et mysqli

Pour garder une compatibilité avec d'anciennes applications n'utilisant pas PDO, vous pouvez avoir à activer une des deux anciennes extensions MySQL. La première s'appelle « `mysql` » et ne peut se connecter qu'aux serveurs MySQL 3.x et 4.0. La seconde s'appelle « `mysqli` » et est apparue avec PHP 5. Elle est peu utilisée mais sait se connecter à toutes les versions de MySQL.

Les deux modules correspondants s'appellent `php_mysql.dll` et `php_mysqli.dll`. Ils peuvent s'activer simultanément :

```
extension_dir = "c:\php\ext\  
extension=php_mysqli.dll  
extension=php_mysql.dll
```

### Installation du serveur MySQL

La dernière étape consiste en l'installation de MySQL. Pour cela, rendez-vous à l'adresse <http://www.mysql.com> pour télécharger la dernière version stable pour Windows. L'installation est automatisée et ne nécessite aucun paramétrage.

Une fois l'installation terminée, allez dans le répertoire `c:\mysql\bin` et lancez l'exécutable `winmysqladmin.exe`. Celui-ci vous permettra de gérer votre base de données et de l'installer en tant que service si vous êtes sous Windows XP, 2000 ou NT.

Il ne reste plus qu'à redémarrer votre serveur Apache pour que PHP prenne en compte les modifications.

Pour vérifier que tout fonctionne bien, la meilleure solution est d'installer le célèbre outil `phpMyAdmin`, qui vous dira tout de suite si votre plate-forme WAMP est prête. Consultez le chapitre 18 pour avoir les détails quant à son installation.

## Installer PHP 5 sous Unix

Nous allons maintenant vous décrire les opérations permettant d'installer PHP et les serveurs associés sur un système Linux (la procédure pour d'autres Unix est globalement la même). Ces descriptions sont volontairement simples pour faciliter la mise en œuvre du point de vue développeur. Pour une utilisation sur un serveur de production, nous vous conseillons vivement de vous référer aux fichiers nommés `INSTALL` dans les sources des différents logiciels et à des ouvrages d'administration Unix. Ces documentations détaillent des questions que nous ne pouvons commenter ici comme la sécurisation de l'installation, les droits d'accès, le partage des ressources ou le *monitoring*.

### Utilisation automatisée

L'installation de PHP sous Linux (et, par extension, sur la plupart des systèmes orientés Unix, Mac OS X compris) est relativement simple. Les seules difficultés que vous pourriez rencontrer sont celles de l'installation des outils de compilation et des bibliothèques utilisées.

Malgré cette facilité de compilation, nous vous conseillons fortement de vous reposer le plus possible sur l'éditeur de votre distribution et d'utiliser les paquets précompilés qu'il vous donne pour Apache, MySQL et PHP. Généralement, les outils d'installation vous permettent de télécharger et d'installer les logiciels et bibliothèques en quelques manipulations.

Sous GNU/Debian Linux vous pouvez lancer l'installation de Apache, MySQL et PHP avec la commande suivante :

```
apt-get install apache mysql mod_php
```

Avec la distribution Linux Mandrake, il faudra utiliser `urpmi` :

```
urpmi apache mysql-server php mod_php
```

Sous RedHat ou Fedora, la commande est similaire mais avec l'outil `yum`. Le nom des paquets peut varier légèrement, mais tous ces outils permettent de faire des recherches dans leur base via des interfaces graphiques si jamais le nom exact vous échappe.

```
yum mod_php apache mysql-server
```

Pour utiliser un module spécifique de PHP, il vous suffira de l'installer avec la même commande. Les paquets des modules PHP sont généralement nommés avec la syntaxe suivante : `php-mysql`, `php-xml`, `php-xsl`, etc.

Utiliser les paquets de votre distribution vous permettra de gagner du temps mais vous offrira aussi une sécurité accrue. Vous aurez la certitude que les configurations par défaut sont étudiées et que votre système pourrait être mis à jour facilement si jamais une faille était découverte dans PHP.

## Installation manuelle d'Apache

L'installation d'Apache à partir des sources comporte énormément de paramètres. Nous nous limiterons à décrire la procédure basique tout en vous donnant des indications pour aller plus loin.

Les sources d'Apache sont disponibles à l'adresse <http://httpd.apache.org/download.cgi>. Vous aurez le choix entre Apache 1.3, 2.0 et 2.2. Les versions récentes ne posent pas de problème particulier si vous utilisez le modèle d'exécution nommé « `prefork` » dans la configuration. C'est le modèle par défaut sur la plupart des installations Linux. Il vous restera à les décompresser dans un répertoire temporaire.

```
cd /tmp
tar xzf apache-1.3.36.tar.gz
cd apache-1.3.36
```

Avant l'étape de compilation, il faut décider quelles sont les options que vous souhaitez utiliser. Vous pouvez lister toutes les options disponibles avec la commande suivante :

```
./configure --help
```

Pour cette description, nous nous contenterons du minimum : déterminer le répertoire destination et compiler la prise en charge des extensions dynamiques (pour pouvoir utiliser le module PHP par la suite). Le répertoire destination est à spécifier après le paramètre `--prefix=`, la prise en charge des extensions dynamiques se demande avec `--enable-module=so`.

```
./configure --prefix=/usr/local/apache --enable-module=so
```

Si la configuration ne vous renvoie pas d'erreur, il est temps de passer à la compilation avec la commande suivante :

```
make
```

Enfin, à la suite de la compilation, vous pouvez demander l'installation des fichiers avec `make install`. Si vous installez Apache dans un répertoire système, vous aurez probablement besoin de vous authentifier en tant qu'utilisateur `root` pour l'installation.

```
make install
```

La suite de la configuration d'Apache se passe de manière similaire à ce qui a été décrit pour Microsoft Windows. D'autres paramètres existent, par exemple pour gérer les droits d'accès, mais vous pouvez les ignorer dans un premier temps si vous utilisez votre plateforme de développement interne.

## Installation manuelle de MySQL

Les options à modifier dans MySQL étant presque nulles, nous vous conseillons plutôt d'utiliser les paquets de votre éditeur ou d'utiliser les binaires disponibles sur le site <http://www.mysql.com/>.

Si toutefois vous souhaitez compiler le programme à partir des sources et si vous utilisez un `gcc` récent comme compilateur (ce qui est probablement le cas sur un système BSD ou Linux), il vous faudra utiliser l'option `-fno-exceptions` dans les drapeaux à la compilation :

```
export CXXFLAGS="$CXXFLAGS -fno-exceptions"
```

Il vous faudra aussi décider à quel utilisateur appartiendra le serveur MySQL. Ici, nous utiliserons `mysql`, que nous créons pour l'occasion.

```
groupadd mysql  
useradd -g mysql mysql
```

### Note

MySQL devrait avoir un utilisateur dédié. Il est risqué de partager ses droits d'accès avec un autre logiciel ou avec l'utilisateur qui exécute PHP.

Une fois les sources décompressées, vous pouvez exécuter l'étape de configuration avec la commande suivante :

```
./configure
```

Vous pouvez obtenir la liste des options possibles avec l'argument `--help`. Les deux options les plus importantes sont le répertoire destination (`--prefix`) et le jeu de caractères à utiliser si vous n'utilisez pas l'ISO-8859-1 (`--with-charset`).

```
./configure --prefix=/usr/local/mysql
```

Il reste alors à lancer la compilation et l'installation. Vous aurez besoin d'avoir les droits d'écriture sur le répertoire cible pour cette dernière étape.

```
make
make install
```

La configuration est contenue dans le fichier `support-files/my-medium.cnf`. Il vous faut la copier dans `/etc/my.cnf` et éventuellement la modifier. Si vous comptez utiliser les tables `innodb` (qui apportent les transactions et l'intégrité référentielle), vous devrez décommenter la ligne correspondante dans le fichier en enlevant le `#` de début de ligne.

```
cp support-files/my-medium.cnf /etc/my.cnf
```

S'il s'agit d'une nouvelle installation, il faut initialiser les bases de MySQL en utilisant le script `mysql_install_db` qui se trouve dans le sous-répertoire `bin` du répertoire cible.

```
cd /usr/local/mysql
bin/mysql_install_db
```

Enfin, il vous reste à donner les bons droits d'accès aux fichiers installés, afin que personne ne puisse modifier ou lire les bases sans passer par le serveur.

```
Cd /usr/local/mysql
chown -R root .
chown -R mysql var
chgrp -R mysql .
```

Vous pourrez alors démarrer le serveur avec la commande suivante :

```
/usr/local/mysql/bin/mysqld_safe --user=mysql &
```

### Important

N'oubliez pas de définir au plus vite un mot de passe pour le super-utilisateur de MySQL, afin que tout le monde ne puisse pas accéder par défaut à toutes vos bases.

## Installation manuelle de PHP

PHP est disponible dans la plupart des systèmes Unix en utilisant les paquets et installations de votre éditeur. Il peut être toutefois utile de compiler vous-même PHP pour ajouter des options peu fréquentes.

Les sources de PHP sont téléchargeables à l'adresse <http://www.php.net/downloads.php>.

### Schéma standard d'installation

Il existe plusieurs types d'installations pour PHP mais toutes partagent le même schéma commun.



## Décompactage des sources

En supposant que vous avez téléchargé les sources sous le format `tar.gz` dans le répertoire courant, vous pouvez les décompresser avec la commande suivante :

```
■ tar xzf php-5.1.4.tar.gz
```

## Préconfiguration

Les options de préconfiguration sont très nombreuses. Pour en obtenir la liste complète, utilisez la commande suivante :

```
■ ./configure --help
```

On peut toutefois individualiser quelques options qui doivent retenir votre attention.

L'option `--prefix` vous permet de spécifier un répertoire destination pour l'installation :

```
■ ./configure --prefix=/usr/local/php
```

D'autres options permettent l'utilisation des modules optionnels :

- `--with-apxs=xxx` : compile PHP en module Apache (recommandé), `xxx` est un sous-répertoire `apxs` dans votre arborescence Apache ;
- `--enable-pdo=shared` : si vous exécutez une version 5.1+ de PHP, le socle commun de PDO est inclus dans cette distribution ; il devrait être automatiquement activé lorsque vous lancerez le script de configuration de PHP. Il est recommandé de compiler PDO en tant qu'extension partagée, ce qui vous permettra de mettre à jour PDO via PECL ;
- `--with-zlib` : permet d'utiliser les fonctions de compression zip ;
- `--with-gd` : permet d'utiliser les fonctions de traitement d'image ;
- `--with-mysql=xxx` : permet d'utiliser les bases de données MySQL (`xxx` est le répertoire d'installation du client `mysql`, généralement `/usr` ou `/usr/local`, Attention : cet argument est obligatoire si vous activez aussi l'extension `mysql_i`) ;
- `--with-mysql_i=xxx` : permet d'installer les fonctions d'accès pour MySQL 4.1 et supérieur (`xxx` est le chemin d'accès vers le programme `mysql_config`, distribué avec MySQL à partir de la version 4.1) ;
- `--with-iconv` : bibliothèque permettant de convertir des textes d'un jeu de caractères à un autre ;
- `--with-imap` : pour pouvoir lire les courriers électroniques par les protocoles POP et IMAP ;
- `--enable-mbstring` : active la prise en charge interne des jeux de caractères internationaux, au prix d'une baisse des performances ;
- `--with-xslt` : pour utiliser la `libxslt` comme moteur de traitement XSLT (voir le chapitre sur XML) ;

- `--enable-ftp` : active la prise en charge des fichiers FTP ;
- `--with-openssl` : active `openssl` pour ouvrir des pages via le protocole HTTPS ;

### Bases de données avec PDO

Nous conseillons d'installer PDO en mode partagé, car la manipulation et les mises à jour se font plus simplement par la suite via PECL. Par exemple, pour ajouter le support de MySQL au socle commun de PDO, il suffira de faire :

```
pear install pdo_mysql
```

L'autre solution consiste à compiler PDO en mode statique en utilisant :

```
--with-pdo-mysql=/usr/local/mysql
```

### Compilation et installation

Une fois la préconfiguration faite, il vous faudra lancer la compilation et l'installation.

```
make  
make install
```

Il est probable qu'il vous faille vous authentifier en tant qu'utilisateur `root` pour accomplir cette dernière étape. Vous aurez en effet besoin des droits d'écriture sur le répertoire cible.

Quelle est la réponse à la question fondamentale de la vie, de l'univers et du reste ?

### Configuration

La dernière étape consiste à copier le fichier `php.ini` dans le sous-répertoire `lib` de votre répertoire cible. Un `php.ini` avec les valeurs recommandées par l'équipe PHP est disponible sous le nom `php.ini-recommended` avec les sources. Nous vous conseillons de l'utiliser.

```
cp php.ini-recommended /usr/local/lib/php.ini
```

### PHP en ligne de commande

Pour créer un PHP disponible en ligne de commande, il vous suffit de passer les options `--enable-cli` et `--disable-cgi` à l'étape de préconfiguration. Un binaire `php` dans le sous-répertoire `bin` sera alors créé pendant la phase d'installation.

### PHP en module Apache

Il existe deux modes de compilation avec Apache. Nous vous détaillons ici la plus souple : la compilation en module dynamique. Pour utiliser PHP en module Apache, vous devrez donc passer le paramètre `--with-apxs` à l'étape de préconfiguration. Si vous utilisez Apache 2 et non Apache 1.3, il faudra utiliser `--with-apxs2` à la place.

PHP modifiera lui-même votre configuration Apache pour demander le chargement automatique de son module. Il ne vous restera plus qu'à associer les fichiers `.php` au module

php pour qu'ils soient exécutés. Ajoutez donc cette ligne à votre fichier de configuration Apache :

```
AddType application/x-httpd-php .php
```

Pour que les modifications soient prises en compte, il vous faudra arrêter et relancer Apache.

**Note**

Un rechargement rapide d'Apache ne sera pas suffisant pour prendre en compte un module PHP qui vient d'être installé. Il faut arrêter complètement le serveur.

### PHP en CGI sur Apache

PHP construit par défaut un exécutable pour le mode CGI si vous ne lui demandez pas un type d'installation spécifique. La configuration Apache est alors similaire à celle pour Microsoft Windows :

```
ScriptAlias /php/ "/usr/local/php/bin"  
Action application/x-httpd-php "/php/php"  
AddType application/x-httpd-php .php
```

### Gestion des droits d'accès

L'avantage du mode CGI, malgré les pertes de performance, est qu'il permet de gérer finement les droits d'accès. En effet, par défaut, tous les scripts s'exécutent avec les mêmes droits d'accès : ceux du serveur web.

Ce comportement peut être gênant si vous avez plusieurs utilisateurs ou sites distincts sur un même serveur. Apache a une fonctionnalité spécifique pour gérer cette situation avec les CGI : `suexec`. En l'activant, PHP exécute chaque script PHP avec les droits d'accès de son propriétaire.

Le module `suexec` touche de près à la sécurité de la plate-forme ; nous préférons vous laisser consulter un ouvrage dédié à Apache et à son administration pour les détails et les conséquences de cette installation. Sachez juste que si vos utilisateurs n'ont pas confiance les uns en les autres et ne veulent pas être bridés en fonctionnalités, c'est probablement la réponse la plus adaptée et il peut être pertinent de s'y arrêter.

### Modules additionnels PECL

Les dépôts PEAR et PECL proposent des modules additionnels pour PHP. On y trouve par exemple des drivers PDO spécifiques ou une classe pour la gestion détaillée du protocole HTTP. Ces modules s'installent en ligne de commande avec l'outil `pear`. Ils sont alors téléchargés, compilés et installés sur la machine. Il ne reste plus qu'à les activer dans le fichier `php.ini`.

```
pear install pdo_mysql
```

## Configuration de PHP avec php.ini

Installer PHP n'est pas bien compliqué, mais la majorité des gens utilisent l'installation par défaut sans savoir que le fichier de configuration `php.ini` est une véritable mine d'or. Ce fichier décrit la configuration initiale de PHP et ses comportements par défaut. Tout programmeur qui se veut expert PHP se doit d'avoir parcouru de nombreuses fois ce fichier. Nous allons ici le parcourir et présenter quelques-unes des directives de configuration les plus intéressantes.

### Note

Ces directives influent sur le comportement du langage. Il est donc possible qu'il vous manque certaines bases pour en comprendre les buts. N'hésitez pas à passer rapidement et à revenir sur cette partie par la suite.

## Utilisation des modules et des extensions

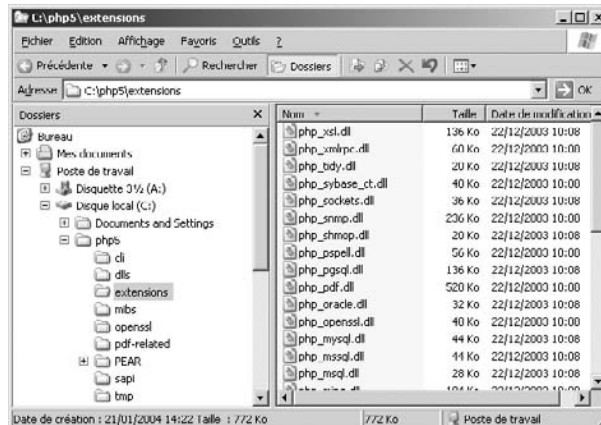
PHP gère une grosse partie de ses fonctionnalités avancées via des modules additionnels. Il est possible d'ajouter ou de retirer des fonctions comme le chiffrement, la connectivité avec Oracle ou MySQL, la manipulation des images, etc. Chacun de ces modules peut être compilé dans PHP de manière statique, ou externalisé pour pouvoir être activé ou désactivé dans la configuration.

Pour utiliser un module de manière statique, il faut passer des options à la compilation de PHP pour qu'il soit intégré à ce moment-là. Sur les plates-formes Unix, vous le ferez via des paramètres tels que `--with-nom_module`. Vous trouverez plus de renseignements dans les paragraphes détaillant l'installation manuelle sous Unix.

### Répertoires contenant les extensions

Sous Microsoft Windows et dans la plupart des distributions Unix binaires, les modules sont gérés dynamiquement. Ils sont alors représentés par des bibliothèques dans le répertoire d'extensions de PHP (des fichiers `.so` sous Linux et `.dll` sous Windows, voir figure 2-5).

Figure 2-5  
Extensions PHP



L'adresse de ce répertoire est donnée par la directive `extension_dir` présente dans le fichier de configuration `php.ini`.

```
extension_dir = "C:/php5/extensions"
```

### Activation d'un module dynamique

Après vous être assuré que le fichier que vous voulez est présent dans le répertoire d'extensions (par exemple `php_gd2.dll` pour la bibliothèque manipulant les images sous Windows), vous pouvez vous rendre dans le fichier de configuration `php.ini`. Vous trouverez des lignes qui ressemblent aux suivantes :

```
;Windows Extensions
;
;extension=php_filepro.dll
extension=php_gd2.dll
;extension=php_gettext.dll
extension=php_iconv.dll
extension=php_imap.dll
;extension=php_interbase.dll
;extension=php_mcrypt.dll
;extension=php_ming.dll
extension=php_mysql.dll
extension=php_mysqli.dll
;extension=php_oracle.dll
;extension=php_pdf.dll
;extension=php_pgsql.dll
;extension=php_snmp.dll
```

Si la ligne est décommentée (pas de point-virgule en premier caractère), PHP chargera l'extension au démarrage et vous pourrez en utiliser les fonctionnalités. Vous pouvez ajouter des lignes si vous avez ajouté des extensions qui ne sont pas dans le fichier par défaut. Pour décharger une extension que vous n'utilisez plus, il suffit de commenter la ligne correspondante. Dans tous les cas, si vous utilisez PHP en tant que module Apache, il faudra redémarrer le serveur web pour que les modifications prennent effet.

Au cours des prochains chapitres, nous vous indiquerons quels modules sont nécessaires et comment les activer.

## Les directives de configuration

### Balises d'ouverture

Comme vous le verrez au chapitre 3 sur les bases du langage, il est possible d'utiliser plusieurs types de balises d'ouverture PHP. Celle qui est recommandée est `<?php` mais il est également possible d'utiliser `<?` si la directive `short_open_tag` est à `0n`. Si vous voulez un code portable, ne les utilisez pas.

```
short_open_tag = Off
```

Certaines personnes habituées à ASP préfèrent utiliser les balises <% de cette plate-forme. Vous pouvez autoriser cette syntaxe avec la directive `asp_tag`.

```
asp_tags = On
```

### Compression des pages

Il est possible, et nous le verrons en détail au chapitre 15 sur les flux de sortie, de dire à PHP de compresser automatiquement les données de sortie au format Zlib. Cela vous permet d'économiser de la bande passante.

```
zlib.output_compression = On  
zlib.output_compression_level = 9
```

### Mode sécurisé (*safe mode*)

PHP propose une option permettant de sécuriser le système et de limiter les possibilités d'interaction entre PHP et le système. Bien qu'étant un peu bloquante pour le développeur, elle permet à l'administrateur d'être sûr que le développeur n'outrépassera pas ses droits. La directive s'appelle `safe_mode`. Si elle est activée, PHP n'autorisera pas les exécutions de programme externe ou l'accès aux fichiers d'autres utilisateurs. Consultez le chapitre 27 sur la sécurité pour en savoir plus.

```
safe_mode = Off
```

### Limitation d'accès aux répertoires

La directive `open_basedir` permet de limiter toutes les manipulations de fichiers à une arborescence définie. Si cette directive est définie, vous n'aurez le droit d'utiliser que les fichiers qui commencent par ce préfixe. Tout autre accès engendrera une erreur.

```
open_basedir = "/home/www";
```

Si le même serveur web sert pour plusieurs utilisateurs, en définissant à chacun un `open_basedir` différent, ils ne pourront pas voir ou modifier les fichiers des autres. L'option est aussi utilisée pour réduire les conséquences d'une faille de sécurité : celui qui aura accès à PHP ne pourra modifier que les fichiers de cette architecture et ne pourra pas aller modifier ou lire des fichiers externes. Vous trouverez plus de détails sur cette option dans le chapitre sur la sécurité.

### Gestion des ressources

Par défaut, un script PHP s'exécute au maximum pendant trente secondes. Passé ce délai, le script s'arrête et affiche le message d'erreur suivant :

```
Fatal error: Maximum execution time of 30 seconds exceeded in d:\www\max_temps.php  
↳ on line XX
```

Il est possible de définir le temps d'exécution maximal via `max_execution_time`. Si vous définissez 0, vos scripts pourront s'exécuter indéfiniment.

```
max_execution_time = 100
```

**Note**

La directive `max_execution_time` est un bon garde-fou contre les boucles infinies. Il est déconseillé de mettre sa valeur par défaut à 0.

Chacun de vos scripts PHP occupe de la mémoire pendant son exécution. Le maximum alloué par défaut est de 8 Mo. Il peut parfois être nécessaire de disposer de plus de mémoire. Pour cela, il faut modifier la directive `memory_limit` :

```
memory_limit = 8M
```

**Gestion des erreurs**

Il existe plusieurs sortes d'erreurs et PHP vous permet de choisir lesquelles il doit afficher. Pour cela, il faut utiliser `error_reporting`. Pour avoir toutes les informations à ce sujet, rendez-vous au chapitre 19 concernant la gestion des erreurs.

```
error_reporting = E_ALL
```

Il est possible d'activer ou non l'affichage des messages d'erreurs sur les pages HTML via `display_error`. Dans le cas de sites en production, il est conseillé de mettre sa valeur à `0ff` pour ne pas troubler l'utilisateur avec des messages obscurs, et pour des raisons de sécurité.

```
display_errors = 0ff
```

En revanche, si vous n'activez pas l'affichage, vous ne pourrez pas avoir de retour sur les erreurs. Afin de remédier à ce problème, il est recommandé d'activer la journalisation des erreurs via le système d'erreurs de votre serveur web.

```
log_errors = 0n
```

**Gestion des données**

Dans cette partie se trouve le principal changement de PHP au cours de sa version 4. Il s'agit du paramètre `register_globals`, qui est depuis la 4.1 à `0ff` par défaut. Pour des raisons de sécurité, nous vous déconseillons très fortement d'activer cette option. Elle reste toutefois disponible pour compatibilité avec les anciens scripts PHP 4.0. Le détail des conséquences sera expliqué au chapitre 8.

```
register_globals = 0ff
```

**Gestion des magic quotes**

Les anciennes versions de PHP embarquaient par défaut un mécanisme destiné à protéger l'utilisateur des injections SQL. PHP filtrait alors toutes les entrées utilisateur (paramètres HTTP reçus en GET, en POST ou via les cookies) et ajoutait une barre oblique inverse (caractère `\`) devant les apostrophes, les guillemets et le caractère nul.

Ce comportement est maintenant déconseillé, car il rend complexe la gestion des données (il faut les gérer différemment selon qu'elles viennent de l'utilisateur ou d'autres sources) et entraîne une baisse légère de performances pour ceux qui n'en ont pas besoin. Dans la suite de ce livre, nous considérerons par défaut que les `magic_quotes` sont désactivés.

Vous pouvez toutefois le réactiver pour compatibilité avec vos anciens scripts en mettant à 0n la directive `magic_quotes_gpc`.

```
magic_quotes_gpc = Off
```

### Gestion des chemins

Par défaut, quand vous essayez d'inclure un fichier PHP dans un autre avec `include()` ou `require()`, PHP les cherche dans le répertoire en cours lors du lancement du script (dans le cadre web, il s'agit normalement du répertoire contenant le script). Il est toutefois possible de définir une liste de répertoires où PHP devra chercher ses fichiers. Les chemins à utiliser doivent être précisés dans la directive `include_path`. Le séparateur est le caractère : sous les systèmes Unix et le caractère ; sous les systèmes Microsoft Windows.

```
include_path = ".:php/includes"
```

L'ordre des chemins est important, PHP s'arrêtera de chercher dès qu'il trouvera un fichier avec un nom correspondant. Si plusieurs chemins spécifiés contiennent un fichier du même nom, c'est le premier qui sera utilisé.

#### Attention

N'oubliez pas d'inclure le répertoire courant (.) dans la liste. Si ce n'était pas le cas, PHP ne chercherait pas dans le répertoire en cours pour trouver un fichier, même s'il existe.

### Gestion des fichiers téléchargés

Vous pouvez autoriser ou non le téléchargement HTTP de fichiers en attribuant la valeur 0n ou 0ff à `file_upload`.

```
file_uploads = On
```

Les fichiers seront automatiquement sauvegardés dans le répertoire temporaire de votre système, sauf si vous spécifiez une adresse différente via `upload_tmp_dir`. Enfin, vous pouvez limiter la taille des fichiers téléchargés :

```
upload_max_filesize = 2M
```

### Gestion des sessions

Comme nous le verrons en détail au chapitre 11 traitant des sessions, il est possible de stocker celles-ci autrement que dans des fichiers (mode par défaut). C'est avec la directive `session.save_handler` que vous le définissez :

```
session.save_handler = files
```



Dans le cas le plus courant (si vous utilisez des fichiers) il est possible de définir le répertoire où seront sauvegardés vos fichiers de sessions. Nous vous conseillons fortement de définir ici un répertoire qui vous est propre et où d'autres utilisateurs du serveur n'ont pas accès en lecture. L'utilisation d'un répertoire en accès public risquerait de poser des problèmes de sécurité liés à la divulgation des identifiants de session.

```
session.save_path = "//data/users/session/"
```

### Inclusion de fichier automatique

PHP vous permet d'inclure automatiquement un fichier PHP avant (ou après) le fichier à exécuter. Vous pouvez ainsi faire charger un fichier avec des variables de configuration qui seront utilisées dans vos scripts, ou ajouter à la fin un fichier qui fait des opérations statistiques.

Si la directive de configuration `auto_prepend_file` contient un chemin de fichier, ce fichier sera inclus automatiquement comme si le fichier appelé avait un `include()` en première ligne.

```
auto_prepend_file = /home/www/configuration.php
```

La directive `auto_append_file` fonctionne de manière similaire mais ajoute le fichier à la fin de l'exécution.

#### Note

Le fichier ne sera ajouté à la fin de l'exécution que si elle se termine normalement. Une erreur fatale, un appel à `exit()` ou `die()` ne permettra pas cette inclusion.

## Gestion de la configuration

Jusqu'ici, nous avons abordé les différentes modifications de configuration qu'il était possible de faire dans le `php.ini`. Il est aussi possible de définir certaines directives de configuration soit au niveau d'Apache pour un répertoire ou un hôte virtuel, soit directement pendant l'exécution d'un script.

### Modification dans un script

Il est possible de modifier la configuration de votre PHP directement dans vos scripts. Pour cela, il vous faut utiliser la fonction `ini_set()` :

```
ini_set(directive, valeur)
```

`ini_set()` change la valeur de l'option de configuration `directive` et lui donne la valeur de `valeur`. `ini_set()`. Elle renvoie la valeur précédente en cas de succès et `FALSE` en cas d'échec. La valeur de l'option de configuration sera modifiée durant toute l'exécution du script et pour ce script spécifiquement. Elle reprendra sa valeur par défaut dès la fin du script.

```
<?php
ini_set('session.save_handler','mm');

// Code
?>
```

**Remarque**

Toutes les options disponibles ne peuvent pas être modifiées avec `ini_set()`. Généralement, toutes les options qui changent le comportement de PHP à l'initialisation ou qui ont trait à la sécurité de PHP ne peuvent pas être modifiées pendant l'exécution. Référez-vous à la documentation pour plus d'informations.

**Connaître une valeur de configuration**

La fonction `ini_get()` permet de récupérer la valeur d'une directive de configuration (modifiable ou pas) telle qu'elle est définie au moment où vous la demandez.

```
<?php
echo ini_get('session.save_handler');
// Affiche files

ini_set('session.save_handler','mm');
echo ini_get('session.save_handler');
// Affiche mm
?>
```

**Restaurer une directive de configuration**

Enfin, après une modification, il est possible de revenir à la valeur par défaut via la fonction `ini_restore()`. Celle-ci prend en paramètre le nom de la directive à restaurer. L'exemple suivant est illustré à la figure 2-6.

```
<?php
echo 'La valeur de session.gc_maxlifetime est : ' ,
    ini_get('session.gc_maxlifetime').'<br>';

ini_set('session.gc_maxlifetime',43200);
echo 'La valeur de session.gc_maxlifetime est : ' ,
    ini_get('session.gc_maxlifetime').'<br>';

ini_restore('session.gc_maxlifetime');
echo 'La valeur de session.gc_maxlifetime est : ' ,
    ini_get('session.gc_maxlifetime').'<br>';
?>
```

**Figure 2-6**  
*Modification de configuration dans un script*



## Modification de configuration via Apache

Si vous utilisez le module PHP pour Apache, vous pouvez modifier la configuration de PHP directement à partir de la configuration d'Apache. Cela vous permet notamment de définir une configuration spécifique pour un répertoire ou pour chaque hôte virtuel. On peut ainsi donner une configuration spécifique à une application ou à un site sans conséquence sur les autres scripts.

Vous avez à votre disposition deux directives : `php_flag` et `php_value`. La directive `php_flag` permet de modifier une directive de configuration qui a une valeur binaire. On passe alors en paramètres le nom de la directive PHP à modifier et la nouvelle valeur. La directive `php_value` fonctionne de manière similaire mais permet de donner une valeur textuelle à une directive PHP.

```
<VirtualHost www.le-comedien.com>
  ServerAdmin boss@le-comedien.com
  DocumentRoot /var/www/le-comedien
  php_flag magic_quotes_gpc off
  php_value include_path "./data/le-comedien"
</VirtualHost>
```

Pour les directives PHP qui ont trait à la sécurité (comme l'activation de `safe_mode`, `open_basedir`, etc.), il vous faudra utiliser les directives équivalentes `php_admin_flag` et `php_admin_value`.

```
<VirtualHost www.le-comedien.com>
  ServerAdmin boss@le-comedien.com
  DocumentRoot /var/www/le-comedien
  php_admin_flag safe_mode off
</VirtualHost>
```

Les directives `php_admin_flag` et `php_admin_value` ne peuvent pas être spécifiées dans les surcharges de configuration Apache (les fichiers `.htaccess`).

## Fichier `php.ini` local

Si vous utilisez PHP en CGI ou en ligne de commande, le moteur cherchera d'abord un `php.ini` dans le répertoire courant avant de chercher le fichier global. Vous pouvez mettre ce mécanisme à profit pour créer un fichier `php.ini` local à un répertoire juste pour quelques scripts.

Ce fichier local peut être un fichier de configuration complet ou contenir juste quelques directives. Il sera alors fusionné avec le fichier global pour définir les directives non existantes.

# 3

## Les structures de base

---

Ce chapitre explique brièvement les notions et les structures de base du langage PHP. L'objectif principal est de permettre au plus néophyte une prise en main et une utilisation immédiates. Le lecteur plus expérimenté y trouvera quelques remarques sur les subtilités de PHP, fruits de l'expérience des auteurs, ainsi que certaines notions peu connues telle que la notation *heredoc* permettant d'insérer une large quantité de texte dans une variable.

### Insertion de PHP dans HTML

Le code PHP peut être directement intégré dans les fichiers HTML. Il peut figurer à différents endroits de ces fichiers, tout en étant entrecoupé de code HTML.

```
<html>
<head><title>Test PHP </title></head>
<body>
<h1>Texte mis en avant</h1>
<?php
echo "<p>ceci est du code PHP</p>";
echo "<p>simple non ? </p>";
?>
</body>
</html>
```

#### Remarque

Nous parlons ici du HTML car c'est l'utilisation la plus courante, mais PHP s'intègre de la même manière dans n'importe quel format texte à partir du moment où le serveur web est configuré pour cela.

## Balises d'ouverture et de fermeture

Le début et la fin des portions de code PHP sont signalés grâce à des balises d'ouverture et de fermeture. Seul ce qui est entre ces balises est interprété par PHP, le reste est envoyé tel quel.

Tableau 3-1 Les différentes balises d'ouverture et de fermeture PHP

Ouverture	Fermeture
<?php	?>
<?	?>
<%	%>
<script language= "php">	</script>

Nous vous conseillons fortement l'utilisation unique de l'ouverture <?php, car elle est la seule vraiment portable sur toutes les configurations.

### Remarque

Si vous utilisez la balise d'ouverture « <? » , vous risquez d'avoir un problème avec les fichiers XHTML qui commencent par la ligne « <?xml version="1.0" encoding="UTF-8"?> ».

Les autres formulations sont activables ou désactivables via des options dans le fichier de configuration `php.ini` (directive `short_tags` pour <? et directive `asp_tags` pour <%).

Il existe une syntaxe supplémentaire : <?=valeur ?>. Elle est équivalente à <?php echo valeur ?>.

```
<?='bonjour'?>
<?=$variable?>
```

### Configuration

Pour utiliser cette dernière syntaxe, il faut que la directive `short_tags` soit activée dans votre configuration. Par défaut, cette directive est désactivée.

## Les commentaires

Comme avec le langage C, les commentaires sont introduits par la séquence `/*` et se terminent par `*/`. Par ailleurs, PHP utilise également les signes de commentaires `//`, qui permettent de commenter une ligne complète.

```
<?php
/*
Commentaires sur plusieurs lignes
```

```
exemple
*/
// Commentaires sur une ligne
echo "ceci n'est pas commenté" ;
?>
```

Un commentaire n'aura aucune influence sur l'exécution et sera ignoré par le moteur. Vous pouvez donc profiter des commentaires pour ajouter des explications sur les parties de code complexes afin d'en faciliter la relecture et la compréhension par la suite.

**Note**

Il est possible d'utiliser la norme PHPDoc pour commenter vos fichiers. Celle-ci vous permet de générer automatiquement une documentation technique. Plus d'informations : <http://www.phpdoc.org>.

## Enchaînement des instructions

Les instructions PHP doivent être placées entre les balises d'ouverture et de fermeture de PHP (<?php et ?>) et être séparées par des points-virgules.

```
<?php
$a = 5 ;
$b = 3 ;
$c = "PHP5" ;
echo $c ;
?>
```

Seuls ces points-virgules séparent les différentes instructions ; les retours à la ligne n'ont aucune influence. Il est donc possible d'imbriquer plusieurs instructions sur la même ligne ou de faire une unique instruction sur plusieurs lignes :

```
<?php
$a = 5 ;
$b = 3 ;
$c = "PHP5" ;
echo $c ;

/* Équivalent à */

$a = 5 ; $b = 3 ;
$c
=
    "PHP5" ; echo $c ;
?>
```

Cette dernière notation, bien que possible, est fortement déconseillée car elle entraîne d'importantes difficultés de relecture et favorise l'apparition d'erreurs.

## Erreur courante

Si vous oubliez le point-virgule, vous verrez apparaître un `Parse error` lors de l'exécution de votre fichier (voir figure 3-1). Cela signifie que PHP, en lisant ligne à ligne le fichier de script, est tombé sur une incohérence de syntaxe (un point-virgule oublié, des guillemets en trop, etc.).

Figure 3-1

*L'erreur classique, le Parse error*



Il s'agit de l'erreur la plus courante : « `Parse error line 56` ». Le réflexe à avoir est le suivant : ouvrez le fichier concerné et rendez-vous à la ligne indiquée (56 dans l'exemple). Regardez la ligne précédente et cherchez d'abord la présence du point-virgule symbolisant la fin de l'instruction.

## Structure du document

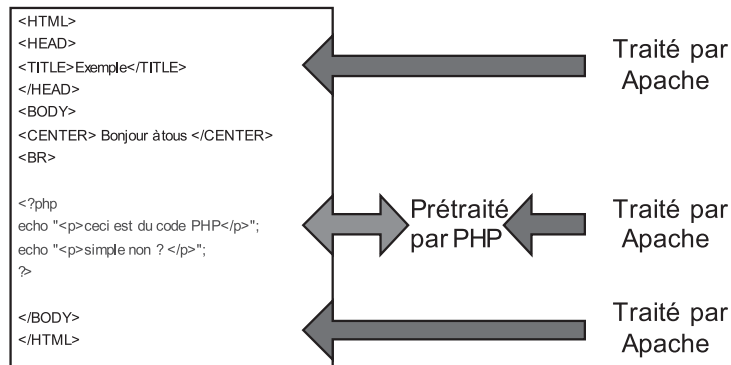
Pour indiquer au serveur qu'un ensemble de lignes sera du PHP, on écrit `<?php`. Pour indiquer au serveur que l'interprétation n'est plus nécessaire, on referme avec la balise `?>`. Ainsi, PHP traitera ce qui est entre ces balises et renverra le reste tel quel au serveur web (donc au navigateur). Des schémas explicatifs sont présentés aux figures 3-2 et 3-3.

### Note

La balise de fermeture `?>` est facultative à la fin du fichier.

Figure 3-2

*Interprétation d'un fichier PHP*



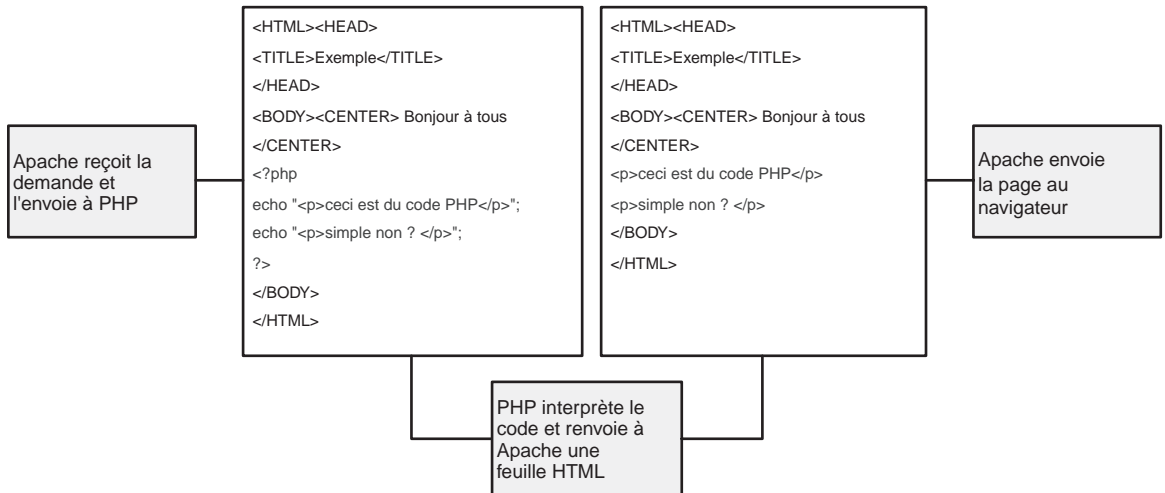


Figure 3-3

Interprétation d'un fichier PHP

### Affichage et envoi de code HTML

La commande `echo` indique au serveur qu'il doit renvoyer les informations contenues entre les guillemets. On notera qu'il est possible de renvoyer du code HTML (balises `<p>` et `</p>`). On peut tout aussi bien renvoyer le code HTML désignant une image :

```

<?php
echo "<p>Ceci est du code PHP</p> ";
echo "<img src='./image.gif' alt=''>";
?>
  
```

### Exécuter du code PHP

Il existe plusieurs façons d'utiliser PHP. La plus courante est l'utilisation de PHP pour une application web. Il est toutefois aussi possible d'exécuter PHP via d'autres architectures.

#### Via un serveur web

Quand vous faites appel à PHP pour gérer une application web, on parle de mode client/serveur. Les scripts PHP sont stockés sur une machine dédiée (le serveur) et leur interprétation s'effectue sur cette même machine. L'appel des scripts et leur affichage se font via un navigateur, sur une autre machine, qui joue le rôle du client.

Cette architecture est la plus répandue pour PHP. Si votre serveur a bien été configuré et installé comme décrit au chapitre 2, il vous suffit de lancer votre navigateur et de taper l'adresse du serveur pour exécuter les scripts souhaités. Les scripts appelés devront tous avoir une extension reconnue par le serveur web comme appartenant à PHP (généra-



lement `.php`, voir au chapitre 2 pour plus d'informations sur la configuration des extensions).

**Note**

Sur les plates-formes de développement, il est fréquent de faire tourner un serveur web sur sa propre machine. Dans ce cas, le serveur et le client (navigateur) sont représentés par la même machine physique. Une erreur fréquente consiste à appeler le fichier de script avec son chemin dans l'arborescence du disque dur (`c:\www\test.php` ou `/data/www/test.php`). Dans ce cas, le navigateur ne passera pas par le serveur web mais lira directement le fichier de script sur le disque dur. Les instructions PHP ne seront donc pas exécutées.

**En ligne de commande**

Pour exécuter un programme PHP en ligne de commande, il vous suffit d'appeler l'exécutable `php` avec deux paramètres : `-q` (qui demande de ne pas afficher les en-têtes HTTP) et le chemin du fichier `php`.

Sous Linux :

```
■ /usr/local/bin/php -q monfichier.php
```

Sous Microsoft Windows :

```
■ c:\php\php.exe -q monfichier.php
```

**Exécution en programme autonome**

Un script PHP ne peut pas être exécuté de façon totalement autonome ; il aura toujours besoin de l'exécutable `php` et de ses bibliothèques. Il est toutefois possible de faire fonctionner un script PHP comme un programme classique tel qu'un script *batch* ou un programme de traitement de texte.

Sous Unix, il faut donner les droits d'exécution au script puis ajouter la déclaration suivante en première ligne (vous aurez peut-être à changer le chemin d'accès à l'exécutable `php`) :

```
■ #!/usr/local/bin/php -q
```

**Note**

On donne généralement les droits d'exécution avec la commande `chmod +x chemin/du/fichier`.

Sous Microsoft Windows, vous pouvez aller dans la configuration des types de fichiers (on obtient la boîte en choisissant les menus Outils > Options des dossiers dans l'explorateur de fichiers). Là, il vous faudra associer l'extension `.php` au programme `php`. Une fois cette manipulation faite, double-cliquez sur un fichier PHP pour le lancer dans une fenêtre DOS.

## Mode embarqué

Enfin, il existe un dernier mode d'exécution pour PHP : le mode embarqué. Il est utilisé par quelques programmes pour permettre l'utilisation de PHP comme langage de macro ou langage de script interne.

Parmi ces utilisations, on peut retrouver MySQL qui a prévu de permettre l'utilisation de procédures stockées en PHP pour ses futures versions.

## Constantes et variables

### Variables

Les variables sont l'ossature de la programmation. Sans elles, les possibilités seraient extrêmement limitées. Pour simplifier, une variable peut être représentée comme un récipient disponible pendant toute l'exécution de votre programme. Ainsi, au cours du script, vous pouvez lui donner des valeurs, les modifier et les utiliser. En PHP, l'utilisation des variables est très simple et ne nécessite aucune déclaration préalable.

### Syntaxe des variables

Les variables en PHP se trouvent sous la forme `$nom_variable`. Elles commencent par le symbole `$` et sont formées d'une suite de lettres, de chiffres et de caractères de soulignements. Le premier caractère du nom d'une variable ne peut pas être un chiffre.

```
<?php
$variable = "ma première variable";
?>
```

#### Attention

Il est important de noter que les noms de variables sont sensibles à la casse. Une majuscule n'est pas équivalente à une minuscule. `$PHP`, `$Php` et `$php` sont trois variables différentes.

Que ce soit pour les noms des variables, pour les noms des fichiers ou tout ce qui concerne le choix des noms des entités du projet, nous vous conseillons de définir une norme commune dès le début du développement. Vous pouvez par exemple décider que les variables, les noms de fichiers et les noms des tables dans la base de données seront tous écrits en minuscules.

**Tableau 3-2 Les noms de variable**

Noms de variables		
Correct	Incorrect	Explication
<code>\$Variable</code>	<code>\$Variable 1</code>	Contient des espaces
<code>\$variable</code>	<code>Variable</code>	Une variable commence toujours par <code>\$</code>
<code>\$variable_double</code>	<code>\$variable-double</code>	Le signe <code>-</code> est interdit
<code>\$variable_email</code>	<code>\$test@yahoo.fr</code>	Les caractères <code>@</code> et <code>.</code> sont interdits.
<code>\$test2</code>	<code>\$2test</code>	Une variable ne commence pas par un chiffre.

## Déclaration et types

Contrairement à d'autres langages de programmation plus susceptibles, PHP n'impose pas de déclarer les variables avant de les utiliser. Cette particularité rend la programmation en PHP plus aisée.

Exemple de programme en C :

```
#include <stdio.h>
#include <string.h>
char   ligne[100];
int    hauteur ;

main()
{
    hauteur = 10 ;
    (void)strcpy(ligne,"droite et courte");
    ...
}
```

Exemple de programme en PHP :

```
<?php
$ligne = "droite et courte";
$hauteur = 10;
// ...
?>
```

En général, une variable ne peut contenir qu'un type de données et ce type doit être déclaré avant d'utiliser la variable (comme dans l'exemple ci-dessus sur C). En PHP, le type d'une variable est déterminé par la valeur qu'on lui donne. Ce type peut changer au cours du programme suivant les affectations. PHP est un langage dit « de typage faible et dynamique ».

L'utilisation en lecture d'une variable non initialisée n'est pas non plus un problème. Une variable inexistante renverra la valeur NULL et, selon votre configuration, pourra ne pas afficher d'erreur. Il vous faut donc faire attention à l'orthographe de vos variables.

### Note sur la configuration

Pendant les développements, nous vous conseillons de mettre le niveau de rapport d'erreur (`error_reporting` dans le `php.ini`) à `E_ALL` afin que PHP vous prévienne lors de l'utilisation d'une variable non initialisée. Il s'agit le plus souvent d'erreurs de programmation qu'il est bon de repérer.

## Portée des variables

Il est important de noter que les variables ont une existence temporaire : elles n'existent que tant qu'elles sont utilisées dans un script. Une fois la page affichée, ces variables cessent d'exister. Il est donc impossible de stocker une valeur dans une variable pour la relire dans un autre script ou dans le même script mais lors d'une autre exécution.

## Variables locales

Les variables de PHP sont par défaut ce qu'on appelle des variables locales. Elles ne sont visibles et lisibles que dans le contexte où elles ont été créées. Ainsi, si je définis une variable `$var` dans la fonction `fct()`, je ne pourrai pas la relire en dehors de cette fonction. Une fois l'exécution de ma fonction terminée, toutes les variables qui y ont été définies sont perdues ; elles sont locales à cette fonction. Si, dans une fonction, vous avez besoin d'informations venant d'un autre contexte, il faudra les faire passer dans les paramètres lors de l'appel à la fonction.

De la même façon, les variables qui sont utilisées en dehors de toute fonction (on parle de variables globales) ne peuvent pas par défaut être utilisées dans les fonctions. Si j'écris une variable `$var` dans le contexte général et que j'écrive aussi une variable `$var` dans ma fonction, les deux représenteront des espaces différents, bien qu'ayant le même nom.

## Utilisation d'une variable globale

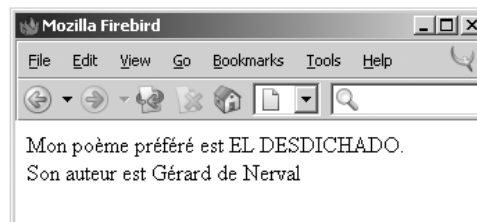
Parfois, l'utilisation de variables globales dans les fonctions peut être utile. On peut alors utiliser le tableau `$GLOBALS[]`. Il s'agit d'un tableau associatif contenant des références sur toutes les variables globales actuellement définies. Ainsi, `$GLOBALS['var']` représente la variable globale `$var`.

```
<?php
function gargarise($var1,$var2){
    echo $var1, $GLOBALS['titre'], $var2, $GLOBALS['auteur'];
}
$titre = 'EL DESDICHADO';
$auteur = 'Gérard de Nerval';
gargarise('Mon poème préféré est ', '<br> Son auteur est ');
// Affiche Mon poème préféré est EL DESDICHADO.
// Son auteur est Gérard de Nerval
?>
```

Dans notre exemple `$var1` et `$var2` sont des variables locales à notre fonction `gargarise()`. On fait appel aux variables globales `$titre` et `$auteur` via la superglobale `$GLOBALS[]`. Le résultat est donné dans la figure 3-4.

Figure 3-4

*Utilisation des variables globales*



Une alternative à l'utilisation du tableau `$GLOBALS` est de déclarer une variable comme étant une variable globale en début de fonction grâce au mot-clé `global`. Il doit être suivi d'un ou plusieurs noms de variables séparés par des virgules.

```
<?php
function gargarise($var1,$var2){
    global $titre, $auteur ;
    echo $var1, $titre, $var2, $auteur;
}
$titre = 'EL DESDICHADO';
$auteur = 'Gérard de Nerval';
gargarise('Mon poème préféré est ', '.Son auteur est ');
// Affiche Mon poème préféré est EL DESDICHADO.
// Son auteur est Gérard de Nerval
?>
```

**Note**

L'instruction `global` ne vaut que pour la fonction où elle apparaît. Pour les autres fonctions, des variables locales continueront à être utilisées. Si vous voulez utiliser des variables globales dans toutes les fonctions, il vous faudra ajouter cette déclaration à chaque fois.

**Test d'existence**

La fonction `isset()` permet de tester si une variable existe.

```
isset( $var )
```

Dans le cas où l'on a précédemment affecté une valeur à `$var`, la fonction renverra la valeur `TRUE`. Dans le cas contraire, la fonction renverra la valeur `FALSE`. Nous reviendrons en détail sur cette fonction dans le chapitre sur la gestion des formulaires, où elle prend toute son utilité.

```
<?php
$s = "test";
echo isset($s); // Renvoie TRUE
echo isset($j); // Renvoie FALSE
?>
```

**Attention**

Si une variable contient une chaîne de caractères vide, elle sera considérée comme ayant un contenu. La fonction `isset()` renverra donc la valeur `TRUE`, même si la chaîne elle-même ne contient rien. Utilisez plutôt la fonction `empty()` si vous souhaitez tester son contenu.

**Destruction**

La fonction `unset()` permet de détruire une variable dans votre programme. Après son exécution, la variable qui lui est passée en paramètre n'existe plus.

```
unset( $var )
```

Nous allons réutiliser l'exemple précédent et voir qu'avant destruction de la variable la fonction `isset()` renvoie `TRUE` et qu'après, elle renvoie `FALSE`.

```
<?php
$s = "test";
echo isset($s); // Renvoie TRUE
unset($s);
echo isset($s); // Renvoie FALSE
?>
```

## Variables dynamiques

Les variables dynamiques (aussi dites variables variables) reposent sur le fait que le nom d'une variable peut lui-même être une variable. Voyons l'exemple ci-après pour appréhender la chose :

```
<?php
$cd = "15 _";
$dvd = "30 _";

$produit = "dvd"; // On choisit comme produit le dvd
echo $$produit; // Affiche 30 _ soit le prix du dvd
?>
```

Il est aussi possible de référencer un nom dynamique en une seule opération grâce à des accolades :

```
<?php
$cd = "15 _";
$dvd = "30 _";

echo ${"dvd"}; // Affiche 30 _

$produit = "dvd";
echo ${$produit}; // Équivaut à echo $dvd et affiche donc 30 _
?>
```

Il est possible de faire des opérations à l'intérieur des accolades, par exemple des concaténations.

```
<?php
$cd = "15 _";
$dvd = "30 _";

echo ${"d" . "vd"}; // Affiche 30 _
?>
```

## Constantes

Le langage PHP définit les constantes à l'aide de la fonction `define()`. Elles ne peuvent plus par la suite recevoir d'autres valeurs. Par convention, on écrit les constantes en majuscules pour faciliter la relecture du code.

```
<?php
define("NOM", "Anaska");
```

```
echo NOM;  
?>
```

**Attention**

Les constantes ne comportent pas de \$ devant leur nom.

Il est fréquent de construire des fichiers qui ne contiennent que des constantes, pour gérer des paramètres de configuration ou des traductions de manière centralisée.

Le portail Xoops (<http://xoops.org>) fait usage d'un tel fichier de constantes pour gérer ses traductions :

```
<?php  
// $Id: install.php,v 1.7 2003/02/12 11:35:36 okazu Exp $  
// Support Francophone de Xoops (www.frxoops.org)  
define("_INSTALL_L0","Bienvenue dans l'assistant XOOPS 2.0");  
define("_INSTALL_L2","Maintenant, changez cette ligne en :");  
define("_INSTALL_L3","Ensuite, à la ligne 35, changez %s en %s");  
define("_INSTALL_L5","ATTENTION !");  
define("_INSTALL_L7","Vos paramètres :&nbsp;");  
define("_INSTALL_L8","Nous avons détecté :&nbsp;");  
...  
?>
```

Un fichier de ce type est géré par langues. Xoops charge alors au démarrage le fichier adéquat, selon la langue à utiliser. Ainsi, dans le code source de l'application, on trouvera les lignes suivantes :

```
switch ($op) {  
default:  
case "langselect":  
    $title = _INSTALL_L0;  
    ...  
}
```

Ici, le titre prendra automatiquement la valeur de la constante définie dans le fichier de langue. Il s'agit de "Bienvenue dans l'assistant XOOPS 2.0" dans notre cas.

## Types de données

PHP dispose de quatre types de données simples : des booléens, des entiers, des nombres à virgule flottante et des chaînes de caractères.

**Remarque**

Bien que vous ne définissiez pas de type pour vos variables, PHP en gère un en interne. En fait, PHP donne dynamiquement un type à la variable selon la valeur que vous lui assignez. Ce type peut changer au cours de l'exécution selon les valeurs affectées.

Vous pouvez connaître le type d'une donnée grâce à la fonction `gettype()` :

```
<?php
echo gettype("chaîne de caractères") ;
// Affiche string
?>
```

Il est aussi possible d'utiliser des fonctions d'accès rapide telle que `is_string()`, qui renvoie `TRUE` si la valeur en argument est une chaîne de caractères et `FALSE` dans le cas contraire.

```
<?php
$var = 12;
if (is_string($var)) {
    echo "chaîne de caractères" ;
} else {
    echo "autre type" ;
}
// Affiche autre type
?>
```

De la même façon, les fonctions `is_double()` et `is_float()` vérifient si la valeur est un nombre à virgule flottante, les fonctions `is_int()` et `is_integer()` vérifient si la valeur est un nombre entier. Les fonctions `is_boolean()`, `is_array()`, `is_null()`, `is_object()` et `is_resource()` vérifient respectivement si la valeur est un booléen, un tableau, la valeur `NULL`, un objet ou une ressource interne.

## Booléens (*boolean*)

Un booléen est une valeur pouvant être soit vraie, soit fausse. Le mot-clé `TRUE` désigne un booléen vrai, et le mot-clé `FALSE` un booléen faux. Ces mots-clés sont insensibles à la casse (ils peuvent être mis en majuscules comme en minuscules).

```
<?php
$bool = TRUE ; // Booléen vrai
$bool = FALSE ; // Booléen faux
?>
```

## Les nombres entiers (*integer*)

Les nombres entiers peuvent être entrés tels quels dans le code. Les entiers négatifs sont à précéder du symbole `-`.

Un entier commençant par un chiffre de 1 à 9 sera interprété selon la base décimale habituelle. C'est la notation que vous utilisez tous les jours. S'il commence par un zéro, il sera compris en base octale (015 sera interprété comme le nombre 13 en base décimale). S'il commence par `0x`, il sera interprété selon la base hexadécimale (`0x1A` sera interprété comme le nombre 26 en base décimale).

```
<?php
```



```
$nombre = 45;
$nb_negatif = -15 ;
$nb_hexa = 0x1A ;
?>
```

La taille des entiers dépend de la plate-forme utilisée, mais la valeur maximale est généralement de l'ordre de 2 milliards (c'est un entier signé de 32 bits). PHP ne reconnaît pas les entiers non signés.

## Les nombres flottants (*double, float*)

Les nombres à virgule flottante sont aussi interprétés directement par le moteur PHP. Ce sont des nombres plus grands que ne peut l'être un entier (un peu plus de 4 milliards sur les systèmes 32 bits), ou comportant une partie décimale.

Un nombre à virgule flottante comporte soit un point (équivalent anglais de la virgule), soit un e (majuscule ou minuscule) séparant l'exposant.

```
<?php
$nombre = 3.14159;
$nombre = 5e7;
$nombre = 1.000;
?>
```

Les nombres à virgule flottante sont par définition imprécis. Pour les gérer, PHP fait des approximations. Vous ne devriez normalement pas vous en soucier car l'erreur d'approximation est négligeable, mais il peut être important de le remarquer si vous comparez deux nombres à virgule flottante : même si pour vous ils sont égaux, ce n'est pas forcément le cas pour PHP à cause des erreurs d'arrondis et d'approximation.

## Les chaînes de caractères (*string*)

### Note

Les lignes qui suivent ne décrivent que les syntaxes qui permettent de créer et d'utiliser des chaînes de caractères. Pour plus d'informations sur les traitements de chaînes de caractères, vous pouvez vous reporter au chapitre 5.

Nous avons vu précédemment que les chaînes de caractères sont généralement délimitées par des guillemets.

```
<?php
$chaine = "Son livre a déclenché la légende";
echo $chaine;
?>
```

## Interprétation des variables

À l'intérieur d'une chaîne entre guillemets, les variables sont automatiquement remplacées par leur valeur. Ainsi, dans le code suivant, la variable `$objet` est automatiquement remplacée par sa valeur quand on affecte `$chaîne` (résultat à la figure 3-5).

```
<?php
$objet = "livre";
$chaîne = "Son $objet a déclenché la légende";
echo $chaîne;
// Affiche Son livre a déclenché la légende
?>
```

Figure 3-5

*Interprétation  
des variables*



Si vous utilisez des variables complexes comme les tableaux ou objets, vous pouvez délimiter l'appel avec des accolades :

```
<?php
$objet = new stdClass() ;
$objet->propriete = "livre" ;
$chaîne = "Son {$objet->propriete} a déclenché la légende";
$tableau['index'] = "livre" ;
$chaîne = "Son {$tableau['index']} a déclenché ...";
?>
```

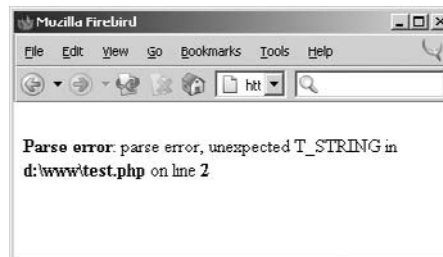
## Le caractère d'échappement (ou protection)

Essayer d'utiliser un texte contenant des guillemets dans une chaîne de caractères elle-même délimitée par des guillemets provoque l'erreur illustrée à la figure 3-6 :

```
<?php
$chaîne = "Son livre "le premz" a déclenché la légende";
// Renvoie un message d'erreur
?>
```

Figure 3-6

*Message d'erreur  
type*



Mettons-nous quelques instants à la place de l'interpréteur PHP. Celui-ci lit le signe égal (=) suivi de guillemets, il en conclut qu'il va recevoir une chaîne de caractères et qu'il va devoir l'assigner à la variable `$chaîne`. Il sait aussi que la chaîne de caractères est délimitée de part et d'autre du même signe (dans ce cas les guillemets). Il alloue donc la valeur "Son livre" à la variable `$chaîne` puis lit la suite, qui ne correspond à rien, donc provoque une erreur.

Pour éviter cela, on protège les guillemets avec le caractère d'échappement, une barre oblique inverse (symbole `\`). Dans ce cas, l'interpréteur PHP ne considérera pas les guillemets comme un signe de fin puisqu'ils sont protégés. On aura donc :

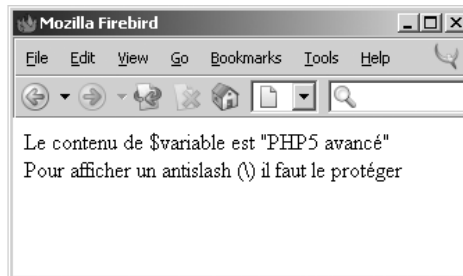
```
<?php
$chaîne = "Son livre \"le premez\" a déclenché la légende";
?>
```

De la même façon, tous les caractères normalement interprétés par PHP peuvent être protégés pour être utilisés tels quels en les préfixant par une barre oblique inverse. C'est en particulier le cas avec `$` pour éviter que ce qui suit ne soit interprété comme une variable à remplacer, et avec `\` pour éviter qu'il soit associé au caractère suivant et ne le protège. Le résultat du code suivant est illustré à la figure 3-7 :

```
<?php
$variable = "PHP5 avancé";
$chaîne = "Le contenu de \"$variable\" est \"$variable\"<br>";
echo $chaîne;
echo "Pour afficher un antislash (\\) il faut le protéger" ;
?>
```

**Figure 3-7**

*Le caractère d'échappement*



D'autres combinaisons peuvent être utilisées. Les plus courantes sont `\n` (caractère de changement de ligne), `\r` (retour chariot) et `\t` (tabulation).

#### Attention

Dans le cas d'une page HTML, il faut bien noter qu'un changement de ligne fait par `\n` affichera un changement de ligne dans le code source du résultat mais pas dans le rendu du navigateur. Un changement de ligne en HTML se fait par `<br>`.

Si vous souhaitez utiliser des caractères spéciaux, vous pouvez les référencer par leur code caractère. Ainsi, `\x20` représente le caractère espace (caractère 20 en hexadécimal dans la table ASCII).

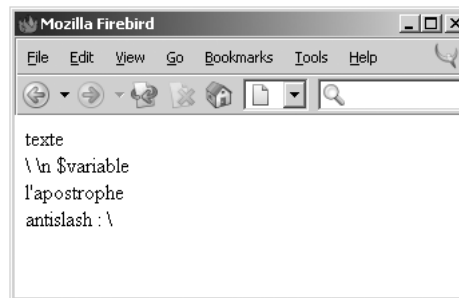
### Délimitation par apostrophes

Lorsqu'une chaîne contient beaucoup de caractères interprétables comme des guillemets ou des barres obliques inverses, il devient complexe de protéger chaque caractère.

Il est alors possible de délimiter une chaîne de caractères avec des apostrophes (caractère `'`). Dans ce cas, seules les apostrophes sont à protéger ; tous les autres caractères peuvent être mis directement dans la chaîne et seront pris tels quels sans être interprétés. Il est toutefois possible (mais pas obligatoire) de protéger aussi la barre oblique inverse. Le résultat du code suivant est donné à la figure 3-8.

```
<?php
echo 'texte<br>' ; // Affiche texte
echo '\ \n $variable<br>' ; // Affiche \ \n $variable
echo '\\'apostrophe<br>' ; // Affiche l'apostrophe
echo 'antislash : \\<br>' ; // Affiche antislash : \
?>
```

**Figure 3-8**  
*Utiliser les  
apostrophes*



#### Note

Certains conseillent d'employer de préférence cette syntaxe quand votre chaîne ne contient ni variables ni caractères spéciaux (comme des retours à la ligne). Vous éviterez ainsi d'éventuelles erreurs de copier-coller. Il arrive en effet qu'on recopie sans faire attention des chaînes contenant le symbole `$`. Avec des apostrophes, il restera tel quel et ne sera pas interprété. Il existe aussi une différence de performance à l'avantage de l'apostrophe (puisque PHP n'a pas à interpréter le contenu), mais elle est négligeable.

### Syntaxe heredoc

Un autre moyen de délimiter les chaînes est d'utiliser la syntaxe dite *heredoc*. Un tel texte est délimité à l'ouverture par trois symboles `<` et un identifiant. Le texte est fermé par une ligne ne contenant que l'identifiant suivi d'un point-virgule.

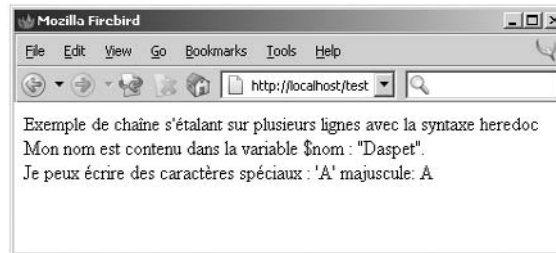
La syntaxe *heredoc* se comporte exactement comme une chaîne à guillemets, sans les guillemets. Cela signifie que vous n'avez pas à échapper les guillemets dans cette syntaxe. Les variables sont remplacées par leur valeur, et le même soin doit leur être apporté que dans les chaînes à guillemets. L'affichage du code suivant peut être vu à la figure 3-9.

```
<?php
$nom = 'Daspet';

$texte =<<<icijemetscequejeveux
Exemple de chaîne
s'étalant sur
plusieurs lignes
avec la syntaxe heredoc<br>
Mon nom est contenu dans la variable \$nom : "$nom". <br>
Je peux écrire des caractères spéciaux : 'A' majuscule: \x41
icijemetscequejeveux;

echo $texte;
?>
```

**Figure 3-9**  
*Syntaxe heredoc*



### Important

Le délimiteur de fin ne doit contenir que l'identifiant et le point-virgule, pas d'espace ou d'indentation, que ce soit avant ou après.

### Accéder à un caractère d'une chaîne

Il est possible d'accéder directement à un caractère dans une chaîne en le référant par sa position. Il suffit alors d'ajouter la position entre accolades après le nom de la variable :

```
<?php
$chaîne = "hello world" ;
echo $chaîne{1} ;
// Affiche e
?>
```

### Attention

Les positions sont calculées à partir de l'index 0. Ici, 1 représente le deuxième caractère.

## Les tableaux (array)

En plus des types de données simples, PHP propose une façon de grouper ces données : les tableaux.

PHP permet deux types de tableaux : les tableaux indexés numériquement et les tableaux associatifs. Il n'est pas nécessaire de déclarer leur taille lors de la déclaration, elle est gérée par PHP.

### Note

Vous ne trouverez ici que les syntaxes de base liées à la définition et à l'utilisation des tableaux. Les fonctions de traitement seront décrites au chapitre 6.

### Tableaux indexés numériquement

Un tableau indexé est une simple liste d'éléments. On peut la créer grâce au mot-clé `array()`, en séparant les valeurs par des virgules.

```
<?php
$tableau = array( 12250, 15555, 12000, 21300, 25252, 20010, 8460);
$tab2 = array( $variable, "texte", 153, 56 );
?>
```

Dans la liste, chaque élément est repéré par sa position, son index. Cet index sera bien entendu unique.

Figure 3-10

*Exemple de tableau indexé numériquement*

**Tableau indexé numériquement**

Index du tableau	Element du tableau
0	12550
1	15555
2	12000
3	21300
4	25252
5	20010
6	8460
7	8500
8	14522
9	28010
10	35120
11	12000

Chacune de ces variables est rangée dans une case. Pour y accéder, il faut indiquer le nom du tableau et la case (l'index) de la variable désirée. La syntaxe est la suivante :

```
$nom_variable_tableau[index]
```

### Important

On notera que les index commencent à partir de 0 et non de 1. Le premier élément est `$tableau[0]`.

Il est possible de lire et écrire directement dans une case grâce à cette syntaxe :

```
<?php
// On commence par assigner des valeurs au tableau
$tableau[0] = 12250;
$tableau[10] = 35120;

// Puis on peut le manipuler
echo $tableau[10] ;
?>
```

Il existe une syntaxe réduite pour ajouter un à un les éléments sans avoir à manipuler les index. Il suffit d'omettre l'index, mais en laissant les crochets. Ces trois codes sont équivalents :

```
<?php
$tabA = array( 1, 2, 3 ) ;

$tabB[0] = 1 ; $tabB[1] = 2 ; $tabB[2] = 3 ;

$tabC[] = 1 ; $tabC[] = 2 ; $tabC[] = 3 ;
?>
```

### Tableaux associatifs

Les tableaux numériques indexés sont faciles d'utilisation, mais peuvent se révéler peu pratiques pour gérer la signification et la place des valeurs contenues. Effectivement, nous devons forcément passer par le numéro correspondant à un élément.

L'alternative proposée par PHP est le tableau associatif. Celui-ci associe une chaîne de caractères à un élément. On parle alors de tableau associatif ou de table de hachage.

La figure 3-11 vous montre un exemple de tableau associatif contenant des informations sur un des utilisateurs présents sur votre base de données.

Figure 3-11

Exemple de tableau associatif

### Tableau associatif

nom	PIERRE de GEYER
prenom	Cyril
sexe	m
ville	Paris
cp	75005
telephone	0143819291
travail	informatique
Pays	france

Pour créer un tableau grâce au mot-clé `array()`, il faut donner la clé et l'élément, séparés par `=>`.

```
<?php
$tab = array(
    "prenom" => "Cyril" ,
    "ville" => "Paris" ,
    "travail" => "informatique"
) ;
?>
```

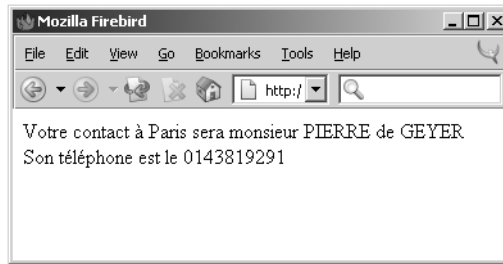
Comme pour les tableaux indexés, il est possible de référencer directement un élément grâce à sa clé. On peut alors considérer que les tableaux indexés ne sont que des tableaux associatifs dont les clés sont numériques et attribuées automatiquement. Le code suivant est affiché dans la figure 3-12.

```
<?php
$tableau['nom'] = "PIERRE de GEYER";
$tableau['prenom'] = "Cyril";
$tableau['ville'] = "Paris";
$tableau['cp'] = "75005";
$tableau['sexe'] = "m";
$tableau['telephone'] = "0143819291";
$tableau['travail'] = "informatique";
$tableau['Pays'] = "france";

echo "Votre contact à {$tableau['ville']} sera monsieur ";
echo $tableau['nom'];
echo "<br>Son téléphone est le ";
echo $tableau['telephone'];
?>
```



**Figure 3-12**  
*Utilisation  
d'un tableau*



#### Remarque

Le mixage des index numériques et des index associatifs est possible.

### Tableaux multidimensionnels

Nous venons de voir comment créer des tableaux simples et des tableaux associatifs. Il est également possible de créer des tableaux à plusieurs dimensions pour stocker, par exemple, une matrice.

En PHP, ces tableaux multidimensionnels sont des tableaux de tableaux, c'est-à-dire qu'un premier tableau contiendra un ensemble de tableaux.

La figure 3-13 vous montre un exemple de tableau multidimensionnel contenant des informations relatives à une matrice.

**Figure 3-13**  
*Tableau  
multidimensionnel*

### Tableau multidimensionnel

Index du tableau	Element du tableau
0	[10,5,45,65,58,41]
1	[5,1,4,165,58,1]
2	[10,4,35,5,48,141]
3	[1,8,85,29,46,137]
4	[6,221,54,78,13,65]
5	[6,32,156,465,32,1]
6	[10,5,45,65,58,41]
7	[10,5,45,65,58,41]
8	[10,5,45,65,58,41]
9	[10,5,45,65,58,41]
10	[10,5,45,65,58,41]
11	[10,5,45,65,58,41]

Pour manipuler des tableaux à  $n$  dimensions, il faudra indiquer  $n$  indices. Ces différents indices permettront à PHP de retrouver le bon élément dans votre tableau. Nous allons baser nos exemples sur un tableau à deux dimensions, mais il est tout à fait possible de lui définir  $n$  dimensions et le raisonnement sera le même.

Dans l'exemple suivant, nous allons définir une matrice 3\*2 :

```
<?php
$matrice[0][0] = 5;
$matrice[0][1] = 4;
$matrice[1][0] = 2;
$matrice[1][1] = 3;
$matrice[2][0] = 8;
$matrice[2][1] = 2;
?>
```

PHP disposera alors dans sa mémoire de la matrice illustrée à la figure 3-14. Le premier indice concerne la partie verticale et le second la partie horizontale comme défini dans la figure 3-15.

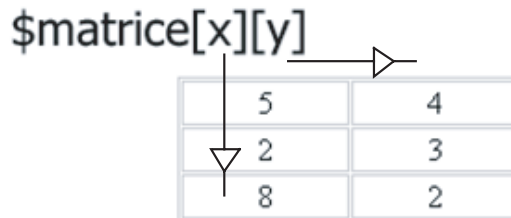
Figure 3-14

Matrice 3\*2

5	4
2	3
8	2

Figure 3-15

Manipulation de  
tableau à deux  
dimensions



Il est toujours possible d'utiliser le mot-clé `array()` pour créer notre tableau, la structure de tableaux dans un tableau est alors clairement visible :

```
<?php
$matrice = array(
    array(5,4),
    array(2,3),
    array(8,2)
);
?>
```

Nous avons donné en exemple des tableaux multidimensionnels avec des index numériques, mais il est de la même façon possible d'utiliser des tableaux associatifs.

## Transtypage

PHP permet de manipuler toutes les données sans déclarer leur type ni s'en soucier. Quand une donnée d'un certain type est attendue et qu'une donnée d'un autre type est fournie, PHP fait une conversion automatique.

Pour la plupart des manipulations, vous n'aurez pas à vous soucier des types. Il est toutefois important de bien connaître les règles de conversion, afin de ne pas être surpris d'un résultat. Ces règles vous seront par exemple particulièrement utiles si vous comparez deux données de types différents (auquel cas PHP fait les conversions nécessaires pour mettre les deux données sous le même type ; mais attention, il peut ne pas faire exactement ce que vous pensiez).

### Règles de conversion

#### Chaîne de caractères vers un nombre

Quand PHP convertit un texte en nombre, il regarde en début de la chaîne de caractères s'il trouve un nombre connu. Si c'est le cas, il fait une conversion directe :

```
echo "3" + 1 ; // Affiche 4
```

PHP interprétera ainsi toutes les syntaxes classiques de représentation numérique. On trouve par exemple le point comme séparateur décimal, la notation exponentielle ou le signe moins pour un nombre négatif :

```
echo 1 + "-1.3e3" ; // Affiche -1299
```

Il est important de noter que PHP lit le texte jusqu'à ce qu'un des caractères ne puisse plus être interprété. Ce qui a déjà été lu servira à la conversion. Ainsi, "3 petits cochons" donnera le chiffre 3.

```
echo 1 + "3 petits cochons" ; // Affiche 4
```

Si aucune interprétation ne peut être faite, la chaîne de caractères sera transformée en une valeur nulle.

```
echo 1 + "petits cochons" ; // Affiche 1
```

#### Tous types vers booléen

Les booléens ne permettent que deux valeurs, une valeur vraie et une valeur fausse.

Sont considérés comme des valeurs fausses :

- les constantes FALSE et NULL ;
- une chaîne de caractères vide ;
- le nombre entier ou à virgule flottante 0 ;
- une chaîne de caractères contenant uniquement le nombre 0 ;
- un tableau vide ;
- un objet avec aucun champ défini.

Toutes les autres données seront considérées comme « vraies ».

## Conversions à partir d'un booléen

Pour toutes les conversions, un booléen vrai se comportera comme un entier de valeur 1. Un booléen faux se comportera comme un entier de valeur 0.

## Conversion vers une chaîne de caractères

Les nombres entiers ou à virgule flottante prendront leur représentation décimale classique. La constante TRUE s'évaluera comme le nombre 1.

Les valeurs NULL, FALSE ou les nombres nuls prendront comme équivalent une chaîne vide et non le caractère 0.

Les tableaux, objets ou ressources afficheront leur type comme valeur.

### Note

Il existe, pour les objets, un moyen de personnaliser la conversion vers une chaîne de caractères avec la méthode `__toString()`. Vous pourrez trouver une description détaillée de cette fonctionnalité dans le chapitre sur la programmation objet.

## Forcer une conversion

PHP fait automatiquement et de manière satisfaisante toutes les conversions nécessaires entre les différents types de données. Vous pouvez toutefois forcer une conversion en ajoutant le type de donnée entre parenthèses devant la donnée.

```
<?php
echo gettype( "12" ); // Affiche string
echo gettype( (integer) "12" ); // Affiche integer
echo gettype( (string) 12 ); // Affiche string
?>
```

Si cette syntaxe vous paraît peu claire, vous pouvez utiliser la fonction `settype()`. Elle prend en premier argument la donnée à convertir et en second argument le type vers lequel faire la conversion.

```
<?php
echo gettype( "12" ); // Affiche string
echo gettype( settype("12", 'integer') ); // Affiche integer
echo gettype( settype(12, 'string') ); // Affiche string
?>
```

Vous pouvez aussi passer par des fonctions dédiées pour les conversions les plus courantes. Ainsi, `intval()` convertit une donnée en paramètre vers un entier, `strval()` fait la conversion vers une chaîne de caractères, `floatval()` et `doubleval()` font la conversion vers un nombre à virgule flottante.

```
<?php
echo gettype( "12" ); // Affiche string
echo gettype( intval( "12" ) ); // Affiche integer
echo gettype( strval( 12 ) ); // Affiche string
?>
```



# 4

## Traitements de base

---

Ce chapitre décrit les principales opérations et les traitements de base que l'on peut effectuer sur les structures abordées au chapitre précédent. Il s'agit notamment des différents opérateurs qui nous permettront de définir et de manipuler des variables, ainsi que des structures de contrôles qui sont indispensables à la réalisation du programme. Enfin, nous aborderons la création de fonctions.

### Les opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables. Ils permettent notamment d'effectuer des opérations, d'affecter ou de comparer des valeurs, etc.

#### *Opérateurs d'affectation*

L'opérateur d'affectation est le signe égal, nous l'avons aperçu dans les exemples précédents. La variable avant le symbole égal (=) prend la valeur spécifiée après.

```
<?php
// Assignement de valeurs à des variables
$nom = 'Mandriva';
// Ici $nom prend la valeur Mandriva
$nombre = 10 ;
?>
```

L'opérateur d'affectation = est une opération qui renvoie une valeur. La valeur renvoyée est la valeur affectée, ce qui permet de faire des affectations en chaîne :

```
<?php
$j = $i = 5 ;
```

```
// Est comme :  
$j = ($i = 5) ;  
// Équivalent à  
$j = 5 ;  
echo ($i = 5) ; // Affiche 5  
?>
```

**Note**

Nous utilisons des parenthèses dans la commande `echo ($i=5)` ; de notre exemple. Celles-ci ne sont pas obligatoires mais rendent la relecture du code plus aisée.

### Affectation par copie et références

Par défaut, l'affectation des variables se fait par copie, c'est-à-dire que la valeur à droite du signe `=` est copiée pour être affectée à la variable de gauche. Par la suite, les deux valeurs sont indépendantes ; modifier l'une ne modifiera pas l'autre.

```
<?php  
$origine = 1 ;  
$copie = $origine ;  
$copie = $origine + 1 ;  
echo $copie ; // Donne 2 (1+1)  
echo $origine ; // Donne 1 (sa valeur n'a pas été modifiée)  
?>
```

**Note**

La seule exception à cette règle concerne les objets. Vous trouverez plus de détails à ce sujet dans le chapitre sur la programmation objet.

Il est possible, au lieu de cela, de créer un lien fort entre les deux variables en ajoutant un `&` juste derrière le signe d'affectation. Dans ce cas, on dit que les deux noms référencent la même valeur. Pour ceux qui viennent du langage C, la notion de référence n'est pas tout à fait la même que la notion de pointeur ; elle se rapproche beaucoup plus de la notion de lien dans les systèmes de fichiers Unix.

Dès lors, on peut modifier la valeur référencée par une variable et la relire via une autre : il s'agira de la même.

```
<?php  
$origine = 1 ;  
$copie =& $origine ;  
$copie = $copie +1;  
  
echo $copie ; // Donne 2 (1+1)  
echo $origine ; // Donne 2 aussi  
// Puisque les deux noms correspondent en fait à la même valeur  
?>
```

Pour effacer une référence, il vous suffit d'utiliser `unset()` sur la variable. Vous effacez alors le lien entre le nom et la valeur. Les éventuels autres noms utilisant la même valeur ne sont pas affectés (voir script suivant et figure 4-1).

```
<?php
$origine = 1 ;
$reference =& $origine ;
$origine = 2 ;
echo 'Valeur de $reference : ', $reference, '<br>';
// Affiche 2
unset($origine) ;
echo 'Valeur de $origine : ', $origine, '<br>';
// N'affiche plus rien
echo 'Valeur de $reference : ', $reference, '<br>';
// Affiche toujours la valeur 2
?>
```

Figure 4-1

*Les références*



## Opérateurs arithmétiques

Les opérateurs arithmétiques en PHP ne nécessitent pas de présentation particulière, car ils sont les opérateurs mathématiques traditionnels. Le tableau 4-1 nous en donne un rapide aperçu.

**Tableau 4-1 Opérateurs arithmétiques**

( \$a = 9; \$b = 4; )				
Opérateur	Opération	Exemple	Description	Résultat
+	Addition	echo \$a + \$b;	Calcule la somme	13
-	Soustraction	echo \$a - \$b;	Calcule la différence	5
*	Multiplication	echo \$a * \$b;	Calcule le produit	36
/	Division	echo \$a / \$b;	Calcule la division	2.25
%	Modulo	echo \$a % \$b;	Calcule le modulo	1

Dans nos exemples, nous affichons à l'écran le résultat renvoyé par les opérations. Notons qu'il serait également possible de renvoyer le résultat dans une variable :

```
<?php
$a = 9 ;
```



```
$b = 4 ;
$resultat = $a + $b;
echo $resultat; // Affiche 13
?>
```

La valeur stockée dans la variable `$resultat` est le résultat obtenu par l'addition des valeurs des variables `$a` et `$b`.

### L'utilité du modulo (%)

Modulo renvoie le reste de la division. Par exemple, pour  $15 \% 2$ , on fait le calcul suivant :  $15 = 7 * 2 + 1$ . Le modulo de  $15\%2$  est donc 1.

Une des applications principales est de savoir si un nombre est pair ou impair. Effectivement, on sait que la parité peut être connue en regardant le reste d'une division par deux.

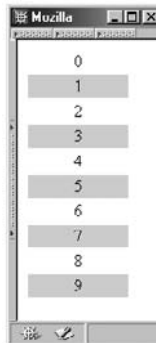
```
15 = 7 * 2 + 1 // Le reste est égal à 1 donc 15 est impair.
128 = 64 * 2 + 0 // Le reste est égal à 0 donc 128 est pair.
<?php
$parite = $nbe % 2;
// Si $parite est égal à 0 cela signifie que $nbe est pair
?>
```

Cela nous permet, par exemple, d'afficher une ligne sur deux d'un tableau HTML avec une couleur de fond différente (figure 4-2).

```
<?php
$i = 0;
echo '<table width="100">';
while ($i < 10){
    if (($i % 2)==0){ // Cas d'une ligne paire
        echo "<tr><td bgcolor=#FFFFFF align=center>$i</td></tr>";
    } else { // Cas d'une ligne impaire
        echo "<tr><td bgcolor=#CCCCCC align=center>$i</td></tr>";
    }
    $i++;
}
echo '</table>';
?>
```

**Figure 4-2**

*Exemple  
d'utilisation  
de modulo*



## Incrémentation

Comme en PERL et en C, PHP possède quelques raccourcis pour vous éviter d'écrire des instructions d'incrémentations telles que :

```
$i = $i+1;
```

Utilisez ++ ou -- respectivement pour incrémenter ou décrémenter une variable d'une seule unité (voir tableau 4-2).

```
<?php
$i = 1 ;
$i = $i+1; // Est équivalent à :
$i++;

$i = $i-1; // Est équivalent à :
$i--;
?>
```

Ces opérateurs peuvent figurer devant ou derrière la variable :

- ++\$a : la variable est incrémentée puis évaluée,
- \$a++ : la variable est évaluée puis incrémentée.

Cette subtilité pourra nous intéresser dans des conditions ou boucles. Cela permet notamment d'inclure l'incrémentations dans la condition.

**Tableau 4-2 Opérateurs d'incrémentations**

Opérateur	Opération	Exemple	Résultat
++	incrémentations	echo \$a++. "#". \$a; echo ++\$a. "#". \$a;	9#10 10#10
--	décrémentations	echo \$a--. "#". \$a; echo --\$a. "#". \$a;	9#8 8#8

## Opérateurs combinés

Il est possible de cumuler l'opérateur d'affectation avec un opérateur arithmétique (voir tableau 4-3).

**Tableau 4-3 Opérateurs combinés**

Opérateur	Opération
+=	Ajoute la valeur de droite à la valeur de gauche. Met le résultat dans la variable de gauche.
-=	Soustrait la valeur de droite à la valeur de gauche. Met le résultat dans la variable de gauche.
*=	Multiplie la valeur de droite par la valeur de gauche. Met le résultat dans la variable de gauche.
.=	Concatène les valeurs. Voir le paragraphe suivant.
X=	En généralisant, si X est un opérateur (+, -, *, /, &, % ou  ), « \$a X= \$b » est équivalent à « \$a = \$a X \$b ».

```
<?php
$a = 10;
$b = 7;

$a *= 5; // $a est égal à 50
$b += $a; // $b est égal à 57

?>
```

## La concaténation

L'opération de concaténation est très importante car elle est souvent utilisée. Cet opérateur vous permet par exemple d'empiler des informations dans une variable.

**Tableau 4-4 Opérateurs de concaténation**

(\$a= 'un ' ; \$b=' texte' ;)			
Opérateur	Opération	Exemple	Résultat
.	Concaténation	echo \$a.\$b	Un texte
.=	Concaténation et assignation	\$a .= \$b;	// \$a == un texte

```
<?php
$a = 'hello ';
$a.= 'world';
// $a est égal à 'hello world'
$table = 'users';
$id = 5;
$sql = 'SELECT * FROM '. $table ." WHERE id='$id'";

echo 'Il est '. date('G \h i') . ' et il fait beau';
?>
```

## Opérateurs de comparaison

Les opérateurs de comparaison sont principalement utilisés en combinaison avec les structures conditionnelles (if, for, while, etc.) que nous verrons peu après. Une liste des opérateurs de comparaison est donnée au tableau 4-5.

Ces opérateurs permettent de comparer deux termes et de renvoyer un booléen vrai ou faux (TRUE ou FALSE) selon la véracité de la comparaison.

**Tableau 4-5 Opérateurs de comparaison**

Opérateur	Signification	Exemple	Description
==	Égal à	\$a == \$b;	Renvoie TRUE si \$a est égal à \$b.
<	Inférieur à	\$a < \$b;	Renvoie TRUE si \$a est plus petit que \$b.

Tableau 4-5 Opérateurs de comparaison (suite)

Opérateur	Signification	Exemple	Description
>	Supérieur à	<code>\$a &gt; \$b;</code>	Renvoie TRUE si \$a est plus grand que \$b.
<=	Inférieur ou égal à	<code>\$a &lt;= \$b;</code>	Renvoie TRUE si \$a est inférieur ou égal à \$b.
>=	Supérieur ou égal à	<code>\$a &lt;= \$b;</code>	Renvoie TRUE si \$a est supérieur ou égal à \$b.
!=	Différent de	<code>\$a != \$b ;</code>	Renvoie TRUE si \$a est différent de \$b.
===	Égal à, en type et en valeur	<code>\$a === \$b ;</code>	Renvoie TRUE si \$a est égal à \$b et si les deux variables ont le même type.
!==	Différent en valeur ou différent en type	<code>\$a !== \$b ;</code>	Renvoie TRUE si \$a a une valeur différente de celle de \$b ou si \$a n'a pas le même type que \$b.

Exemple :

```
<?php
$a = 9;
$b = 5;
// $a == $b renverra 0 (faux)
// $a != $b renverra 1 (vrai)
// $a < $b renverra 0 (faux)
?>
```

#### Attention

Il ne faut pas confondre l'opérateur d'affectation = et l'opérateur de comparaison ==. Cela entraîne souvent des boucles infinies (`while ($i=5)`) car `$i=5` renvoie toujours 5, donc jamais FALSE.

#### L'opérateur « === »

La facilité d'utilisation des variables peut entraîner des problèmes de sens. Ainsi, PHP considère comme égaux la chaîne de texte '2' et l'entier 2. Cela peut être problématique dans certains cas où vous aimeriez faire une distinction claire entre les différents types de données. Pour ces cas, il existe l'opérateur de comparaison ===. Il vérifie l'égalité des valeurs mais aussi l'égalité des types.

```
<?php
$a = '34';
$b = 34 ;
$a == $b; // Renvoie TRUE
$a === $b ; // Renvoie FALSE
?>
```

De même, il existe un opérateur !== qui vérifie la différence entre deux valeurs en vérifiant la concordance des types.

## Opérateurs logiques

Les opérateurs logiques servent énormément dans les structures de contrôle. Le tableau 4-6 présente les plus courants.

**Tableau 4-6 Opérateurs logiques**

Opérateur	Exemple	Évalué à vrai (TRUE) quand :
!	! \$b;	\$b ne renvoie pas TRUE.
&&	\$a && \$b;	\$a et \$b renvoient TRUE.
	\$a    \$b;	Au moins l'un des deux renvoie TRUE.
AND	\$a AND \$b;	\$a et \$b renvoient TRUE.
OR	\$a OR \$b;	Au moins l'un des deux renvoie TRUE.
XOR	\$a XOR \$b;	L'un des deux, et uniquement l'un des deux, renvoie TRUE.

### Note

Les opérateurs || et OR, ainsi que && et AND donnent exactement le même résultat. La seule différence réside dans les priorités d'applications qui sont décrites plus loin. OR et AND ont une priorité faible alors que || et && ont une priorité forte.

Les opérateurs logiques sont utilisés pour combiner les résultats de conditions logiques entre eux. En voici un exemple :

```
<?php
$a = 9;
$b = 5;
$c = 11;
$resultat = ($b < $a) && ($a < $c );
// Renverra 1 (vrai)
// On indique ici que $a est compris entre $b et $c.
$resultat = ($b < $a) XOR ($a < $c );
// Renverra 0 (faux)
// On indique ici que $a doit être plus grand que $b ou que $a doit être plus petit
// que $c mais pas les deux.
?>
```

Toutes ces conditions sont traitées par priorité, aussi nous vous recommandons d'entourer chacune des conditions par des parenthèses pour éviter les erreurs.

## Opérateurs sur les bits

Les opérateurs suivants permettent d'agir directement sur la représentation binaire d'un nombre, et donc sur les bits le composant. La liste des opérateurs bit à bit gérés par PHP est donnée dans le tableau 4-7.

Tableau 4-7 Opérateurs binaires

Opérateur	Signification	Exemple	Description
&	ET	<code>\$a &amp; \$b;</code>	Pour tout x, met à 1 le bit de la position x si les bits correspondants de \$a et de \$b sont à 1.
	OU	<code>\$a   \$b;</code>	Pour tout x, met à 1 le bit de la position x si le bit correspondant de \$a ou de \$b est à 1.
<<	Décalage à gauche	<code>\$a&lt;&lt;\$b;</code>	Pour tout x, met le bit à 1 si le bit (x-\$b) de \$a est à 1.
>>	Décalage à droite	<code>\$a&gt;&gt;\$b;</code>	Pour tout x, met le bit à 1 si le bit (x+\$b) de \$a est à 1.
^	XOR	<code>\$a ^ \$b</code>	Pour tout x, met à 1 le bit de la position x si le bit correspondant de \$a est à 1 ou le bit de \$b est à 1, mais pas les deux.
~	Négation	<code>~ \$a</code>	Pour tout x, met à 1 le bit de la position x si le bit correspondant de \$a est à 0.

Les opérateurs bit à bit permettent de traiter les nombres entiers sous la forme de séries de bits. Ces séries de bits représentent le nombre en question.

```
<?php
$un = 1 ; // S'écrit 1 en binaire
$deux = 2 ; // S'écrit 10 en binaire
$trois = $un | $deux ; // S'écrit 11 en binaire
$quatre = $deux << $un ; // S'écrit 100 en binaire
?>
```

### Priorités entre opérateurs

Il est souvent nécessaire d'imbriquer plusieurs opérateurs. Dans ce cas, il faut tenir compte des priorités pour que l'interpréteur PHP puisse les traiter dans le bon ordre. Le tableau 4-8 récapitule les priorités des différents opérateurs (le premier est le plus prioritaire).

Tableau 4-8 Priorité des opérateurs

1	( ) [ ]
2	-- ++ !
3	* / %
4	+ -
5	< <= >= >
6	== != ===
7	&
8	
9	&&
10	
11	Affectation, opérateurs combinés (= += -=,...)
12	AND
13	XOR

## Structures de contrôle

Les structures de contrôle permettent de répéter certaines actions ou de soumettre certaines exécutions à des conditions. En PHP, leur syntaxe est similaire à celle du langage C.

Ces structures fonctionnent pour la plupart à partir d'un test. Ce test est une expression qui doit renvoyer une valeur comprise comme un booléen. Le plus souvent, on utilisera les opérateurs logiques et de comparaison, mais il est possible d'avoir une expression complexe comprenant des appels de fonctions et des affectations de variables.

### Les conditions

Dans vos scripts, il sera important de pouvoir effectuer une prise de décision, et donc de poser des conditions à l'exécution de telle ou telle action.

Exemple :

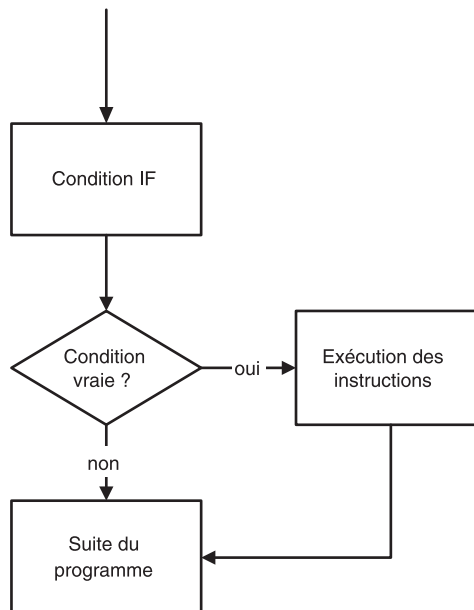
- Si le visiteur a entre 18 et 34 ans, je lui assigne le profil « jeune ».
- Si cette personne a entre 35 et 60 ans, je lui assigne le profil « mature ».

### L'instruction `if(){}`

L'instruction `if(){}`  est la structure de test la plus basique. Elle permet d'exécuter une suite d'instructions en fonction d'une condition. La condition entre parenthèses est évaluée et les instructions situées entre les accolades sont exécutées seulement si l'évaluation aboutit à `TRUE` (voir figure 4-3).

Figure 4-3

*Instruction `if(){}`*



En PHP, cela s'écrit de la façon suivante :

```
if (condition)
{
    instructions
}
```

Exemple :

```
<?php
$age = 25;
if($age < 18)
{
    echo 'Vous êtes trop jeune pour entrer ici';
    exit(); // La fonction exit() arrête l'exécution du script
}
?>
```

La condition peut être complexe ; son unique prérequis est de renvoyer une valeur qui sera interprétée comme un booléen. Ainsi, si `$age>18` et `$age<35` sont des tests valables, il est possible de les combiner grâce à l'opérateur `&&` pour en faire une expression unique qui ne renverra TRUE que si les deux composantes renvoient TRUE : `($age>18 && $age<35)`.

```
<?php
$age = 25;
if($age > 18 && $age < 35)
{
    echo 'Votre profil est : jeune adulte';
}
?>
```

La clause `else{}`

Nous n'avons actuellement vu que le cas où la condition est vérifiée. On peut également spécifier une suite d'instructions à exécuter lorsque la condition n'est pas réalisée, avec l'instruction `else{}` (voir figure 4-4).

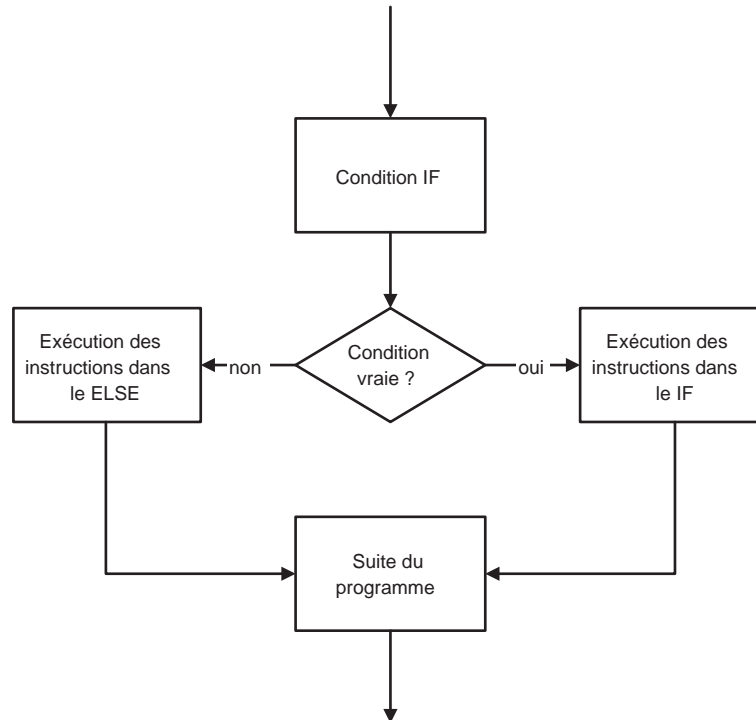
```
if (condition)
{
    instructions si la condition est vérifiée.
}else{
    instructions si la condition n'est pas vérifiée.
}
```

Exemple :

```
<?php
$temps = 'moche';
if($temps == 'ensoleillé')
{
    echo 'Il fait beau';
}else{
    echo 'Il ne fait pas beau';
}
?>
```



Figure 4-4  
Condition if - else



### L'instruction elseif(){}

Enfin, il est possible d'enchaîner une série d'instructions if (sans avoir besoin de les imbriquer) à l'aide de l'instruction elseif (figure 4-5).

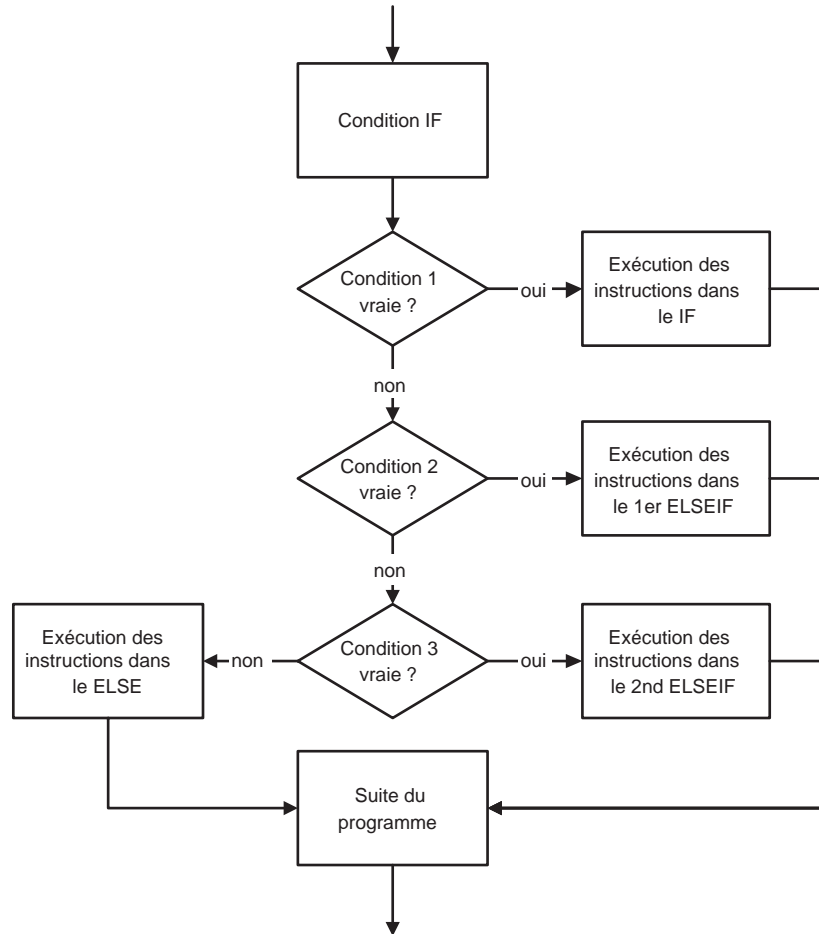
```
if (condition1)
{
    instructions si la condition1 est vérifiée.
}elseif (condition2){
    instructions si la condition2 est vérifiée.
}elseif(condition3){
    instructions si la condition3 est vérifiée.
}else{
    instructions si les conditions ne sont pas vérifiées.
}
```

### Exemple :

```
<?php
$nombre = 2 ;
if ($nombre > 1)
{
    echo "$nombre est supérieur à 1";
} elseif ($nombre < 1) {
```

```
    echo "$nombre est inférieur à 1";  
  } else {  
    echo "$nombre est égal à 1";  
    // Ici on sait que $nombre est égal à 1 car s'il n'est  
    // ni supérieur, ni inférieur à 1 c'est qu'il est égal à 1.  
  }  
  ?>
```

**Figure 4-5**  
*Instruction if - elseif*



Voyons maintenant un exemple qui nous permettra d'utiliser une nouvelle fonction : la fonction `mt_rand()`.

```
<?php  
$salaire = mt_rand(1,6000) ;  
if ($salaire < 1000) {  
    echo 'Vous êtes payé en dessous du SMIC';  
}
```

```
} elseif ($salaire < 3000) {
    echo 'Vous êtes raisonnablement bien payé';
    // On notera ici qu'il n'est pas nécessaire de répéter la
    // condition impliquant que le salaire est supérieur à 1000
} else {
    echo 'Contactez-moi, votre travail m'intéresse !';
}
?>
```

La fonction `mt_rand()` prend en argument deux paramètres, le minimum et le maximum, et fournit une valeur aléatoire comprise entre ces deux valeurs. Vous trouverez tous les détails sur `mt_rand()` au chapitre 7, traitant des fonctions usuelles.

### Les accolades dans les conditions

Comme vous avez pu le constater, des accolades suivent la condition dans nos exemples. Il est cependant possible de s'en passer quand une seule instruction suit la condition.

Exemple :

```
<?php
if($temps == 'ensoleillé')
    echo 'Il fait beau';
?>
```

Si les accolades ne sont pas placées, seule la première instruction sera effectuée.

```
<?php
if($temps == 'ensoleillé')
    echo 'Il fait beau';
echo ' et chaud'; // Cette instruction sera toujours réalisée
?>
```

#### Note

Nous vous conseillons de toujours mettre les accolades ; cela vous permettra d'éviter de nombreuses erreurs. Prenez ce réflexe à vos débuts et changez-en éventuellement après.

### L'instruction switch

Cette instruction permet de faire plusieurs tests sur la valeur d'une variable, ce qui évite de faire plusieurs `if` imbriqués et simplifie ainsi la lecture du code.

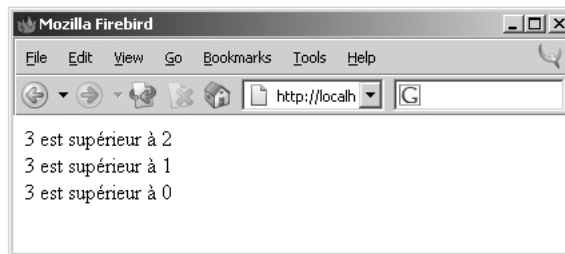
```
<?php
$nombre = mt_rand(0,4);
switch ($nombre)
{
case 4:
    echo "$nombre est supérieur à 3 <br>";
case 3:
    echo "$nombre est supérieur à 2 <br>";
case 2:
```

```
    echo "$nombre est supérieur à 1 <br>";  
case 1:  
    echo "$nombre est supérieur à 0 <br>";  
    break ;  
default:  
    echo "$nombre est 0 <br>";  
}  
?>
```

Les parenthèses qui suivent le mot-clé `switch()` indiquent une expression dont la valeur est testée successivement par chacun des `case`. Lorsque la valeur correspond à un `case`, la suite d'instructions est exécutée jusqu'à la fin du `switch` ou l'apparition d'un `break`. Si aucune correspondance n'est trouvée, alors le code est exécuté à partir du mot-clé `default`.

**Note**

Une fois la correspondance trouvée, toutes les instructions sont exécutées. Il faut bien remarquer que dans notre exemple, le nombre 3 afficherait trois lignes et non pas une seule. Le rendu de l'exemple est donné à la figure 4-6.

**Figure 4-6***Instruction switch*

Il est possible d'émuler le comportement d'une suite `if/elseif/else` en mettant un `break` pour chaque `case`. C'est toutefois une syntaxe déconseillée car sujette à erreur (il arrive fréquemment qu'on oublie le `break` ou qu'on ne le voit pas pendant la relecture). Pour des suites de conditions exclusives, préférez l'utilisation de conditions classiques, plus adaptées.

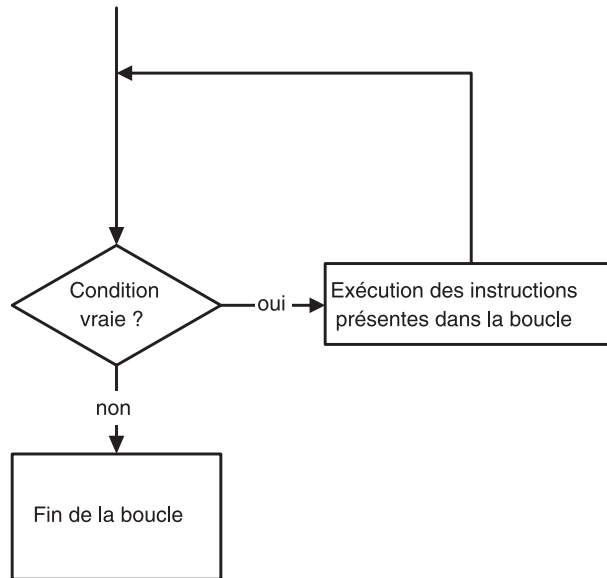
## Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois une même série d'instructions en fonction d'une (ou plusieurs) condition(s).

### L'instruction while

L'instruction `while(){}` correspond à « tant que ». Donc, on pourra exécuter des instructions tant qu'une condition sera remplie (voir figure 4-7).

Figure 4-7  
*L'instruction while*



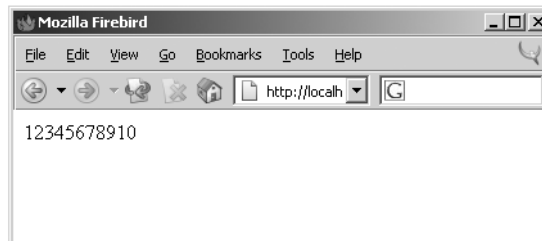
En PHP, cela s'écrit de la façon suivante :

```
while ( condition ){  
    instructions  
}
```

Le programme commence par tester si la condition est vraie. La boucle `while(){}` exécute alors le code du programme jusqu'à ce que la condition devienne fausse.

```
<?php  
$i = 1;  
while ( $i <= 10 )  
{  
    echo $i;  
    $i++;  
}  
?>
```

Figure 4-8  
*Exemple  
d'utilisation  
de while*



**Attention**

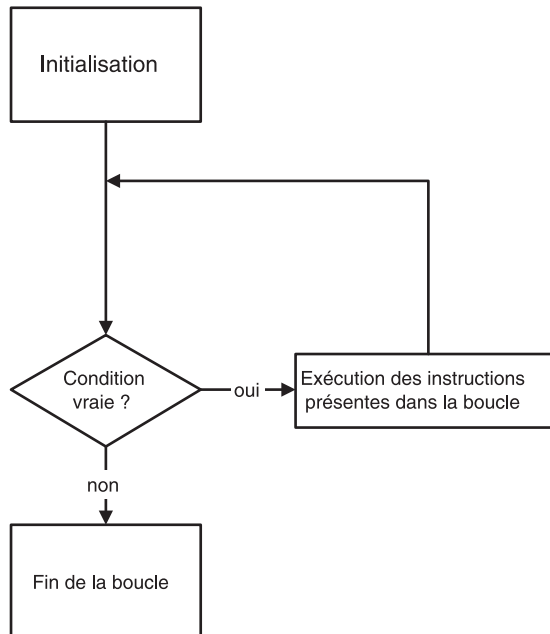
La condition n'est testée qu'après exécution complète du bloc d'instructions. Si la condition est évaluable à faux avant la fin, le reste des instructions s'exécutera tout de même.

L'instruction peut aussi être utilisée avec une syntaxe alternative `do {} while()`. Dans ce cas, le premier test n'est fait qu'après la première exécution de la boucle.

```
<?php
$i = 1;
do
{
    echo "$i";
    $i++;
}
while ($i <= 10);
?>
```

**L'instruction for**

**Figure 4-9**  
*L'instruction for*



La structure d'une boucle `for` est :

```
for (expression1 ; condition ; expression2) {
    Code à exécuter
}
```

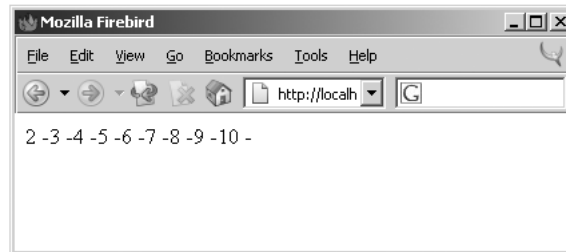
- `expression1` est exécutée une fois à l'entrée de la boucle pour l'initialiser.
- La `condition` est testée à chaque fois qu'on se propose de repasser dans la boucle, y compris la première fois. En général, cela permet de tester un compteur.
- `expression2` est exécutée à la fin d'un passage dans la boucle. En général, on incrémente une variable qui est utilisée dans le test de la condition.

Cette instruction est souvent utilisée pour des boucles de longueur déterminée puisque son utilisation s'avère assez simple. Le résultat du code suivant est donné à la figure 4-10.

```
<?php
for ( $i = 2; $i <= 10 ; $i++ ) {
    echo "$i -";
}
?>
```

**Figure 4-10**

*Exemple  
d'utilisation  
de la boucle for*



## L'instruction foreach

PHP inclut une commande `foreach()`, comme en Perl. C'est un moyen simple de parcourir un à un les éléments d'un tableau.

Il y a deux syntaxes possibles. La seconde est une extension mineure mais pratique de la première :

```
foreach ($array as $element) instruction;
foreach ($array as $key=>$element) instruction;
```

La première syntaxe passe en revue le tableau `$array`. À chaque itération, la valeur de l'élément courant est assignée à `$element` et le pointeur interne de tableau est avancé d'un élément (ce qui fait qu'à la prochaine itération, on accédera à l'élément suivant).

La deuxième forme fait exactement la même chose, mais c'est la clé de l'élément courant qui est assignée à la variable `$key`.

### Note

L'instruction `foreach()` travaille sur une copie du tableau spécifié, et non sur le tableau lui-même. Par conséquent, les modifications ne seront pas prises en compte pour l'exécution de la boucle elle-même.

La figure 4-11 présente le résultat de l'exécution de l'exemple suivant :

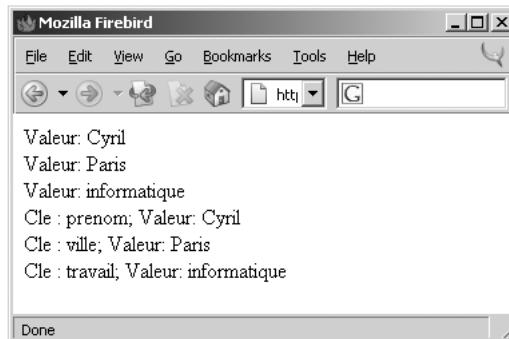
```
<?php
$tab = array(
    'prenom' => 'Cyril' ,
    'ville' => 'Paris' ,
    'travail' => 'informatique'
) ;

foreach ($tab as $element) {
    echo "Valeur: $element<br>\n";
}

foreach ($tab as $cle => $valeur) {
    echo "Cle : $cle; Valeur: $valeur<br>\n";
}
?>
```

**Figure 4-11**

*Exemple  
d'utilisation de la  
boucle foreach*



### Utilisation par références

Il est possible, depuis PHP 5, d'utiliser les valeurs par référence et non par copie. Si vous modifiez une valeur, elle sera alors modifiée dans le tableau d'origine.

```
<?php
$tab = array(1, 2, 3) ;
foreach($tab as $valeur) {
    $valeur = $valeur - 1 ;
}
echo $tab[0] + $tab[1] + $tab[2] ; // Affiche 6
foreach($tab as &$valeur) {
    $valeur = $valeur - 1 ;
}
echo $tab[0] + $tab[1] + $tab[2] ; // Affiche 3
?>
```



## Les instructions d'arrêt

### Break

L'instruction `break` permet de sortir d'une structure conditionnelle telle que `for`, `while`, `foreach` ou `switch`.

```
while(TRUE) {
    echo 'affiché une fois' ;
    break ;
    echo 'jamais affiché' ;
}
```

`break` accepte un argument numérique optionnel qui vous indiquera combien de structures emboîtées ont été interrompues.

```
<?php
for( $i=1 ; $i<=3 ; $i++) {
    while(TRUE) {
        echo 'texte affiché une fois - ' ;
        while(TRUE) {
            echo 'texte aussi affiché une seule fois - ' ;
            break(2) ;
            echo 'texte jamais affiché - ' ;
        }
        echo 'texte jamais affiché - ' ;
    }
    echo 'texte affiché une fois - ' ;
    break ;
    echo 'texte jamais affiché - ' ;
}
?>
```

### Continue

L'instruction `continue` est utilisée dans une boucle afin d'éviter les instructions de l'itération courante et donc pour passer directement à la suivante.

`continue` accepte également un argument numérique optionnel qui vous indiquera combien de structures emboîtées ont été ignorées.

```
<?php
for( $i=1 ; $i<=3 ; $i++) {
    // N'affiche le texte qu'une seule fois quand $i==3
    if ( $i !=3 ) continue ;
    echo 'On en est à la troisième itération' ;
}
?>
```

## Les fonctions utilisateurs

Il existe deux types de fonctions en PHP : les fonctions dites natives, que vous pouvez employer sans faire appel à des bibliothèques (`phpinfo()`, `echo`, etc.) et les fonctions dites utilisateurs, qui sont déclarées par vous-même ou qui sont définies dans une bibliothèque.

Une fonction utilisateur peut être résumée à un sous-programme que l'on appelle depuis le programme principal.

Contrairement à une instruction simple, une fonction est un ensemble d'instructions qui peuvent être parfois très complexes. On regroupe donc toutes ces instructions en une fonction que l'on pourrait donc assimiler à un sous-programme. Ce groupe d'instructions est lancé à chaque appel de la fonction par le programme principal. Il sera par la suite possible d'exécuter ce bloc en une commande (via le nom de la fonction), au lieu de la recopier plusieurs fois dans le code.

Une fonction peut aussi renvoyer une valeur et prendre des paramètres, un peu comme une opération classique telle que la multiplication.

### Déclaration d'une fonction

La définition d'une fonction s'appelle déclaration et peut se faire n'importe où dans le code grâce au mot-clé `function`.

```
<?php
function Nom_De_La_Fonction($argument1, $argument2, ...) {
    // Liste d'instructions ;
}
?>
```

Les arguments sont les paramètres que l'on passe à la fonction. Il peut y en avoir un, plusieurs, ou même aucun (dans ce cas, on laisse les parenthèses vides). Les arguments peuvent être de simples variables, mais aussi des tableaux ou des objets.

### Valeur par défaut

Il est possible de donner une valeur par défaut à ces arguments. Pour cela, il suffit de faire comme une affectation dans la déclaration :

```
<?php
function Nom_De_La_Fonction($argument1 = 'valeur_par_defaut') {
    // Liste d'instructions ;
}
?>
```

### Valeur de retour

La fonction peut renvoyer une valeur grâce au mot-clé `return`. Lorsque l'instruction `return` est rencontrée, la valeur de retour est envoyée au programme appelant et l'exécution de la fonction est stoppée.

Une fonction peut contenir plusieurs instructions de retour, mais l'exécution de la fonction s'arrêtera à la première mise en œuvre.

Dans notre exemple, nous allons créer une fonction qui affiche du texte en fonction de deux paramètres. Le premier est le nom de la personne (`$qui`), le second, optionnel, permet de définir un texte qui sera affiché avant le nom de la personne. Dans le cas où le premier paramètre est vide, la fonction renvoie `FALSE` pour indiquer une erreur.

```
<?php
function dire_texte($qui, $texte = 'Bonjour') {
    if(empty($qui)){
        // Si $qui est vide, on retourne faux
        return FALSE;
    }else{
        echo "$texte $qui";
        // On affiche le texte
        return TRUE;
        // Fonction exécutée avec succès
    }
}
?>
```

## Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en lui passant les paramètres nécessaires.

```
■ Nom_De_La_Fonction(argument1, argument2, ...);
```

Certains arguments peuvent être optionnels, lorsqu'une valeur par défaut leur a été donnée. Dans l'exemple précédent, `$qui` est obligatoire, alors que `$texte` est optionnel.

```
<?php
// Déclaration de la fonction dire_texte()
// Passage des deux paramètres
dire_texte('cher phpeur', 'Bienvenue');
// Affiche "Bienvenue cher phpeur"

// Utilisation de la valeur par défaut du deuxième paramètre
dire_texte('cher phpeur');
// Affiche "Bonjour cher phpeur"
?>
```

On utilise ici la fonction créée précédemment et on y fait appel en testant la valeur de retour.

```
<?php
// Déclaration de la fonction dire_texte()
// Utilisation de la valeur de retour
if(!dire_texte("")){
    // Provoque l'affichage ci-dessous
    echo "Erreur";
}
```

```
// Puisque la chaîne passée est vide la fonction retourne false
}
if(!dire_texte("cher phpeur")){
    // Affiche "Bonjour cher phpeur"
    //"Erreur" ne s'affiche pas
}
?>
```

## Visibilité des variables

Les variables n'ont pas toutes la même visibilité dans un script. Par exemple, les variables déclarées dans une fonction ne seront utilisables que dans celle-ci. Inutile donc (par défaut) d'essayer de les appeler en dehors. De la même façon, les variables déclarées dans votre script ne seront pas accessibles dans une fonction. Les deux espaces sont complètement indépendants.

```
<?php
$params = 3 ;

function decremente($valeur) {
    $valeur = $valeur -1 ;
    echo $params ; // N'affiche rien, $params n'étant pas défini
    // dans le contexte de la fonction
}
decremente( $params ) ;
echo $params ; // Affiche 3, la variable modifiée et la valeur
// actuelle ne sont pas les mêmes
echo $valeur ; // N'affiche rien, $valeur n'étant pas défini
// dans le contexte global
?>
```

## Portée des variables

Les variables extérieures à une fonction ne sont pas disponibles dans cette fonction et vice versa (les variables utilisées dans une fonction ne sont pas répercutées à l'extérieur de celle-ci).

Une variable a donc une portée plus ou moins grande selon l'endroit où elle est définie.

Il existe plusieurs niveaux de définition de variable :

- Le niveau `global` permet à une variable d'être visible dans la fonction et à l'extérieur de la fonction.
- Le niveau `static` permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script. Cette variable conservera ses différentes valeurs à chaque nouvel appel de la fonction.
- Le niveau `local`, utilisé par défaut, permet de définir une variable qui ne sera visible que dans la fonction en cours.

Dans l'exemple suivant nous allons voir que l'utilisation d'une variable statique permet de disposer dans une fonction d'une variable locale persistant durant toute l'exécution du script.

Ainsi, nous allons ajouter des camions puis les afficher.

```
<?php
$chaine = 'Nombre de camions : ';
function ajoute_camion($mode='') {
    global $chaine;
    static $nb=0;
    $nb++;
    // On incrémente le nombre de camions
    if($mode == 'affiche') {
        echo $chaine.$nb;
        // On affiche le nombre de camions
    }
}

ajoute_camion(); // nb == 1
ajoute_camion(); // nb == 2
ajoute_camion(); // nb == 3
ajoute_camion('affiche');
// Affiche Nombre de camions : 4
?>
```

Nous verrons au chapitre 8 qu'il est possible d'utiliser des globales pour avoir accès à diverses variables.

### Passage par copie ou référence

Par défaut, PHP utilise les paramètres avec un passage dit « par copie ». La valeur utilisée par la fonction n'est donc pas celle donnée en argument mais une copie. Si vous la modifiez à l'intérieur de la fonction, cela n'aura pas d'influence en dehors.

```
<?php
function decremente($valeur) {
    $valeur = $valeur -1 ;
}
$params = 3 ;
decremente( $params );
echo $params ; // Affiche 3
?>
```

#### Note

La seule exception à cette règle concerne les objets, qui sont toujours passés par référence depuis PHP 5. Vous trouverez plus de détails à ce sujet dans le chapitre sur la programmation orientée objet.

Il est toutefois possible de déclarer un paramètre comme devant être passé par référence et non par copie. PHP utilise alors la valeur d'origine et non pas une copie. On montre

qu'on souhaite un passage par référence en préfixant le paramètre d'un & dans la définition de la fonction.

```
<?php
function decremente(&$valeur) {
    $valeur = $valeur -1 ;
}
$params = 3 ;
decremente( $params ) ;
echo $params ; // Affiche 2
?>
```

#### Note

Avec PHP 4, il était impossible de définir une valeur par défaut pour un paramètre passé par référence. C'est enfin chose possible avec la version 5.

### Valeur de retour par référence

De même que les paramètres, les valeurs de retour sont aussi envoyées par copie (sauf pour les objets). En général, ce comportement ne change rien puisque la variable renvoyée est une variable locale. Si toutefois vous utilisiez une variable globale et que vous souhaitiez pouvoir l'utiliser par référence par la suite, il vous faudrait ajouter un & devant le nom de la fonction dans la déclaration. Il est important que les fonctions procédant ainsi soient peu nombreuses et bien connues, car en oublier une peut amener des erreurs (on croit utiliser une variable isolée mais elle va en réalité modifier d'autres parties du script).

```
<?php
function &decremente(&$valeur) {
    $valeur = $valeur -1 ;
    return $valeur ;
}
$params = 3 ;
$nouveau =& decremente( $params ) ;
$nouveau = $nouveau -1 ;
echo $params ; // Affiche 1
?>
```

On peut remarquer l'utilisation d'un deuxième & lors de l'affectation. Il serait en effet inutile de retourner une référence pour en copier simplement la valeur lors de l'affectation. Ici, on retourne une référence et on la lie avec un nouveau nom grâce à =&.

### Retourner plusieurs valeurs

Lorsque vous souhaitez qu'une fonction retourne plusieurs valeurs, le plus simple est d'utiliser un tableau.

```
<?php
function nom_fonction() {
```

```
.....
return array( 'contenu1', $variable2, $variable3 );
// On retourne les valeurs voulues dans un tableau
}
$retour = nom_fonction();
echo "$retour[0] - $retour[1] - $retour[2]";
?>
```

Le mot-clé `list()` permet d'affecter en une opération plusieurs variables venant d'un tableau. Il est particulièrement utile pour récupérer plusieurs valeurs de retour d'une fonction.

```
<?php
function traduction() {
    $tab[] = 'nom' ;
    $tab[] = 'prenom' ;
    $tab[] = 'telephone' ;
    return $tab;
}
list($nom, $prenom, $telephone) = traduction() ;
echo "$nom $prenom, $telephone" ;
// Affiche nom prenom, telephone
?>
```

## ***Nombre de paramètres indéfini***

Vous remarquerez au fur et à mesure de votre utilisation que certaines fonctions de PHP acceptent un nombre indéfini d'arguments. Avoir ce comportement avec des valeurs par défaut nécessiterait de définir énormément de valeurs par défaut et de gérer autant de conditions dans le code. Même ainsi, le fonctionnement ne serait pas parfait, car vous ne pourriez que définir un grand nombre de paramètres, pas un nombre indéfini.

Il est possible de travailler avec un nombre réellement indéfini d'arguments à l'aide des fonctions `func_num_args()`, `func_get_arg()` et `func_get_args()`. La première des trois permet de connaître le nombre d'arguments utilisés pour appeler votre fonction, la deuxième permet de récupérer un argument à partir de sa position, et la troisième vous retourne un tableau contenant les différents arguments utilisés à l'appel de votre fonction.

Il vous suffit alors d'utiliser une ou plusieurs de ces trois fonctions dans le corps de la vôtre et de ne spécifier aucun paramètre dans la déclaration.

```
<?php
function indefinie() {
    echo 'Il y a eu ', func_num_args(), ' arguments : ' ;
    $tab = func_get_args();
    echo implode(', ', $tab) ;
}

indefinie(0, 1, 2, 3, 4, 5, 6) ;
// Affiche : Il y a eu 7 arguments : 0, 1, 2, 3, 4, 5, 6
?>
```

## Inclure des bibliothèques ou des fichiers

PHP permet l'utilisation de deux fonctions très simples et cependant très utiles qui permettent la réutilisation de code. En utilisant les fonctions `require()` ou `include()`, il est possible de charger des fichiers dans un script PHP. En simplifiant, on peut considérer qu'à l'endroit où vous incluez une bibliothèque, PHP recopiera celle-ci. Cela permet de gagner du temps et de centraliser les opérations récurrentes.

Pour faire une comparaison avec le langage C, on pourrait assimiler ces fonctions à `#include`.

Cela permet donc de créer des fichiers communs à tous les scripts. Il n'existe pas d'obligation dans la dénomination de vos fichiers de bibliothèques, cependant on utilise généralement l'extension `.inc.php`.

Par exemple, voici un fichier `a_inclure.inc.php` :

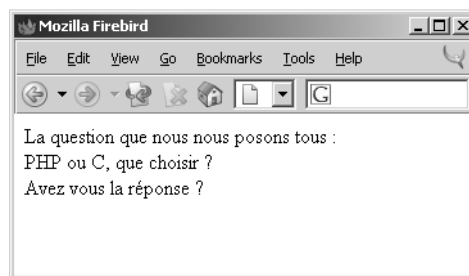
```
<?php
echo 'PHP ou C, que choisir ?<br>';
?>
```

Et voici un fichier principal utilisant le précédent :

```
<?php
echo 'La question que nous nous posons tous :<br>';
include ('a_inclure.inc.php');
echo 'Avez vous la réponse ?<br>';
?>
```

Figure 4-12

*Inclusion de fichiers*



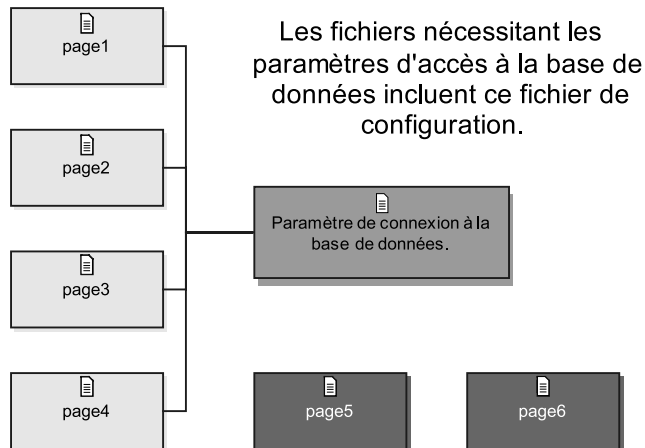
Au chargement du fichier principal, vous remarquerez que le fichier `a_inclure.inc.php` a été interprété (voir figure 4-12). Du coup, l'exemple est équivalent à ceci :

```
<?php
echo 'La question que nous nous posons tous :<br>';
echo 'PHP ou C, que choisir ?<br>';
echo 'Avez vous la réponse ?<br>';
?>
```



Les inclusions peuvent par exemple servir à la gestion des paramètres de connexion à une base de données (voir figure 4-13).

**Figure 4-13**  
*Inclusion de fichiers*



Une autre utilisation des fonctions `include()` et `require()` est de moduler une page d'un site. Il est fréquent de créer des pages d'aspect semblable en insérant un même fichier haut de page (*header*) et un même fichier bas de page (*footer*). Vous trouverez plus d'informations à ce sujet au chapitre 23 concernant les gabarits.

### ***Différence entre `require()` et `include()`***

On inclut donc un fichier en utilisant soit la fonction `include()`, soit la fonction `require()`. Pourquoi deux fonctions ?

Il existe une différence importante entre les deux. Avec `include()`, un fichier est inclus dynamiquement, lors de l'exécution du code. L'instruction est réévaluée à chaque passage et ne provoque qu'un *warning* en cas d'erreur. L'instruction `require()` ne réévalue pas le contenu sur un second passage (par exemple un `require $var` dans une boucle inclura toujours la même chose, même si `$var` change entre-temps), et provoque une erreur en cas d'échec.

### ***`require_once()` et `include_once()`***

`require_once()` et `include_once()` ont la même fonction que `require()` et `include()`. La seule différence est qu'elles s'assurent que le fichier que l'on essaie d'inclure ne l'a pas déjà été. Cela se révèle pratique lors de l'utilisation de fonctions ou de classes pour éviter les redéclarations qui engendrent des erreurs.

# 5

## Traitements de chaînes

---

Les données manipulées dans un contexte Web sont pour l'essentiel des données textuelles. Il est donc important de connaître les différentes fonctions de traitements de chaînes pour éviter de réinventer la roue.

PHP offre un large éventail de fonctions spécialisées pour vous éviter d'avoir à utiliser des procédures bas niveau comme vous pourriez le faire en langage C. Vous n'aurez par exemple presque jamais besoin de parcourir une chaîne caractère par caractère.

Parmi les applications simples, on peut citer la transformation de majuscules en minuscules, la troncature à une certaine taille, le remplacement des accents par des lettres non accentuées, etc. Pourtant, si ces fonctions peuvent servir à mettre en forme des données, leur cadre est beaucoup plus large. Vérifier la forme des chaînes de caractères est par exemple une nécessité. Il peut être également utile de changer le format d'une chaîne avant insertion dans une base de données ou sur un système avec un jeu de caractères différent. Nous allons au cours de ce chapitre vous proposer un panorama des fonctionnalités de ce domaine.

Avant d'aborder la suite, nous vous conseillons d'avoir lu la partie sur les chaînes de caractères du chapitre 3, afin de vous remémorer les procédures de base du type `string`.

### Fonctions d'affichage

#### *Affichages simples*

Les mots-clés `echo()` et `print()` permettent d'afficher tous types de valeurs. Les parenthèses autour de la valeur à afficher sont optionnelles.

```
echo( 'Eric Daspet' ) ;  
echo 'Cyril Pierre de Geyer' ;
```

L'instruction `echo` permet aussi, quand les parenthèses ne sont pas utilisées, de renvoyer plusieurs valeurs à la suite, en les séparant par des virgules. On évite ainsi d'avoir à faire une concaténation entre plusieurs chaînes avant affichage.

```
echo 'PHP ', 5, ' avancé', $auteurs ;
```

## Affichages avec masques

Plutôt que de faire de multiples concaténations, vous pouvez définir une structure générique et demander à PHP de remplacer certaines composantes par des paramètres. On parle alors d'affichage paramétré. Dans PHP, les paramètres à remplacer sont des caractères précédés du symbole de pourcentage. On peut définir plusieurs types de paramètres : entiers, chaînes, nombres décimaux, etc.

Ainsi, dans notre exemple, PHP affichera « PHP 5 est disponible sur <http://fr.php.net/> » :

```
<?php
$version = 5 ;
$miroir = 'fr' ;
$masque = 'PHP %d est disponible sur http://%s.php.net' ;
printf($masque, $version, $miroir) ;
?>
```

## Des masques pour gérer plusieurs langues

Ce type de syntaxe est très pratique quand les phrases à afficher sont définies ailleurs que les données. Une des applications courantes est l'utilisation de fichiers de traduction : tous les masques sont alors définis dans un fichier de configuration et dépendent de la langue. Leur utilisation ne dépend alors pas de la mise en forme associée à la langue en question.

```
<?php
$langue = $_COOKIE['langue'];
$fr = '%d est la version de %s' ;
$en = '%d is %s\'s version' ;
if ($langue === 'en')
{
    printf($en, 5, 'PHP') ;
}
else{
    printf($fr, 5, 'PHP') ;
}
?>
```

## Syntaxe des masques de printf et assimilés

Les masques sont symbolisés par le signe de pourcentage `%` suivi de leur type. Le premier masque sera remplacé par le premier paramètre, et ainsi de suite. À la suite du `%` doit suivre une lettre désignant le type de donnée à afficher :

- `%s` correspond à une chaîne de caractères.

- `%d` et `%u` symbolisent des nombres entiers, le premier signé, le deuxième non signé.
- `%f` affiche un nombre à virgule flottante.
- `%b`, `%o`, `%x` représentent des entiers, mais à afficher respectivement en notation binaire, octale et hexadécimale. La notation hexadécimale utilise des minuscules pour les caractères alphabétiques, utilisez `%X` pour avoir des majuscules.

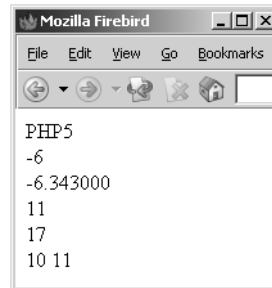
**Note**

La suite `%%` permet d'afficher un signe `%` sans l'interpréter comme un paramètre.

Le résultat de l'exemple suivant est illustré à la figure 5-1.

```
<?php
printf('%s <br>', 'PHP5') ; // Affiche PHP5
printf('%d <br>', -6) ; // Affiche -6
printf('%f <br>', -6.343) ; // Affiche -6.343
printf('%b <br>', 3) ; // Affiche 11 car 1*2 + 1*1
printf('%o <br>', 15) ; // Affiche 17 car 1*8 + 7*1
printf('%x %X <br>', 16, 17) ; // Affiche 10 11
?>
```

**Figure 5-1**  
*Utilisation  
de printf()*



### Taille minimale d'une donnée

Une fonctionnalité utile vous permet de définir la taille minimale que doit prendre une donnée. Cela vous permet notamment d'être sûr de la taille que fera votre chaîne pour le rendu visuel de vos applications. Pour ce faire, il faut compléter le masque par des 0 ou des espaces (valeur par défaut), à droite ou à gauche (par défaut).

Ce code est à insérer entre le type de données et le `%`. Ainsi, `%03d` affichera un entier et le préfixera par des zéros de façon à avoir un minimum de trois chiffres. Il est aussi possible de compléter la chaîne par des lettres ; on les préfixe alors avec une apostrophe (`'`).

Pour compléter à droite, il faut insérer le symbole `-`.

```
<?php
printf('%03d', 1) ; // Affiche 001
printf('%-4s B', 'A') ; // Affiche A B (4 espaces)
?>
```

Pour les nombres à virgule flottante, il est possible de définir la précision. Ainsi, 08.4 permet de forcer 8 digits au total dont 4 décimales à droite de la virgule.

```
<?php
printf('%08.4f', 1.0) ; // Affiche 001.0000
?>
```

### Fonctions assimilées à printf

La fonction `printf()` renvoie la chaîne résultante sur l'affichage. Une seconde fonction `sprintf()` renvoie la chaîne en retour de fonction.

```
<?php
printf('%s', 'bonjour') ; // Affiche "bonjour" directement

$texte = sprintf('%s', 'bonjour') ; // Met le texte en variable
echo $texte ; // Affiche "bonjour" lors de l'appel à echo
?>
```

Si vous devez passer un nombre important de paramètres à la fonction `printf()`, vous pouvez utiliser un tableau les contenant tous. Dans ce cas, utilisez les fonctions `vprintf()` et `vsprintf()`, qui sont similaires mais prennent les différents paramètres de remplacement dans un tableau unique au lieu de plusieurs arguments séparés.

```
<?php
$data = array( 'PHP', 5 ) ;
vprintf('%s %d', $data) ; // Affiche "PHP 5" directement

$texte = vsprintf('%s %d', $data) ; // Met le texte en variable
echo $texte ; // Affiche "PHP 5" lors de l'appel à echo
?>
```

### Scanner une chaîne de caractères

Deux fonctions permettent de faire l'opération inverse et de parcourir une chaîne pour en récupérer les valeurs selon un masque défini. S'il y a plusieurs valeurs à récupérer, elles sont retournées dans un tableau.

`sscanf()` permet de récupérer les variables à partir d'une chaîne de caractères et `fscanf()` permet de parcourir un fichier. Le premier paramètre est la chaîne à parcourir ou le pointeur de fichier, le second est le masque. Les deux renvoient la valeur `FALSE` si la chaîne en entrée ne correspond pas au masque.

```
<?php
$texte = '3 arrondi de 3.14' ;
list($entier, $chaîne, $flottant) = sscanf($texte, '%d %s de %f') ;
echo $entier, '<br>';
echo $chaîne, '<br>';
echo $flottant, '<br>';
?>
```

## Informations sur une chaîne

### Accéder à un caractère précis

Pour accéder directement à un caractère en connaissant sa position, il vous suffit d'ajouter sa position entre accolades à la fin de la variable. Le premier caractère est à l'index 0 et non à l'index 1.

```
<?php
$texte = 'PHP 5 avancé' ;
echo $texte[1] ; // Affiche H
?>
```

### Valeur ASCII d'un caractère

La valeur ASCII d'un caractère est donnée par la fonction `ord()` (pour *ordinal value* en anglais). La fonction `chr()` fait l'opération inverse et renvoie un caractère à partir de son code ASCII.

```
<?php
echo ord('a') ; // Renvoie 97
echo chr(97) ; // Renvoie a
?>
```

On peut se servir des caractères ASCII et de la conversion pour afficher l'ensemble de l'alphabet. La lettre a minuscule correspond au code ASCII 97 et z correspond à 122. Il suffit donc de faire une boucle et de convertir le code ASCII en caractères.

```
<?php
// Script permettant d'afficher l'alphabet

for ($i=97 ; $i <= 122 ; $i++){
    $chaîne .= chr($i);
}
echo $chaîne;

?>
```

On peut faire la même chose en majuscules en parcourant les correspondances des codes ASCII allant de 65 à 90.

### Taille d'une chaîne

La taille d'une chaîne de caractères (le nombre de signes, espaces et caractères blancs compris) est donnée par la fonction `strlen()`.

```
<?php
$livre = 'PHP 5 avancé' ;
echo strlen($livre) ; // Affiche 12
?>
```

**Important**

Si vous utilisez Unicode, certains caractères peuvent utiliser plus d'un octet sur le disque ou en mémoire. Dans ce cas, c'est la taille en octets et non en caractères qui est retournée par `strlen`. La chaîne « Éric Daspet » retournera donc une taille de 12 et non de 11 si vous utilisez un codage UTF-8 (le E accentué prend deux octets). Une description plus complète de la problématique est abordée plus loin dans ce chapitre.

Il vous est aussi possible de trouver le nombre de mots d'une chaîne grâce à la fonction `str_word_count()`.

```
<?php
$livre = 'PHP 5 avancé' ;
echo str_word_count($livre) ; // Affiche 3
?>
```

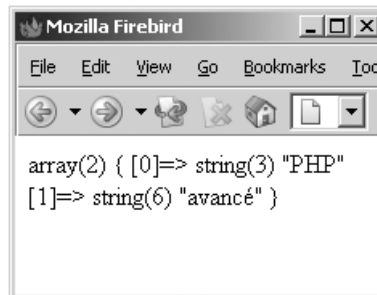
**Lister les mots d'une chaîne**

Si vous fournissez un deuxième argument à la fonction `str_word_count()`, elle vous renverra la liste de tous les mots dans un tableau. Si la valeur de l'argument est 1, l'index sera l'ordre des mots, à partir de 0 (voir exemple suivant et figure 5-2).

```
<?php
$livre = 'PHP 5 avancé' ;
$tab = str_word_count($livre, 1) ;
var_dump($tab) ;
?>
```

**Figure 5-2**

*Lister les mots d'une chaîne*



Si la valeur de l'argument est 2, l'index sera la position du premier caractère du mot dans la chaîne.

```
<?php
$livre = 'PHP 5 avancé' ;
$tab = str_word_count($livre, 2) ;
var_dump($tab) ;
/* Affiche :
array(
```

```
0 => 'PHP' ,
4 => 5 ,
6 => 'avancé'
)
*/
?>
```

## Position d'une sous-chaîne

La fonction `strpos()` permet de connaître la position d'une chaîne dans une autre. La valeur `FALSE` est renvoyée si la chaîne n'est pas trouvée.

```
strpos(chaîne, sous_chaîne_a_rechercher)
```

### Attention

Il ne faut pas confondre l'index 0 avec la valeur `FALSE`. Pour faire la distinction, vous pouvez utiliser l'opérateur `===` qui fait une comparaison de type en plus d'une comparaison de valeur.

Le code exemple suivant est illustré à la figure 5-3.

```
<?php
$pos = strpos('eric.daspet@dreams4net.com', '@') ;
if($pos === FALSE) {
    echo 'Le caractère @ n'est pas présent';
} else {
    echo "Le caractère @ est présent à la position $pos";
}
?>
```

**Figure 5-3**

*Connaître la position d'un caractère*



Un troisième argument est disponible, définissant la position du caractère à partir duquel commencer la recherche. Dans la pratique, cela permet d'ignorer les caractères déjà analysés.

La fonction `stripos()` est identique mais permet de ne pas tenir compte de la casse lors de la recherche. La fonction `strrpos()` permet, elle, de faire la recherche de droite à gauche au lieu de gauche à droite. Il existe une fonction `strripos()`, cumulant les deux.



## Présence de certains caractères

Vous pouvez avoir besoin de vérifier la présence de caractères ou de sous-chaînes dans un texte. Généralement, cela sert pour vérifier des données transmises par un utilisateur.

La fonction `strspn()` retourne la longueur de la première sous-chaîne contenant uniquement les caractères spécifiés. La fonction `strcspn()` fait l'opération inverse et retourne la longueur de la première sous-chaîne ne contenant aucun des caractères spécifiés.

Ces fonctions sont utiles pour connaître la présence de caractères non prévus ou interdits dans une chaîne.

```
<?php
$chaîne = "chaîne à vérifier" ;
$masque = "'";
if( strcspn($chaîne, $masque) == strlen($chaîne) ) {
    echo 'il n y a pas d\'apostrophes' ;
} else {
    echo 'il y a des apostrophes' ;
}
?>
```

## Conversions et formatages

Nous verrons au fur et à mesure des chapitres que certains caractères ont des significations particulières selon leur contexte d'utilisation. C'est par exemple le cas lors d'un traitement dans une base de données où l'on doit protéger les apostrophes (qui autrement délimitent les chaînes). De telles préparations permettent d'éviter que certains caractères soient interprétés par le format destination et soient utilisés en tant que commandes ou délimitations au lieu d'être pris tels quels. L'oubli de telles conversions est la base de problèmes de sécurité comme l'injection SQL ou le *Cross Site Scripting*.

## Protections et échappements

### Protections classiques

Ce que nous appelons les protections classiques sont les protections de caractères à l'aide de la barre oblique inverse (caractère « \ »). C'est par exemple le type de protection utilisé par PHP pour les chaînes entre guillemets. L'utilisation de la barre oblique inverse est probablement la forme de formatage la plus répandue.

```
echo "Mes barres obliques inverses protègent mes \"guillemets\"";
```

#### Rappel

Sans le caractère de protection, l'interpréteur PHP aurait cru que la chaîne de caractères s'arrêtait au second guillemet rencontré. Il n'aurait alors pas compris la suite et aurait affiché une erreur.

La fonction `addslashes()` permet de protéger automatiquement les guillemets, apostrophes et barres obliques inverses en les préfixant automatiquement. Cette fonction est particulièrement utile dans la préparation de requêtes SQL pour que les apostrophes ne soient pas interprétées comme des délimiteurs de texte mais comme de simples caractères.

```
<?php
$nom = "PIERRE de GEYER d'ORTH";
$prenom = "Cyril";

$nom = addslashes($nom);
$prenom = addslashes($prenom);
$sql = "INSERT INTO user (prenom, nom) VALUES ('$prenom', $nom)";
// Sans l'appel à addslashes() l'apostrophe du nom
// aurait pu provoquer une erreur.
?>
```

La fonction `addcslashes()` est la même fonction, plus étendue. Elle convertit aussi les fins de ligne et les retours chariot (en `\n` et `\r`), ainsi que les caractères dont le code ASCII est inférieur à 32 ou supérieur à 126 (avec la syntaxe `\xx` où `xx` est le code ASCII du caractère). Le premier paramètre est la chaîne à convertir, le second paramètre contient la liste des caractères à échapper. Vous pouvez définir des suites de caractères en les séparant par deux points (exemple `\0..\32`).

```
<?php
$texte = "texte\n\r\"'texte" ;

// Affiche texte et "'texte, sur deux lignes
echo $texte ;

// Affiche texte et "\"'texte, sur deux lignes
echo addslashes($texte) ;

// Affiche texte\n\r\"'texte, sur une ligne
echo addcslashes($texte, "\"'\n\r") ;
?>
```

#### Attention

Faites attention si vous échappez les caractères `n`, `r`, `t` ou `0`, car `\n`, `\r`, `\t` et `\0` ont une signification spéciale dans beaucoup de langages. Ils risqueraient d'y être interprétés.

Les fonctions `stripslashes()` et `stripccslashes()` permettent de faire les opérations inverses et de récupérer une chaîne échappée pour la transformer en chaîne simple. Attention, dans cette conversion tous les codes commençant par une barre oblique inverse sont convertis en utilisant la convention du langage C : les codes `\n`, `\t`, `\r` et `\nnn` définissent les caractères de fin de ligne, de tabulation, de retour chariot, et le caractère de valeur ASCII `nnn`.

## Protection pour bases de données

Les protections à faire avant insertion dans une base de données dépendent de la base de données utilisée. Il suffit généralement d'opérer un échappement des apostrophes en \`'` ou `''` (deux apostrophes consécutives).

PDO vous propose la méthode `quote()` de l'objet de connexion pour effectuer cette opération. Elle prend en argument une chaîne de caractères et la formate de façon à pouvoir l'utiliser directement dans une requête SQL : des délimiteurs de chaîne sont insérés autour et les caractères spéciaux sont échappés.

```
<?php
$nom = "PIERRE de GEYER d'ORTH";
$nom = $dbh->quote($nom);

// Insertion d'un enregistrement
$sql = "INSERT INTO auteur (login, nom)
        VALUES ('Cyruss6', $nom)";
$dbh->exec($sql);
?>
```

Si vous n'utilisez pas PDO, vous pouvez employer les fonctions natives liées à votre SGBD. Par exemple, pour MySQL, on peut utiliser la fonction `mysqli_escape_string()`, alors que pour PostgreSQL, il s'agit de `pg_escape_string()` et `pg_escape_bytea()`.

```
<?php
$texte = "L'arrivée" ;
$protect = mysqli_escape_string($texte) ;
$sql = "SELECT * FROM table WHERE texte = '$protect' " ;
?>
```

## Protections pour HTML

### Conversion des entités

Lorsque vous envoyez des chaînes de caractères vers l'affichage dans le cadre d'une page web, ces caractères sont interprétés. Ainsi, `<br>` ne s'affichera pas mais provoquera un changement de ligne. Pour éviter que ces chaînes soient interprétées, il faut en convertir les caractères spéciaux (`<`, `>` et `&`) en entités (`&lt;`, `&gt;` et `&amp;`).

La fonction `htmlspecialchars()` permet d'effectuer cette conversion. Elle prend deux paramètres optionnels en plus de la chaîne à transformer.

Le premier paramètre permet de convertir aussi les guillemets et les apostrophes. La valeur par défaut (`ENT_COMPAT`) convertit les guillemets mais pas les apostrophes, la valeur `ENT_NOQUOTES` ne convertit aucun des deux, la valeur `ENT_QUOTES` convertit les deux. Convertir les guillemets (ou apostrophes) n'est obligatoire que pour une valeur délimitée elle-même par des guillemets (ou apostrophes).

Le deuxième paramètre définit le jeu de caractères à utiliser lors de la conversion. Le jeu par défaut est ISO-8859-1.

Le code suivant vous montre différents exemples, son résultat est illustré à la figure 5-4.

```
<?php
$texte = "valeur avec & <br> et avec \" et ' " ;

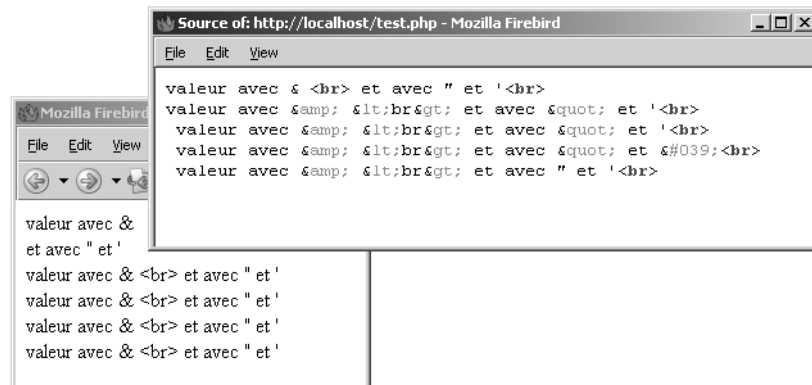
// Ne convertit rien, tout est interprété
echo $texte , "<br>\n" ;

// Convertit les caractères &, >, < et "
echo htmlspecialchars($texte) , "<br>\n " ;
echo htmlspecialchars ($texte, ENT_COMPAT) , "<br>\n " ;

// Convertit les caractères &, >, <, " et '
echo htmlspecialchars ($texte, ENT_QUOTES), "<br>\n " ;

// Convertit les caractères &, > et < uniquement
echo htmlspecialchars ($texte, ENT_NOQUOTES) , "<br>\n " ;
?>
```

**Figure 5-4**  
*Différentes conversions*



Il est aussi possible de convertir tous les caractères spéciaux sous forme d'entités (par exemple les caractères accentués) via la fonction `htmlentities()`. Son utilisation est identique à la précédente.

La fonction inverse est `html_entity_decode()`, qui prend aussi les mêmes paramètres.

### Conversion des changements de lignes

En HTML, le caractère de fin de ligne est un caractère dit caractère blanc, ayant la même signification qu'une espace. Un texte classique de plusieurs paragraphes sera rendu comme un seul paragraphe une fois inséré dans du HTML. Vous ne pourrez voir la différence qu'en affichant la source de la page. La balise `<br />` est le moyen, en XHTML, de forcer un passage à la ligne.

Pour permettre à un texte de s'afficher dans une page HTML avec des fins de ligne là où c'était prévu, il vous faudra probablement convertir votre texte en HTML. À défaut de

conversion complète utilisant les balises HTML de paragraphe, PHP vous offre la possibilité d'ajouter automatiquement un `<br />` avec chaque caractère de fin de ligne en utilisant la fonction `nl2br()`. Une fois le texte converti inséré dans du HTML, il donnera visuellement le rendu prévu.

```
<?php
$texte = "première ligne \n deuxième ligne" ;
echo nl2br($texte) ;
?>
```

Le code HTML obtenu sera :

```
première ligne <br />
deuxième ligne
```

Ce dernier texte s'affichera effectivement sur deux lignes dans le rendu du navigateur.

### Suppression des balises HTML

Une dernière fonction permet de protéger ses sorties HTML. Au lieu de convertir les balises HTML en entités, elle les supprime simplement. `strip_tags()` prend en paramètres la chaîne à transformer et, optionnellement, une liste de balises HTML à autoriser. Cette fonction est très pratique, car elle vous permet de limiter les caractères HTML dans une variable envoyée par un utilisateur. Sur un forum, par exemple, vous pourriez n'autoriser que le gras et l'italique (`<b>` et `<i>`).

#### Attention

Les balises autorisées ne sont pas modifiées et les attributs `onmouseover` ou autres attributs autorisant le JavaScript peuvent causer des problèmes de sécurité.

Si la chaîne HTML en entrée est mal formée ou contient des balises mal codées, `strip_tags()` renverra une erreur.

```
<?php
$texte = '<h1><a href="page.html">titre</a></h1>' ;
echo "Texte brut : $texte " ;
echo "Texte protégé : " ;
echo strip_tags( $texte, '<h1><em><strong>' ) ;
?>
```

## Conventions d'affichage locales

Certaines conventions d'affichage dépendent du pays. Ainsi, les Français utilisent la virgule pour délimiter la partie décimale d'un nombre alors que les Anglo-Saxons utilisent le point. PHP sait gérer ces localisations et dispose pour cela d'une fonction qui vous permet d'adapter vos textes.

La fonction `setlocale()` permet d'initialiser la localisation à utiliser par la suite. Elle prend deux paramètres :

```
setlocale (catégorie, localité)
```

Le premier est la catégorie d'application. La localisation peut être appliquée :

- aux comparaisons de chaînes de caractères, valeur `LC_COLLATE` applicable à la fonction `str_coll()` ;
- pour les tris et conversions, valeur `LC_CTYPE` utilisée par exemple pour la fonction `strtoupper()` ;
- pour les conventions monétaires, valeur `LC_MONETARY` ;
- pour les séparateurs dans les valeurs numériques, valeur `LC_NUMERIC` ;
- pour les dates et heures, valeur `LC_TIME` ;
- à toutes les catégories précédentes, avec la valeur `LC_ALL` (utilisée par défaut).

Le deuxième paramètre définit la localisation à utiliser (par exemple `fr_FR@euro`). Vous pouvez définir une suite de localisations, envoyée sous forme de tableau ou en ajoutant des paramètres à la fonction. Dans ce cas, la première localisation fonctionnelle sera utilisée. Une chaîne vide ne fera aucune action mais renverra la valeur utilisée actuellement ; une valeur numérique nulle utilisera la variable d'environnement `LANG`.

```
<?php
setlocale(LC_ALL, 'fr_FR@euro', 'fr_FR', 'FR', 'fr') ;
?>
```

## Jeux de caractères

Les fonctions standards de traitements de chaînes de PHP ne contiennent aucune fonction pour traiter les jeux de caractères. Il vous faudra vous reporter vers les modules `iconv` ou `mbstring`. Si vous n'utilisez que le français et l'anglais, le codage par défaut (ISO-8859-1) vous convient probablement et vous pouvez sauter cette section.

### Codages caractères

Un codage caractère est ce qui permet à l'ordinateur de représenter un caractère sous forme binaire. Historiquement, les caractères sont stockés sur 7 bits et ne comprennent que l'alphabet américain. Le nom de ce codage est US-ASCII.

Par la suite, les différents pays ont adopté des représentations sur 8 bits, de façon à doubler le nombre de caractères disponibles et pouvoir utiliser des caractères supplémentaires. Le jeu de caractères ouest européen, utilisé en France, est nommé ISO-8859-1. Il a les mêmes 128 premiers caractères que l'ASCII mais contient en plus sur les 128 suivants les caractères accentués latins utilisés en France ou en Italie. Un deuxième jeu est apparu récemment, gérant le caractère `_` (euro) et quelques autres : le jeu ISO-8859-15.

Le problème de ces multiples jeux de caractères est qu'un même texte peut être interprété de multiples façons selon le jeu utilisé pour la lecture. Pour pallier ce problème, il existe un codage global contenant tous les caractères de tous les alphabets utilisés. Deux jeux de caractères utilisent ce codage : UTF-8 et UTF-16 (le premier tend à s'imposer sur l'autre).

Dans votre environnement, vous n'aurez probablement à manipuler que trois formats : US-ASCII car il est le format par défaut du XML, ISO-8859-1 car il est le format par défaut du HTTP (donc des transferts via le Web) et UTF-8 qui est le jeu international (souvent utilisé avec XML).

## Le module iconv

Le module `iconv` sert uniquement à faire des conversions entre jeux de caractères. Vous pouvez le compiler sous Unix avec le paramètre `--with-iconv` lors de la procédure de configuration. Sous Windows, il vous faudra modifier le `php.ini` pour « décommenter » l'extension et copier le fichier `iconv.dll` de la distribution `php` vers votre répertoire `system32` (lui-même probablement situé à l'adresse `c:\windows`).

La fonction principale du module se nomme `iconv()`. Elle permet de convertir une chaîne d'un jeu à un autre. Le premier argument est le jeu de caractères de départ, le deuxième est le jeu de caractères destination, et le troisième argument est la chaîne à convertir.

```
■ $chaîne_utf = iconv('ISO-8859-1', 'UTF-8', $chaîne_iso) ;
```

La fonction renvoie la chaîne convertie au format demandé.

## Le module mbstring

`mbstring` est un module complet permettant de gérer entièrement les jeux de caractères. Initialement, il a été développé dans le but de gérer les langues asiatiques.

Certains jeux de caractères nécessitent un codage sur plusieurs octets pour les caractères spéciaux (comme les accents). Ce codage peut casser le fonctionnement des fonctions de chaînes classiques (par exemple tromper les fonctions qui comptent le nombre de caractères et qui pourraient compter double les lettres accentuées). Il s'agit alors de donner un remplacement pour toutes ces fonctions afin qu'elles tiennent compte du fait qu'un caractère peut comporter plusieurs octets.

### Installation

Le module `mbstring` doit absolument être compilé en même temps que PHP. Lors de l'étape de pré-configuration, vous aurez à activer les options `--enable-mbstring=LANG` (où `LANG` représente votre langue, `all` pour toutes les compiler) et `--enable-mbstring-regex` pour activer la reconnaissance des expressions régulières.

Ce module nécessite de plus quelques directives de configuration dans le `php.ini` :

- `mbstring.language` définit la langue à utiliser et configure le jeu de caractères en conséquence (défaut à `English`).
- `mbstring.encoding_translation`, `mbstring.http_input` et `mbstring.http_output` permettent de convertir automatiquement les entrées et sorties HTTP dans les jeux de caractères précisés. La première directive doit être à `On` pour activer la fonctionnalité. Pour les deux autres, il faut fournir un jeu de caractères à utiliser ou la valeur `pass` pour sauter la conversion. Pour les entrées, il est possible de définir une liste séparée par des virgules pour détecter automatiquement le jeu à utiliser. On peut également fournir la valeur `auto`.

- `mbstring.internal_encoding` définit le jeu de caractères utilisé en interne par PHP (ISO-8859-1 par défaut).
- `mbstring.func_overload` permet, s'il est activé, de remplacer le contenu des fonctions de traitement de chaîne classiques par celui des fonctions commençant par `mb_`, afin de ne rien changer au code.

Il est aussi fréquent, pour activer la conversion automatique de toutes les pages, d'utiliser les fonctions de gestion de tampon. Pour cela, il suffit d'activer la directive `output_buffering` et de mettre à `mb_output_handler` la directive `output_handler`. Vous trouverez plus de détails sur la gestion du tampon de sortie au chapitre 15.

```
output_buffering = 0n ;
output_handler = mb_output_handler ;
```

## Utilisation

Les possibilités d'utilisation de ce module sont nombreuses mais nous ne pouvons, faute de place, les décrire toutes dans ce livre.

Les fonctions sont pour la plupart simplement les fonctions de traitement de chaînes converties afin de prendre en compte les jeux de caractères différents de celui par défaut. Vous y trouverez aussi des expressions régulières POSIX adaptées (voir le chapitre 26 sur les expressions régulières) et des fonctions permettant de redéfinir tout ce qui a été configuré dans le `php.ini`.

Les fonctions de traitement de chaînes ont généralement un nom et un fonctionnement similaire à leur équivalent, et sont simplement préfixées par `mb_`. Les seules différences consistent parfois en un paramètre supplémentaire et optionnel définissant le jeu de caractères à utiliser. La directive `mbstring.func_overload` permettant d'utiliser les noms habituels et de ne pas se préoccuper des équivalences, nous ne les détaillerons donc pas ici.

Deux fonctions restent tout de même assez importantes pour en parler : `mb_convert_encoding()` et `mb_convert_variables()`, qui permettent de convertir textes ou variables d'un jeu de caractères à un autre.

La première prend en argument le texte à transformer, le jeu de caractères du texte et le jeu de caractères vers lequel faire la transformation, et retourne la chaîne transformée.

La seconde fonction permet de transformer en une fois plusieurs variables. Elle prend un nombre de paramètres variable : les deux premiers sont les jeux de caractères de départ et destination, les autres sont les variables à transformer. Les textes transformés seront positionnés dans ces mêmes variables et ne seront pas retournés par la valeur de retour de la fonction.

```
$texte_iso = 'voilà une chaîne accentuée' ;
$texte_utf = mb_convert_encoding($texte_iso, 'ISO-8859-1', 'UTF-8');
```



## Manipulations sur les chaînes

Les fonctions de manipulations sont celles qui permettent de faire des opérations sur une chaîne. Sauf exception signalée, la chaîne d'origine n'est pas modifiée, la chaîne résultante est toujours renvoyée dans la valeur de retour de la fonction.

### Recherche d'une sous-chaîne

La fonction `strstr()` fonctionne de manière similaire à `strpos()` mais retourne le reste de la chaîne à partir de la position repérée, au lieu de la position elle-même. Elle a ses équivalents `stristr()` pour l'analyse sans prise en compte de la casse, et `strrchr()` pour l'analyse de droite à gauche.

```
<?php
strstr('eric.daspet@dreams4net.com', '@') ;
// Renvoie @dreams4net.com
?>
```

### Récupérer une sous-chaîne

Il est possible de récupérer une sous-partie d'une chaîne de caractères si on en connaît la position. La fonction `substr()` prend en argument une chaîne de caractères référence, une position de départ et une longueur. Cette fonction renvoie les caractères à partir de la position initiale jusqu'à atteindre la longueur définie.

```
<?php
$texte = 'PHP 5 avancé' ;
echo substr($texte, 6, 2) ; // Renvoie av
?>
```

Si vous omettez le troisième argument, toute la chaîne à partir de la position de départ sera retournée.

### Remplacer un motif

La fonction `str_replace()` permet de remplacer un motif dans une chaîne. Le premier argument est la sous-chaîne à rechercher, le deuxième argument est la chaîne de remplacement, et le troisième est la chaîne de référence où faire le remplacement.

```
<?php
$texte = 'PHP 4 avancé' ;
$cherche = '4' ;
$remplace = '5' ;
echo str_replace($cherche, $remplace, $texte) ;
// Renvoie PHP 5 avancé
?>
```

Si vous fournissez un tableau pour la chaîne à rechercher, il sera interprété comme une liste de chaînes à remplacer. Si la chaîne de remplacement est aussi un tableau, alors

chaque élément du tableau de recherche sera remplacé par l'élément correspondant du tableau de remplacement.

```
<?php
$texte = 'PHP 4 débutant' ;
$cherche = array('4', 'débutant') ;
$remplace = array('5', 'avancé') ;
echo str_replace($cherche, $remplace, $texte) ;
// Renvoie PHP 5 avancé
?>
```

Si vous passez une variable en quatrième argument, elle sera prise par référence et contiendra le nombre de remplacements faits.

La fonction `str_ireplace()` fait la même opération, avec les mêmes paramètres, mais fera une recherche insensible à la casse des caractères.

Si vous ne connaissez pas la chaîne à remplacer mais sa position, vous pouvez utiliser la fonction `substr_replace()`. Elle prend en arguments la chaîne de référence, la chaîne de remplacement, et la position du premier caractère à remplacer.

```
<?php
$texte = 'PHP 4';
$position = strpos($texte, '4') ;
$remplace = '5' ;
echo substr_replace($texte, $remplace, $position) ;
// Renvoie PHP 5
?>
```

## Fonctions d'élagage

L'élagage est la procédure qui consiste à retirer les caractères blancs avant et après un texte, un peu comme les 0 avant un nombre.

La fonction `trim()` retire les caractères blancs avant et après la chaîne de référence.

```
<?php
$texte = ' PHP 5 avancé ' ;
echo strlen($texte) ; // Renvoie 22
$texte = trim($texte) ;
echo strlen($texte) ; // Renvoie 12, on a enlevé 10 espaces
?>
```

Vous pouvez spécifier la liste des caractères à supprimer dans le second paramètre (optionnel). Par défaut sont considérés comme caractères blancs :

- l'espace normal (caractère ASCII 32) ;
- la tabulation horizontale (ASCII 9) ;
- le caractère de fin de ligne (ASCII 10) ;
- le retour chariot (ASCII 13) ;

- le caractère nul (ASCII 0) ;
- et la tabulation verticale (ASCII 11).

**Note**

`rtrim()` ne retire que les espaces à droite, `ltrim()` ceux de gauche.

## Remplissage

L'opération inverse de l'élagage est le remplissage. La fonction `str_pad()` permet de compléter une chaîne jusqu'à une certaine longueur.

```
<?php
str_pad('', 10) ;
// Complète jusqu'à 10 caractères avec des espaces
?>
```

Par défaut, le remplissage se fait avec des espaces, vous pouvez définir un caractère différent ou même une suite de caractères, en les fournissant en troisième argument.

Le remplissage se fait naturellement sur la droite. Vous pouvez modifier ce comportement à l'aide du quatrième argument :

- `STR_PAD_RIGHT` : compléter à droite ;
- `STR_PAD_LEFT` : compléter à gauche ;
- `STR_PAD_BOTH` : compléter des deux côtés.

```
<?php
// Donne un X et 9 espaces à droite
str_pad('X', 10, STR_PAD_RIGHT) ;
// Donne un X et 9 espaces à gauche
str_pad('X', 10, STR_PAD_LEFT) ;
?>
```

## Changement de casse

Le changement de casse est la transformation de caractères minuscules en majuscules ou inversement. Ces fonctions sont en général appelées juste avant l'affichage pour normaliser la présentation de données. On peut aussi s'en servir avant d'insérer des informations dans une base de données pour normaliser les enregistrements.

**Remarque**

Les caractères considérés comme à convertir et les correspondances entre majuscules et minuscules dépendent de la localisation utilisée (voir plus haut dans ce chapitre).

Les fonctions `strtoupper()` et `strtolower()` convertissent respectivement les chaînes de caractères en majuscules et minuscules.

```
<?php
$texte = 'Avec des MAJSCULES et minuscules';

echo strtoupper($texte);
// Renvoie AVEC DES MAJUSCULES ET MINUSCULES

echo strtolower($texte);
// Renvoie avec des majuscules et minuscules
?>
```

Probablement plus adaptées aux besoins standards, les fonctions `ucfirst()` et `ucwords()` convertissent en majuscules respectivement le premier caractère de la chaîne et le premier caractère de chaque mot.

```
<?php
$texte = "Avec des majuscules et minuscules" ;

echo ucfirst($texte) ;
// Renvoie Avec des majuscules et minuscules

echo ucwords($texte) ;
// Renvoie Avec Des Majuscules Et Minuscules
?>
```

## Coupure de paragraphes

Lors d'un envoi en texte pur et non en HTML, par exemple pour envoyer un courrier électronique, il est courant de limiter la taille des lignes afin de faciliter la lecture. Par messagerie électronique par exemple, on a l'habitude de couper les paragraphes à une valeur entre 72 et 80 caractères. Pour des colonnes de texte, on se base le plus souvent entre 40 et 70 caractères par ligne, des paragraphes trop larges nécessitent plus d'attention.

Dans le monde Web, on a également souvent besoin de limiter la taille d'un texte et il est difficile de ne pas couper un mot en son milieu. Par exemple, pour mettre en avant le début d'une actualité mais en limitant le nombre de caractères à 50.

La fonction `wordwrap()` permet d'opérer cette coupure de manière automatisée, et en respectant l'intégrité des mots (pas de coupures au milieu d'un mot, il revient entièrement à la ligne).

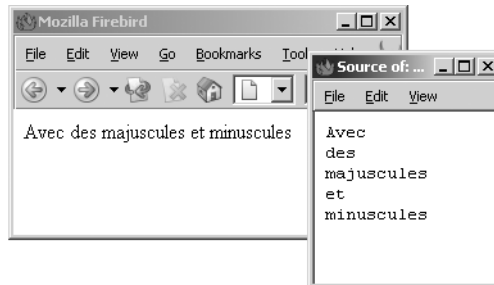
```
echo wordwrap($tres_long_texte) ;
```

Par défaut `wordwrap()` coupe les lignes à 75 caractères avec un caractère de fin de ligne. Vous pouvez changer ces comportements à l'aide d'arguments optionnels.

Si vous fournissez un deuxième argument à la fonction, il sera pris comme la longueur maximale du texte (voir code suivant et figure 5-5).

```
<?php
$texte = 'Avec des majuscules et minuscules' ;
echo wordwrap($texte,5) ;
?>
```

**Figure 5-5**  
*Coupure des paragraphes*



Le troisième argument sert à changer le caractère de coupure. Si par exemple vous désirez une sortie HTML, vous pouvez utiliser le code suivant :

```
echo wordwrap($tres_long_texte, 20, '<br>') ;  
// Utilise <br> comme séparateur
```

Le dernier paramètre (optionnel) permet de préciser le mode opératoire à suivre pour les mots plus longs que la largeur maximale. Si la valeur est vraie, ils seront coupés, sinon ils seront laissés tels quels.

```
echo wordwrap($tres_long_texte, 20, TRUE) ;  
// Les mots trop longs seront coupés
```

# 6

## Utilisation des tableaux

---

Les tableaux sont extrêmement utiles et souvent employés dans la majorité des applications. Nous avons vu au chapitre 3 la syntaxe de base pour les manipuler. Cependant, avec l'expérience et la pratique, on en vient à utiliser des fonctionnalités plus pointues pour lesquelles on a besoin d'effectuer des traitements spécifiques sur des tableaux. PHP offre de nombreuses fonctions pour vous faciliter la tâche. Nous allons donc ici voir comment gérer les différents types de tableaux, comment les manipuler, les ajouter, les parcourir, rechercher dedans, les épurer, les trier, etc.

### Taille d'un tableau

La fonction `count()` compte le nombre d'éléments d'un tableau.

```
count (var)
```

Le résultat ne dépend pas de l'indice maximal mais uniquement du nombre d'éléments.

```
<?php
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count($a);
// $result == 3

$b[0] = 7;
$b[5] = 9;
$b[10] = 11;
$result = count($b);
// $result == 3;
?>
```

**Note**

La fonction `count()` renvoie 1 quand on y fait appel en lui passant une chaîne de caractères non vide à la place d'un tableau.

Cette fonction peut être utile dans le cadre de l'utilisation de boucles. Elle permet de déterminer à partir de quand doit s'arrêter un `while()` par exemple.

```
<?php
$tab[0] = 1;
$tab[1] = 3;
$tab[2] = 5;
$result = count($tab);
$i = 0;
while ($i <= $result){
    echo $tab[$i];
    $i++;
}
// Affiche 135
?>
```

**Note**

La fonction `sizeof()` est un alias de `count()`. Les deux noms correspondent à la même fonction.

## Recherche d'un élément

### *Présence dans le tableau*

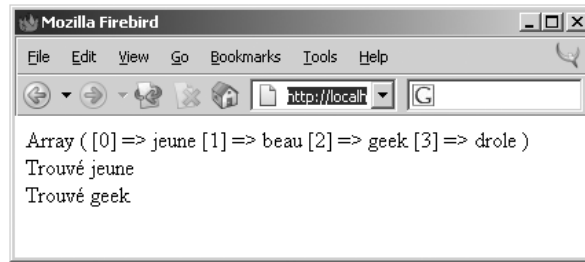
■ `in_array (expression, tableau [,strict])`

La fonction `in_array()` permet de savoir si un élément (désigné plus haut par « expression ») se trouve dans le tableau passé en argument. On peut également utiliser le dernier paramètre (optionnel) pour faire une vérification du type et s'assurer que l'élément cherché a bien le même type que l'élément trouvé ; dans le cas contraire, le chiffre 0 validera une recherche sur la chaîne '0', par exemple. Vous pouvez consulter le résultat de l'exemple suivant à la figure 6-1.

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
print_r($caract);
if (in_array('jeune', $caract)) {
    echo '<br>Trouvé jeune';
}
if (in_array ('geek', $caract)) {
    echo '<br>Trouvé geek';
}
?>
```

**Figure 6-1**

Présence d'un élément dans un tableau



La fonction `in_array()` est donc extrêmement utile, principalement lorsque vous utilisez des bases de données, car celles-ci renvoient généralement les informations dans des tableaux.

Le dernier paramètre, optionnel, permet de vérifier également que le type est le même. Si ce paramètre n'est pas défini, la fonction considérera que la chaîne de caractères '3' est égale à l'entier 3.

## Recherche de la clé correspondante

```
array_search (expression, tableau [,strict])
```

La fonction `in_array()` ne vous informe que de la présence d'un élément. Vous pouvez aussi récupérer la clé correspondant à l'élément recherché via la fonction `array_search()`. Elle fonctionne de manière similaire à `in_array()` mais renvoie la clé correspondante comme valeur de retour.

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
print_r($caract);
$cle = array_search('beau', $caract) ;
echo "La valeur 'beau' est à la clé $cle" ;
// Affiche La valeur 'beau' est à la clé 1
?>
```

Il est important de noter que la clé peut être nulle. Pour tester si une clé a été trouvée, il vous faudra utiliser l'opérateur `===` :

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
print_r($caract);
if (FALSE === array_search('jeune', $caract)) {
    echo '<br>Trouvé jeune';
}
if (in_array ('geek', $caract)) {
    echo '<br>Trouvé geek';
}
?>
```



## Nombre d'occurrences d'un élément

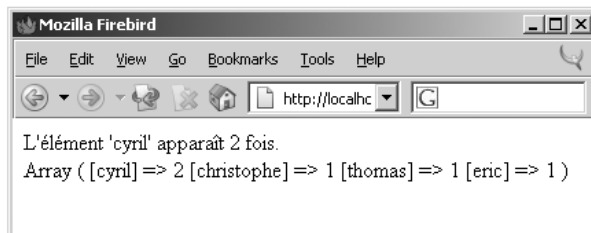
Vous pouvez chercher le nombre d'occurrences de chaque élément d'un tableau avec la fonction `array_count_values()`. Elle vous renverra un tableau associatif avec, pour chaque valeur, le nombre d'occurrences trouvées.

Dans l'exemple suivant, nous disposons d'un tableau contenant une liste de prénoms. Nous allons y compter le nombre d'occurrences de chaque prénom. Le résultat du script est visible à la figure 6-2.

```
<?php
$tab = array('cyril', 'christophe', 'cyril', 'thomas', 'eric' );
$cpt = array_count_values($tab) ;
echo "L'élément 'cyril' apparaît ", $cpt['cyril'], " fois.<br>";
print_r($tab);
?>
```

Figure 6-2

*Occurrences  
dans un tableau*



## Récupération aléatoire d'éléments

La fonction `array_rand()` permet de tirer au hasard un ou plusieurs éléments d'un tableau et de vous envoyer les clés associées. Elle prend en paramètres le tableau source et un nombre d'éléments à retourner.

```
<?php
$tab = array(1,2,3,4,5,6) ;
$rand = array_rand($tab, 2) ;
echo "Clés tirées au hasard : $rand[0] et $rand[1] <br>" ;
echo 'Element 0 : ', $tab[$rand[0]], ' <br>' ;
echo 'Element 1 : ', $tab[$rand[1]], ' <br>' ;
?>
```

Si vous ne spécifiez pas de second argument, un seul élément sera tiré au hasard. Quand un seul élément est retourné, la clé de cet élément vous sera directement retournée, elle ne sera pas dans un tableau.

```
<?php
$tab = array(1,2,3,4,5,6) ;
$rand = array_rand($tab) ;
echo "Clé tirée au hasard : $rand <br>" ;
echo 'Element : ', $tab[$rand], ' <br>' ;
?>
```

## Trier les tableaux

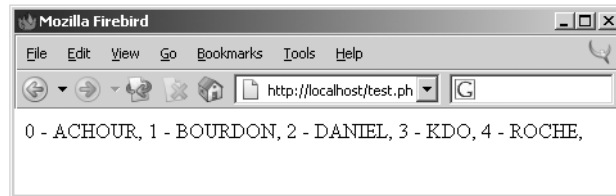
### Tri par valeur

Les entrées d'un tableau sont ordonnées. Il vous arrivera donc probablement souvent de vouloir trier les différents éléments par leur valeur. Vous pouvez alors utiliser la fonction `sort()` en fournissant votre tableau en argument. Cette fonction ne renvoie rien et se contente de trier le tableau par ordre alphabétique.

Dans l'exemple suivant, nous avons en entrée un tableau contenant le nom de certains relecteurs de ce livre et nous allons les classer par ordre alphabétique. Le résultat est visible dans la figure 6-3.

```
<?php
$caract = array ('ROCHE', 'KDO', 'BOURDON', 'DANIEL', 'ACHOUR');
sort($caract);
foreach($caract as $cle => $valeur) {
    echo "$cle - $valeur, ";
}
?>
```

Figure 6-3  
Classer un tableau



Un tableau peut contenir des index numériques et des index textuels. Vous pouvez forcer une comparaison numérique en fournissant `SORT_NUMERIC` en second paramètre. La constante `SORT_STRING` permet de forcer une comparaison textuelle. `SORT_REGULAR` est la valeur par défaut, qui permet de trier les deux types d'index.

```
sort($caract, SORT_STRING);
```

### Tri en ordre inverse

Il est possible de faire un tri inverse (les valeurs les plus grandes se retrouveront en premières positions) via la fonction `rsort()`, qui fonctionne de manière identique à `sort()`.

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
rsort($caract);
foreach($caract as $cle => $valeur) {
    echo "$cle - $valeur, ";
}
// Affiche 0 - jeune, 1 - geek, 2 - drole, 3 - beau,
?>
```

## Garder les associations clé-valeur

Les fonctions `sort()` et `rsort()` trient le tableau en redéfinissant les clés. Après conversion, le tableau sera indexé numériquement avec des index qui se suivent à partir de zéro.

Les fonctions `asort()` et `arsort()` sont en tous points identiques à ces premières, mais gardent les associations clé-valeur. Les différentes clés ne sont donc pas réécrites, seules les positions sont changées.

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
asort($caract) ;
foreach($caract as $cle => $valeur) {
    echo "$cle - $valeur, " ;
}
// Affiche 1 - beau, 3 - drole, 2 - geek, 0 - jeune,
?>
```

## Tri par clé

Les fonctions `ksort()` et `krsort()` sont identiques aux fonctions `asort()` et `arsort()`, si ce n'est que les tris se font en fonction des clés d'index et non des valeurs du tableau.

```
<?php
$caract = array ('jeune', 'beau', 'geek', 'drole');
asort($caract) ;
foreach($caract as $cle => $valeur) {
    echo "$cle - $valeur, " ;
}
// Affiche 1 - beau, 3 - drole, 2 - geek, 0 - jeune,

krsort($caract) ;
foreach($caract as $cle => $valeur) {
    echo "$cle - $valeur, " ;
}
// Affiche 3 - drole, 2 - geek, 1 - beau, 0 - jeune,
?>
```

## Tri naturel

Les tris faits avec la fonction `sort()` sont des tris informatiques. La chaîne `texte12` arrivera entre `texte1` et `texte2`. Il est possible d'obtenir un tri plus humain avec la fonction `natsort()`.

```
<?php
$caract = array ('texte1', 'texte3', 'texte2', 'texte12');
sort($caract) ;
foreach($caract as $valeur) {
    echo "$valeur, " ;
}
}
```

```
// Affiche texte1, texte2, texte2, texte3

natsort($caract) ;
foreach($caract as $valeur) {
    echo "$valeur, " ;
}
// Ici, le 12 passe après le 2
// Affiche texte1, texte2, texte3, texte12
?>
```

La fonction `natcasesort()` est équivalente à `natsort()`, mais fait une comparaison insensible à la casse.

### Trier avec une fonction utilisateur

Si les différentes méthodes de tri précédentes ne vous conviennent pas, il est possible de trier un tableau *via* une fonction utilisateur à l'aide de `usort()`. Il faut alors lui donner en arguments le tableau à trier et une fonction de comparaison.

La fonction de comparaison doit accepter deux valeurs en arguments et doit retourner un nombre inférieur, supérieur ou égal à zéro selon que la première valeur est inférieure, supérieure ou égale à la seconde.

```
<?php
// Tri par la taille de la chaîne de caractères
function cmp($a, $b) {
    if (strlen($a) < strlen($b)) {
        return -1 ;
    } elseif (strlen($a) == strlen($b)) {
        return 0 ;
    } else {
        return 1 ;
    }
}

$fruits[0] = 'citrons';
$fruits[1] = 'pommes';
$fruits[2] = 'abricots';

usort($fruits, 'cmp');

foreach($fruits as $cle => $valeur) {
    echo "$valeur, ";
}
// Affiche pommes, citrons, abricots
?>
```

Il existe aussi une fonction `uksort()`, qui fonctionne de manière identique, mais qui trie les clés au lieu des valeurs.

## Tri multicritère

La fonction `array_multisort()` permet de trier un tableau sur plusieurs critères, généralement plusieurs colonnes. Elle prend en arguments un ou plusieurs tableaux. Ces différents tableaux sont triés en fonction des valeurs du premier tableau. En cas de valeurs identiques, ce sont les valeurs du deuxième tableau qui seront analysées, et ainsi de suite. Les associations clé-valeur sont sauvegardées.

```
<?php
$tab1 = array( 2, 6, 9, 6 ) ;
$tab2 = array( 3, 2, 1, 8 ) ;

array_multisort($tab1, $tab2) ;

foreach($tab1 as $cle => $valeur) {
    echo "$cle-$valeur, " ;
}
// Affiche 0-2,1-6,3-6,2-9,

foreach($tab2 as $cle => $valeur) {
    echo "$cle-$valeur, " ;
}
// Affiche 0-3,1-2,3-8,2-1
?>
```

Il est possible d'ajouter des paramètres pour chaque critère de tri. Il suffit alors de passer différentes constantes en arguments, juste après le tableau à trier. Les constantes `SORT_ASC` et `SORT_DESC` permettent respectivement de définir un tri ascendant ou descendant. Les constantes `SORT_REGULAR`, `SORT_STRING` et `SORT_NUMERIC` sont les mêmes que pour la fonction `sort()`.

L'instruction suivante permet de trier les tableaux avec, comme premier critère, un tri numérique ascendant des valeurs du premier tableau et, comme deuxième critère, un tri descendant des valeurs du second tableau :

```
array_multisort($tab1, SORT_ASC, SORT_NUMERIC, $tab2, SORT_DESC) ;
```

## Extractions et remplacement

### Affecter des variables

Lorsqu'une fonction retourne un tableau, il est possible d'en récupérer les différents paramètres dans des variables en une opération. L'instruction `list()` prend en paramètres une liste de variables. Lors d'une affectation avec un tableau, la première variable se verra affecter la valeur de l'index 0 du tableau, la seconde se verra affecter la valeur de l'index 1, et ainsi de suite.

```
<?php
$tab = array(1, 2, 3, 4) ;
list($a, $b, $c, $d) = $tab ;
```

```
echo "$a-$b-$c-$d" ;  
// Affiche 1-2-3-4  
?>
```

Si vous utilisez un tableau associatif plutôt qu'un tableau indexé numériquement, vous pouvez utiliser `extract()` pour affecter chaque élément à une variable du même nom que son index.

```
<?php  
$tab = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4) ;  
extract($tab) ;  
echo "$a-$b-$c-$d" ;  
// Affiche 1-2-3-4  
?>
```

## Sérialisation de tableaux

Il est possible de convertir facilement une chaîne de caractères en tableau et un tableau en chaîne de caractères à l'aide des fonctions `explode()` et `implode()`.

La fonction `implode()` prend en arguments une chaîne séparatrice et un tableau. La valeur de retour sera alors la concaténation de toutes les valeurs du tableau séparées par la chaîne séparatrice.

```
<?php  
$tab = array( 1 , 3 , 5 , 6 ) ;  
echo implode('*', $tab) ;  
// Affiche 1*3*5*6  
?>
```

La fonction `explode()` fait l'opération inverse et divise une chaîne de caractères pour former un tableau à partir d'un séparateur :

```
<?php  
$tab = array( 1 , 3 , 5 , 6 ) ;  
$chaine = implode('*', $tab) ;  
echo $chaine ;  
// Affiche 1*3*5*6  
  
$tab = explode('*', $chaine) ;  
// On récupère le tableau d'origine  
?>
```

## Extraction d'un sous-tableau

Nous avons vu la fonction `substr()` au chapitre précédent ; elle permet de récupérer une sous-partie d'une chaîne de caractères à partir de sa position. La fonction `array_slice()` fonctionne de manière similaire pour les tableaux. Elle prend en paramètres un tableau référence, une position de départ et un nombre maximal d'éléments à retourner. Elle va alors vous retourner les éléments du tableau référence à partir de la position spécifiée.

```
<?php
$tab = array(1,2,3,4,5,6,7) ;
$soustab = array_slice($tab, 2, 3) ;
echo implode(',', $soustab) ;
// Affiche 3,4,5
?>
```

**Attention**

La position de départ est numérotée à partir de zéro.

Comme pour `substr()`, si vous ne définissez pas de nombre maximal d'éléments, `array_slice()` vous retournera tous les éléments jusqu'à la fin du tableau.

## Remplacement d'un sous-tableau

La fonction `array_splice()` fonctionne de manière similaire à `substr_replace()`. Elle permet de remplacer une sous-partie d'un tableau par une autre. Le tableau référence et la position du premier élément à remplacer sont à fournir en premier et second paramètres. Si le second paramètre est négatif, la position de l'élément à remplacer est calculée à partir de la fin du tableau au lieu du début.

Un nombre d'éléments à remplacer est donné en troisième argument ; s'il n'est pas présent, ce sont tous les éléments à partir de la position de départ et jusqu'à la fin du tableau qui sont remplacés. S'il est négatif, il désigne un nombre d'éléments qui seront laissés à la fin du tableau, les éléments jusque-là seront remplacés.

Les éléments à insérer à la place des anciens sont à fournir en quatrième et dernier paramètres. Il n'est pas obligatoire que les nombres d'éléments insérés et remplacés soient identiques. Si vous ne spécifiez pas ce quatrième paramètre, PHP se contentera de supprimer les éléments, sans rien insérer à la place.

```
<?php
$tab = array('rouge', 'vert', 'bleu', 'jaune');
array_splice($tab, 2);
// $tab est maintenant array('rouge', 'vert')

$tab = array('rouge', 'vert', 'bleu', 'jaune');
array_splice($tab, 1, -1);
// $tab est maintenant array('rouge', 'jaune')

$tab = array('rouge', 'vert', 'bleu', 'jaune');
array_splice($tab, 1, count($tab), 'orange');
// $tab est maintenant array('rouge', 'orange')

$tab = array('rouge', 'vert', 'bleu', 'jaune');
array_splice($tab, -1, 1, array('noir', 'marron'));
// $tab est maintenant array('rouge', 'vert',
// 'bleu', 'noir', 'marron')
```

```
$tab = array('rouge', 'vert', 'bleu', 'jaune');
array_splice($tab, 3, 0, 'violet');
// $tab est maintenant array('rouge', 'vert',
// 'bleu', 'violet', 'jaune');
?>
```

## Gestion des clés et des valeurs

### Liste des clés utilisées

La fonction `array_keys()` vous renvoie un tableau indexé numériquement et contenant la liste des clés utilisées dans le tableau passé en argument.

```
<?php
$tab = array( 'a' => 1 , 'c' => 5 ) ;
$cles = array_keys($tab) ;
echo implode('-', $cles) ;
// Affiche a-c
?>
```

### Liste des valeurs utilisées

La fonction `array_values()` est équivalente à la fonction `array_keys()`, mais récupère les valeurs du tableau au lieu des clés. Il s'agit en fait de convertir un tableau associatif en un tableau indexé numériquement à partir de zéro.

```
<?php
$tab = array( 'a' => 1 , 'c' => 5 ) ;
$val = array_values($tab) ;

$cles = array_keys($val) ;
echo implode('-', $cles) ;
// Affiche 0-1
echo implode('-', $val) ;
// Affiche 1-5
?>
```

### Échanger les clés et les valeurs

Vous pouvez intervertir les clés et les valeurs d'un tableau associatif avec la fonction `array_flip()`. Les clés deviendront alors les valeurs et les valeurs deviendront les clés. Si une valeur avait plusieurs occurrences, seule la dernière serait utilisée.

```
<?php
$tab = array( 'a' => 1 , 'c' => 5 ) ;
$flip = array_flip($tab) ;

$cles = array_keys($flip) ;
```



```
echo implode('-', $cles) ;  
// Affiche 1-5  
  
echo implode('-', $flip) ;  
// Affiche a-c  
?>
```

## Fusions et séparations

### *Fusion de plusieurs tableaux*

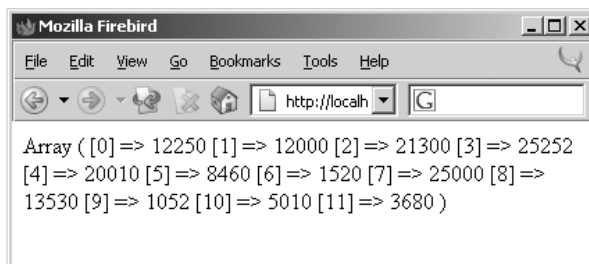
```
array_merge (tableau1, tableau2 [, tableau ...])
```

`array_merge()` permet de fusionner plusieurs tableaux. Cette fonction rassemble les éléments de plusieurs tableaux, en ajoutant les valeurs de l'un à la fin de l'autre. La fonction renvoie un tableau. Dans le cas des tableaux associatifs, s'ils ont des clés communes, la dernière valeur rencontrée écrasera la précédente. Le résultat de l'exemple suivant est donné à la figure 6-4.

```
<?php  
$result_2002 = array( 12250, 12000, 21300, 25252, 20010, 8460);  
$result_2003 = array( 1520, 25000, 13530, 1052, 5010, 3680);  
$result_2002_2003 = array_merge($result_2002, $result_2003);  
  
print_r($result_2002_2003);  
?>
```

Figure 6-4

*Fusion de tableaux*



### Fusion récursive

Si vos tableaux contiennent d'autres tableaux, il est aussi possible de les fusionner récursivement avec `array_merge_recursive()` ; les sous-tableaux sont alors fusionnés entre eux aussi.

```
<?php  
$tab1 = array('a' => 1, 'b' => array( 2 ) ) ;  
$tab2 = array('c' => 3, 'b' => array( 4 ) ) ;  
$merge = array_merge_recursive($tab1, $tab2) ;
```

```
// Les valeurs de $tab2 non présentes dans $tab1 on été ajoutées
echo $merge['a'] , ' - ', $merge['c'], '<br>' ; // Affiche 1 - 3

// Les sous-tableaux ont été fusionnés :
echo implode(' - ', $merge['b']) ; // Affiche 2 - 4
?>
```

Si la fonction compare un tableau avec une valeur unique, la valeur sera ajoutée au tableau :

```
<?php
$tab1 = array('a' => 1, 'b' => array( 2 ) ) ;
$tab2 = array('c' => 3, 'b' => 4 ) ;
$merge = array_merge_recursive($tab1, $tab2) ;

// La valeur a été ajoutée au sous-tableau
echo implode(' - ', $merge['b']) ; // Affiche 2 - 4
?>
```

## Séparation d'un tableau en plusieurs

La fonction `array_chunk()` est la fonction inverse de `array_merge()`. Elle permet de séparer un grand tableau passé en premier paramètre en plusieurs petits, la taille de ces derniers étant déterminée par le second argument.

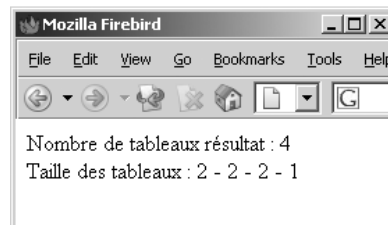
Le résultat du code suivant est donné à la figure 6-5 :

```
<?php
$tab = array(1,2,3,4,5,6,7) ;
// On sépare en petits tableaux de 2 éléments chacun
$tabs = array_chunk($tab, 2) ;

// Teste le nombre de tableaux résultats :
echo 'Nombre de tableaux résultat : ' ,
    count($tabs), '<br>' ;

// Teste la taille des tableaux résultats
echo 'Taille des tableaux : ' , count($tabs[0]) , ' - ' ;
echo count($tabs[1]), ' - ' , count($tabs[2]), ' - ' ;
echo count($tabs[3]), '<br>' ;
?>
```

**Figure 6-5**  
*Séparation de  
tableaux*



Lors de la réécriture des tableaux, la fonction `array_chunk()` écrase les clés et indexe les nouveaux tableaux numériquement. Vous pouvez demander de garder les associations clé-valeur en spécifiant un booléen vrai en troisième argument.

```
$tabs = array_chunk($tab, 2, TRUE) ;
```

## Différences et intersections

### Différences entre tableaux

La fonction `array_diff()` vous permet de calculer la différence entre plusieurs tableaux. Elle prend en paramètres deux tableaux (ou plus), et renvoie la liste des éléments qui sont dans le premier tableau mais dans aucun des suivants. L'association entre les clés et les valeurs est préservée.

```
<?php
$tab1 = array(1, 2, 3, 4, 5, 6, 7) ;
$tab2 = array(1, 3, 5, 7) ;
$tab3 = array(1, 2, 3) ;
$diff = array_diff($tab1, $tab2, $tab3) ;
echo implode('-', $diff) ;
// Affiche 4-6
?>
```

La fonction `array_diff_assoc()` fonctionne de manière similaire à `array_diff()`, mais vérifie aussi la correspondance des clés. Un élément du premier tableau sera renvoyé s'il n'est présent dans aucun autre tableau avec la même clé comme index.

```
<?php
$tab1 = array(1, 2, 3, 4, 5, 6, 7) ;
$tab2 = array(1, 3, 5, 4) ;
$tab3 = array(1, 7, 3) ;
$diff = array_diff_assoc($tab1, $tab2, $tab3) ;
echo implode('-', $diff) ;
// Affiche 2-5-6-7
?>
```

### Intersections entre deux tableaux

À l'inverse de `array_diff()`, la fonction `array_intersect()` retourne la liste des éléments du premier tableau qui sont présents dans tous les autres tableaux donnés en arguments.

```
<?php
$tab1 = array(1, 2, 3, 4, 5, 6, 7) ;
$tab2 = array(1, 3, 5, 7) ;
$tab3 = array(1, 2, 3) ;
$inter = array_intersect($tab1, $tab2, $tab3) ;
echo implode('-', $inter) ;
// Affiche 1-3
?>
```

La fonction `array_intersect_assoc()` est similaire, mais vérifie aussi que les index sont identiques. Un élément du premier tableau n'est renvoyé que s'il est présent dans tous les autres avec le même index.

## Gestion des doublons

La fonction `array_unique()` prend en argument un tableau et enlève les doublons de valeurs.

```
<?php
$tab = array(1, 2, 3, 4, 5, 2, 4) ;
echo implode('-', $tab), '<br>' ;
// Affiche 1-2-3-4-5-2-4

$tab = array_unique($tab) ;
echo implode('-', $tab) ;
// Affiche 1-2-3-4-5
?>
```

## Gestion des piles et des files

PHP n'a pas de type de données spécifique pour gérer des piles et des files. On peut cependant utiliser les tableaux pour gérer les mêmes fonctionnalités.

Une pile est une liste d'éléments qui sont gérés selon la règle « premier entré, dernier sorti ». La gestion d'une pile nécessite deux fonctions qui sont remplies par `array_pop()` et `array_push()`.

La fonction `array_push()` prend en argument un tableau et une ou plusieurs valeur(s). Les valeurs spécifiées seront ajoutées à la fin du tableau : on parle alors d'empilage.

```
<?php
$tab = array() ;
array_push($tab, 1, 3, 5) ;
/* Équivalent à */
$tab = array() ;
$tab[] = 1 ;
$tab[] = 3 ;
$tab[] = 5 ;
?>
```

La fonction `array_pop()` permet de dépiler un élément du tableau passé en argument, c'est-à-dire retourner la dernière valeur du tableau et de l'effacer. Si le tableau est vide, la valeur NULL est retournée.

```
<?php
$tab = array() ;
array_push($tab, 1, 3, 5) ;
echo array_pop($tab) ; // Affiche 5
echo array_pop($tab) ; // Affiche 3
?>
```

Il est aussi possible de gérer des files, « premier entré, premier sorti », grâce aux fonctions supplémentaires `array_unshift()` et `array_shift()`. Ces deux fonctions sont similaires à `array_push()` et `array_pop()` mais agissent sur le début du tableau et non sur la fin.

## Navigation dans les tableaux

La structure `foreach` vue au chapitre 4 permet de naviguer facilement à travers un tableau. Il est toutefois possible de naviguer manuellement à travers un tableau avec un jeu de quelques fonctions.

PHP stocke avec le tableau un curseur qui pointe vers un des éléments du tableau. Traverser le tableau veut dire faire évoluer ce curseur à travers toutes les positions et à chaque fois lire l'élément pointé.

Les fonctions `reset()`, `next()`, `prev()` et `end()` déplacent respectivement le curseur à la position zéro, suivante, précédente et à la dernière position. Elles prennent toutes le tableau à traverser comme unique argument et retournent la valeur du nouvel élément pointé, ou la valeur `FALSE` en cas d'erreur. La fonction `current()` se déroule de manière similaire, mais retourne la valeur courante sans déplacer le curseur.

Ces fonctions sont généralement difficiles à utiliser car elles ne permettent pas de faire la différence entre une erreur et un élément de tableau ayant la valeur `FALSE`. Elles ne permettent pas non plus de récupérer les index utilisés dans le tableau. Il est possible de simplifier en utilisant la fonction `each()`. Elle prend en paramètre un tableau, retourne une liste contenant la clé et la valeur courante, puis incrémente la position du curseur. La valeur `FALSE` est renvoyée quand le curseur a dépassé le dernier élément.

```
<?php
$tab = array('a', 'b', 'c') ;
reset($tab) ;
while( list($cle, $valeur) = each($tab) ) {
    echo "$cle-$valeur," ;
}
// Affiche 0-a,1-b,2-c,
?>
```

### Note

Utiliser une fonction de traitement sur le tableau modifiera généralement la position du curseur que vous utilisez.

# 7

## Fonctions usuelles

---

L'une des forces de PHP est son grand nombre de fonctions, traitant de quasi tous les domaines en rapport avec Internet et même bien au-delà. Dans ce chapitre, nous allons vous présenter quelques-unes de ces fonctions importantes que nous n'avons pu associer à un chapitre en particulier. Nous aborderons des fonctions d'aide au débogage, de gestion des dates, mais aussi des fonctions réseau et de chiffrement.

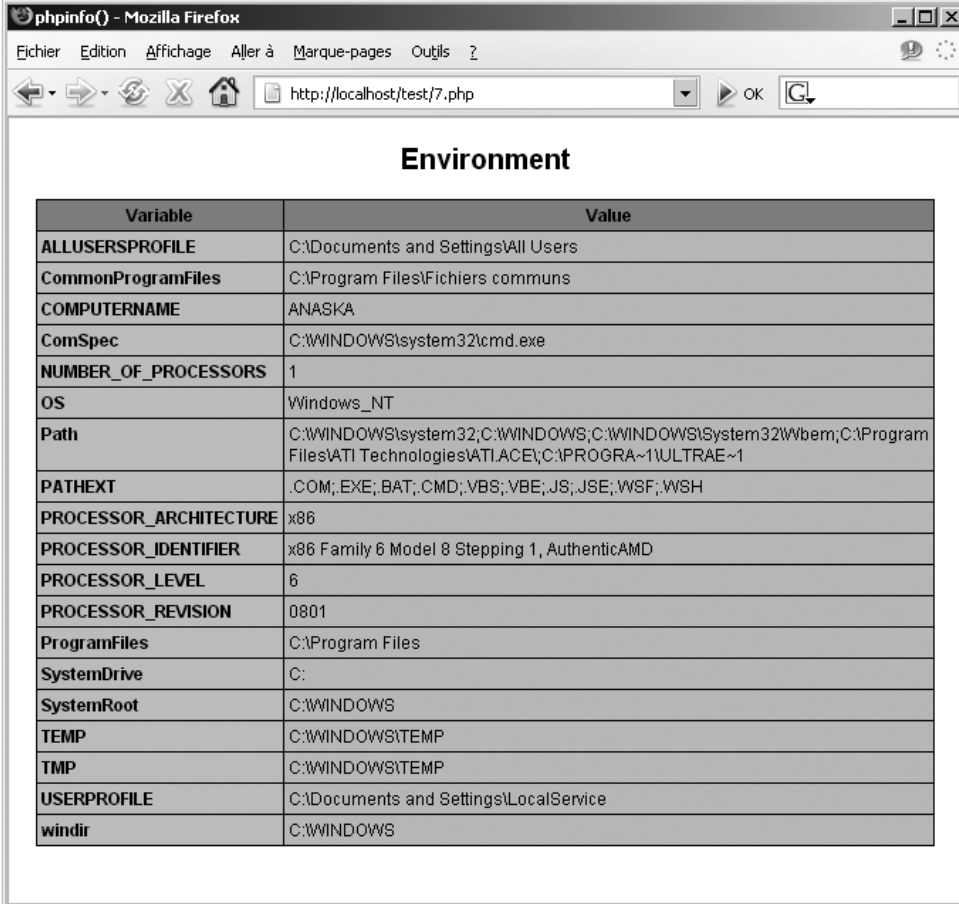
### Fonction d'affichage

#### *Informations de configuration*

PHP peut vous communiquer tout ce qu'il sait sur sa configuration grâce à la célèbre fonction `phpinfo()`. En mettant simplement un appel à cette fonction dans un script, PHP renvoie une page HTML avec des informations sur le système, les options qui ont servi pendant la compilation, les extensions activées, les variables d'environnement, les entêtes HTTP et les informations de configuration.

```
■ phpinfo ( [type d'information] )
```

Comme tous les systèmes sont configurés différemment, `phpinfo()` sert généralement à vérifier la configuration ainsi que les variables prédéfinies, pour une plate-forme donnée (voir exemple à la figure 7-1).



Variable	Value
ALLUSERSPROFILE	C:\Documents and Settings\All Users
CommonProgramFiles	C:\Program Files\Fichiers communs
COMPUTERNAME	ANASKA
ComSpec	C:\WINDOWS\system32\cmd.exe
NUMBER_OF_PROCESSORS	1
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATIACE\;C:\PROGRA~1\ULTRAE~1
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE	x86
PROCESSOR_IDENTIFIER	x86 Family 6 Model 8 Stepping 1, AuthenticAMD
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	0801
ProgramFiles	C:\Program Files
SystemDrive	C:
SystemRoot	C:\WINDOWS
TEMP	C:\WINDOWS\TEMP
TMP	C:\WINDOWS\TEMP
USERPROFILE	C:\Documents and Settings\LocalService
windir	C:\WINDOWS

Figure 7-1

Fonction `phpinfo()` : variables d'environnement

L'affichage peut être personnalisé en utilisant une ou plusieurs des constantes définies dans le tableau 7-1. Elles peuvent être combinées avec l'opérateur `OR` et doivent être passées en paramètres. Vous pouvez aussi les additionner.

Tableau 7-1 Les options de la fonction `phpinfo()`

Nom (constante)	Valeur	Description
INFO_GENERAL	1	Ligne de configuration, chemin du <code>php.ini</code> , date de compilation, serveur web, système, etc.
INFO_CREDITS	2	Crédits de PHP. Voir aussi <code>phpcredits()</code> .
INFO_CONFIGURATION	4	Valeurs courantes locales et générales des directives PHP. Voyez aussi la fonction <code>ini_get()</code> .

Tableau 7-1 Les options de la fonction `phpinfo()`

Nom (constante)	Valeur	Description
INFO_MODULES	8	Modules chargés et leur configuration spécifique.
INFO_ENVIRONMENT	16	Informations sur les environnements de variables, qui sont disponibles dans la variable <code>\$_ENV</code> .
INFO_VARIABLES	32	Toutes les variables prédéfinies, issues de l'environnement, la méthode GET, la méthode POST, les cookies et le serveur.
INFO_LICENSE	64	Licence PHP.

```
<?php
// Affiche toutes les informations
phpinfo();

// Affiche uniquement le module d'information
phpinfo(INFO_MODULES);
// phpinfo(10) fournirait les informations correspondant
// au module 8 et au module 2 (addition des valeurs).
?>
```

**Note**

Certaines des informations affichées sont désactivées si la directive `expose_php` est configurée avec la valeur `off`. Cela inclut les logos PHP et Zend, ainsi que les crédits.

Vous pouvez également n'afficher que les variables EGPCS (*Environment, GET, POST, Cookie, Session*) et donc vous servir de la fonction `phpinfo(32)` pour déboguer (voir figure 7-2).

Figure 7-2  
*phpinfo()*  
pour aider  
au débogage

Variable	Value
<code>_REQUEST["nouvelles"]</code>	php5
<code>_REQUEST["id"]</code>	156
<code>_GET["nouvelles"]</code>	php5
<code>_GET["id"]</code>	156
<code>_SERVER["COMSPEC"]</code>	C:\WINDOWS\system32\cmd.exe
<code>_SERVER["DOCUMENT_ROOT"]</code>	d:/www
<code>_SERVER["HTTP_ACCEPT"]</code>	application/x-shockwave-flash;text/xml,application/xml,application/xhtml+xml;text/html;q=0.9,t
<code>_SERVER["HTTP_ACCEPT_ENCODING"]</code>	gzip, deflate



## Affichage de débogage

Lorsque vous travaillez, et en l'absence de débogueur, il est souvent utile de pouvoir afficher n'importe où dans votre script la valeur d'une variable. L'affichage de tableaux ou d'objets nécessite des lignes spécifiques ; or il n'est pas toujours simple ou pratique d'avoir à faire dix lignes de code juste pour afficher une variable en prenant en compte son type. Pour cela, on peut utiliser la fonction `print_r()` :

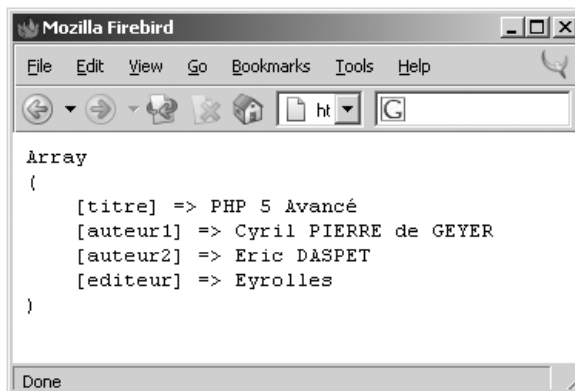
```
print_r (variable)
```

La fonction `print_r()` affiche des informations à propos d'une variable, de manière qu'elle soit lisible. Cette fonction affiche tels quels les entiers, chaînes ou nombres à virgule flottante. Pour les tableaux et les objets, les valeurs seront présentées dans un format explicitant les associations clé-valeur. Un exemple est donné à la figure 7-3.

```
<?php
$tab = array('titre'=>'PHP 5 Avancé',
            'auteur1' => 'Cyril PIERRE de GEYER',
            'auteur2' => 'Eric DASPET',
            'editeur'=> 'Eyrolles');

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

Figure 7-3  
Fonction `print_r()`

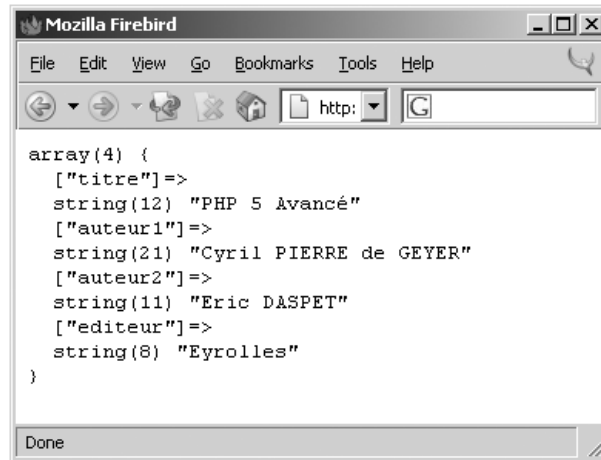


La fonction `print_r()` permet de donner un aperçu du contenu qui soit facilement lisible par un humain. Il est possible d'obtenir une description de variable plus formelle et plus complète avec la fonction `var_dump()` (un affichage par `var_dump()` est montré à la figure 7-4).

```
<?php
$tab = array('titre'=>'PHP 5 Avancé',
            'auteur1' => 'Cyril PIERRE de GEYER',
```

```
'auteur2' => 'Eric DASPET',  
        'editeur'=> 'Eyrolles');  
echo '<pre>' ;  
var_dump($tab);  
echo '</pre>' ;  
?>
```

**Figure 7-4**  
Fonction  
`var_dump()`



#### Note

Ces fonctions retournent un affichage orienté texte, sans balisage HTML. Pour une meilleure clarté lors des débogages sur navigateur, il vous est conseillé d'entourer ces fonctions d'un tag de préformatage (`<pre>`).

## Coloration syntaxique de code

Vous avez peut être remarqué sur certains sites traitant de PHP que le code donné dans les tutoriaux est coloré. Cette coloration n'est généralement pas faite à la main. PHP propose par défaut une fonction, `highlight_string()`, qui permet de colorer un code PHP et de l'envoyer à l'affichage.

■ `highlight_string` (code)

Il est aussi possible d'afficher directement le contenu d'un fichier de cette façon, au lieu d'utiliser une chaîne de caractères. Il vous suffit pour cela de faire appel à `highlight_file()` au lieu de `highlight_string()`.

■ `highlight_file` (fichier)

## Fonctions mathématiques

Les fonctions présentées ici fonctionnent uniquement avec des entiers signés sur 32 bits et des nombres à virgule flottante. Pour des traitements mathématiques plus poussés, reportez-vous à l'extension `BCMath`.

### Connaître les extrémités

La fonction `max()` retourne la plus grande valeur numérique parmi les valeurs passées en paramètres.

```
max (arg1 , arg2 [, ... ])
```

Si le premier paramètre est un tableau, `max()` retourne la plus grande valeur de ce tableau. Si le premier paramètre est un entier, une chaîne ou un nombre à virgule flottante, `max()` requiert au moins deux paramètres et retourne alors le plus grand d'entre eux. Le nombre d'arguments est alors illimité.

Si au moins une valeur est un nombre à virgule flottante, elles seront toutes traitées comme des nombres à virgule flottante.

```
<?php
echo max(1, 3, 5, 6, 7); // 7
echo max(array(2, 4, 5)); // 5
?>
```

Avec plusieurs tableaux, la fonction `max()` fait les comparaisons de gauche à droite et renvoie un tableau.

```
<?php
$val = max(array(2, 4, 8), array(2, 5, 7)); // array(2, 5, 7)
?>
```

La fonction `min()` est l'opposée de `max()`. Elle retourne la valeur la plus petite. Le fonctionnement est en tout point identique à `max()`.

### Arrondir des valeurs

Trois fonctions permettent d'arrondir des nombres : `round()` renvoie l'entier le plus proche, `floor()` arrondit à l'entier immédiatement inférieur et `ceil()` retourne l'entier immédiatement supérieur.

```
round (nombre [, précision] )
```

La fonction `round()` retourne la valeur arrondie du nombre passé en paramètre à l'entier le plus proche. Si vous définissez une précision via le second paramètre, la fonction arrondira alors à cette précision. Une précision négative permettra d'arrondir à une puissance de 10. La valeur `-2` permettra par exemple d'arrondir à la centaine.

```
<?php
echo round(3.4); // Affiche 3
```

```
echo round(3.5); // Affiche 4
echo round(3.6); // Affiche 4
echo round(3.6, 0); // Affiche 4
echo round(1.95583, 2); // Affiche 1.96
echo round(1241757, -3); // Affiche 1242000
?>
```

### Arrondi supérieur

```
ceil (nombre )
```

La fonction `ceil()` retourne l'entier supérieur du nombre passé en paramètre. Utiliser `ceil()` sur un entier ne sert à rien.

```
<?php
echo ceil(7.2); // 8
echo ceil(99.999); // 100
?>
```

### Arrondi inférieur

```
floor(nombre )
```

La fonction `floor()` retourne l'entier inférieur du nombre passé en paramètre.

```
<?php
echo floor(7.2); // 7
echo floor(99.999); // 99
?>
```

## Créer des valeurs aléatoires

Que ce soit pour tester le comportement de votre application ou pour tout autre chose, il est utile de pouvoir créer des valeurs aléatoires.

Par défaut, PHP utilise le générateur de nombres aléatoires du système avec la fonction `rand()`. Cependant, nous vous recommandons d'utiliser la fonction `mt_rand()` car celle-ci utilise le générateur de nombres aléatoires « Mersenne Twister », qui produit des nombres utilisables en cryptographie et qui est quatre fois plus rapide.

```
mt_rand([minimum,maximum])
```

Pour obtenir un nombre entier entre 5 et 15 inclus, il faut utiliser `mt_rand(5,15)`. Le rendu de l'exemple suivant est donné à la figure 7-5.

```
<?php
$salaire = mt_rand(1,6000) ;
if ($salaire < 1000) {
    echo "Vous êtes payé $salaire _, c'est en dessous du SMIC";
} elseif ($salaire < 3000) {
    echo "Avec $salaire _, vous êtes raisonnablement bien payé";
} else {
```

```

    echo "$salaire _ ! Contactez-moi, votre travail m'intéresse !";
}
?>

```

Figure 7-5

Utilisation de la fonction `mt_rand()`



## Travailler sur différentes bases

On travaille généralement avec des chiffres codés sur la base 10. Cependant, il est parfois nécessaire de travailler sur des binaires (base 2) ou des hexadécimaux (base 16). Pour gérer des conversions de nombres entre différentes bases, PHP propose la fonction `base_convert()` :

```
base_convert (nombre , frombase , tobase )
```

La fonction `base_convert()` retourne une chaîne contenant l'argument nombre représenté dans la base `tobase`. La base de représentation de nombre est donnée par `frombase`. `frombase` et `tobase` doivent être compris entre 2 et 36 inclus. Les bases supérieures à 10 seront représentées avec les lettres de `a=10` à `z=36`. La figure 7-6 illustre le résultat de l'exemple suivant.

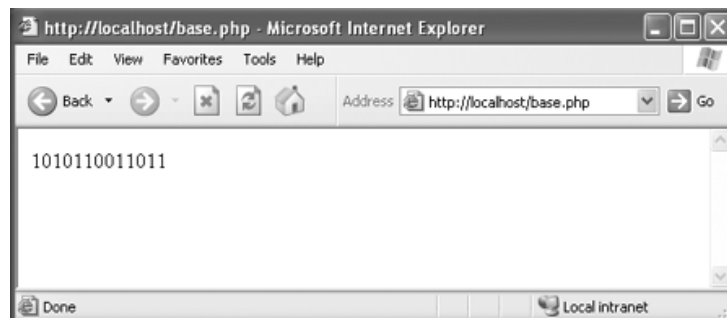
```

<?php
$binaire = base_convert(5531, 10, 2);
echo $binaire;
?>

```

Figure 7-6

Conversion de bases



Des fonctions dédiées à une conversion donnée existent pour des accès rapides :

- `bindec()` : convertit de binaire en décimal ;
- `decbin()` : convertit de décimal en binaire ;
- `dechex()` : convertit de décimal en hexadécimal ;
- `decoct()` : convertit de décimal en octal.

## Fonctions de date

### Note

Depuis PHP 5.2, de nouvelles fonctions de date sont apparues. Elles permettent d'utiliser des dates sous forme objet plutôt que des timestamp Unix. Ces objets sont créés à partir de la fonction `date_create()`. Ces objets bénéficient des mêmes possibilités que les anciens timestamp mais aussi d'une gestion des fuseaux horaires beaucoup plus poussée.

### Formater une date/heure locale

```
date (format , timestamp )
```

La fonction `date()` retourne une date sous forme d'une chaîne, au format donné par le premier paramètre. La date est fournie par le second paramètre sous la forme d'un *timestamp* Unix. Si le second paramètre n'est pas renseigné, la date courante est utilisée.

Le format utilisé en premier paramètre permet de définir ce qui sera retourné par la fonction. Par exemple, la chaîne `Y-m-d` permet d'afficher la date sous la forme `2004-01-31`. Une description des différents caractères significatifs spécifiables dans le format est donnée au tableau 7-2.

```
<?php
$format = 'd-m-y, H:i:s' ;
$date = date($format) ;
echo $date ;
// Affiche une date sous la forme : 31-04-04, 23:59:59
?>
```

Tableau 7-2 Les paramètres du format de date

Caractère	Description	Exemple
d	Jour du mois, sur deux chiffres (avec un zéro initial)	01 à 31
g	Heure, au format 12h, sans les zéros initiaux	1 à 12
G	Heure, au format 24h, sans les zéros initiaux	0 à 23
h	Heure, au format 12h, avec les zéros initiaux	01 à 12
H	Heure, au format 24h, avec les zéros initiaux	00 à 23

Tableau 7-2 Les paramètres du format de date (*suite*)

Caractère	Description	Exemple
i	Minute avec les zéros initiaux	00 à 59
j	Jour du mois sans les zéros initiaux	1 à 31
m	Mois au format numérique, avec zéros initiaux	01 à 12
M	Mois, en trois lettres, en anglais	Jan à Dec
n	Mois sans les zéros initiaux	1 à 12
s	Seconde avec zéros initiaux	00 à 59
W	Numéro de la semaine dans l'année	42
Y	Année à quatre chiffres	2004
y	Année à deux chiffres	04
Z	Jour de l'année	312

### Timestamp Unix

Le *timestamp* est l'unité couramment utilisée pour les dates sur les systèmes Unix. Il s'agit d'un décompte des secondes écoulées depuis 1970 (il est toutefois possible sur la plupart des systèmes d'accéder aux dates entre 1901 et 1970 en utilisant un entier signé négatif pour le nombre de secondes). Ce format a deux avantages : il utilise un simple nombre entier pour être stocké et facilite les manipulations de dates (on peut directement ajouter ou soustraire des dates et des périodes, toutes exprimées en secondes).

Sur les systèmes 32 bits, ce format présente le désavantage d'être limité à l'année 2038 (à cette date, le nombre de secondes écoulées dépassera la taille d'un entier 32 bits). Le pari fait est qu'à cette date, la plupart des systèmes seront en 64 bits, donc que la limite de 2038 sera reculée à une date quasi inatteignable.

Pour calculer un *timestamp* à partir d'une représentation de date, pour pouvez utiliser la fonction `strtotime()`, qui convertit une date ISO en *timestamp*. Il existe plusieurs formats de date, mais ceux que vous rencontrerez sur Internet (dans les formats de fichiers standards ou dans les bases de données par exemple) seront généralement compris par `strtotime()`.

```
<?php
echo strtotime ('16 November 1976');
echo strtotime ('1976-11-16');
?>
```

De plus, certaines bases de données disposent de fonctions pour convertir leurs propres formats de date en *timestamps* (avec MySQL, on utilise la fonction `UNIX_TIMESTAMP()`).

Pour obtenir un *timestamp* à partir des différentes composantes d'une date, on utilise la fonction `mktime()` :

```
mktime (heure, minute, secondes, mois, jour, annee)
```

**Attention**

L'ordre des arguments est différent de celui de la commande Unix habituelle `mktime()`, et fournit des résultats aléatoires si on oublie cet ordre. C'est une erreur très commune que de se tromper de sens.

Les arguments peuvent être omis, de droite à gauche, et tous les arguments manquants sont utilisés avec la valeur courante de l'heure et du jour. Le dernier argument prend en compte l'heure d'hiver et l'heure d'été. La valeur 1 indique l'heure d'hiver, 0 l'heure d'été, et dans le doute on met -1. L'exemple suivant est illustré à la figure 7-7.

```
<?php
echo date("d/M/Y", mktime (0,0,0,12,32,2004))."<br>";
echo date("d/m/Y", mktime (0,0,0,13,1,2005))."<br>";
echo date("M-d-Y", mktime (0,0,0,1,1,2001));
?>
```

**Figure 7-7***Fonction de date***Comparer les dates**

Les fonctions de date peuvent être utiles à de nombreuses autres tâches, par exemple vérifier la validité d'une date (pour contrôler une valeur saisie par un utilisateur).

```
checkdate ( mois, jour, annee )
```

La fonction `checkdate()` renvoie `TRUE` si la date représentée par le jour, le mois et l'année donnés en paramètres est valide, `FALSE` sinon. Les années bissextiles sont prises en compte.

```
<?php
// On récupère la date d'anniversaire fournie
// sous la forme JJ/MM/AAAA
$anniversaire = $_REQUEST['anniversaire'];
$tab = explode("/", $anniversaire);

// On vérifie que cette date est valide
if (checkdate($tab[1], $tab[0], $tab[2])){
    echo 'La date est valide.';
}
```



```
}elseif  
    echo 'La date est invalide.';  
}  
?>
```

### Ordre des arguments

Remarquez que l'ordre des arguments n'est pas l'ordre utilisé couramment en France, mais celui des pays anglophones.

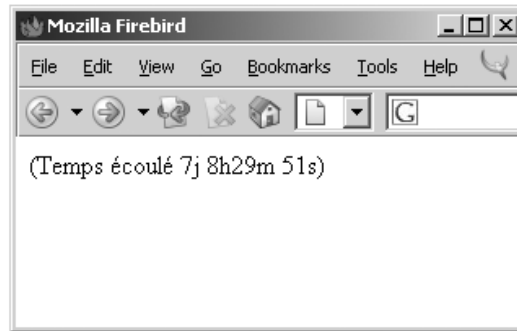
Il est également possible, grâce aux fonctions de date, de calculer les temps d'exécution de tout ou partie de vos scripts. Il vous faut pour cela utiliser la fonction `time()`, qui renvoie le *timestamp* actuel, aux différents moments que vous souhaitez évaluer et de stocker les valeurs afin de les comparer. La fonction `microtime()` fonctionne de manière identique, mais renvoie aussi la partie décimale de l'heure en cours (les millièmes de seconde).

```
<?php  
$tps1 = time();  
// Divers traitements.  
$tps2 = time();  
$tps = $tps2 - $tps1;  
echo "Le temps nécessaire à l'exécution des traitements est $tps";  
?>
```

Une solution plus élégante et généraliste permettant de calculer un délai pourrait être donnée par le code suivant (illustré à la figure 7-8) :

```
<?php  
function tempspasse($time)  
{  
    // Calcul du temps écoulé (en secondes)  
    $diff = time()-$time;  
    $dif_jour = floor($diff/60/60/24);  
    $diff -= $dif_jour*60*60*24;  
    $dif_heure = floor($diff/60/60);  
    $diff -= $dif_heure*60*60;  
    $dif_min = floor($diff/60);  
    $diff -= $dif_min*60;  
    $dif_sec = $diff;  
    return '(Temps écoulé '.$dif_jour.'j '.$dif_heure.'h'.  
        $dif_min.'m '.$dif_sec.'s)';  
}  
/* On va afficher le temps qui s'est écoulé depuis  
la création du fichier phpinfo.php: */  
echo tempspasse(filectime('phpinfo.php'));  
?>
```

Figure 7-8  
Fonctions de date



## Fonctions réseau

PHP 5 dispose de nombreuses fonctions permettant d'obtenir des informations réseau.

### Résolution DNS d'une adresse IP

■ `checkdnsrr (hôte [, type])`

La fonction `checkdnsrr()` vérifie qu'il existe bien un enregistrement DNS de type `type` correspondant au paramètre `hôte`. Elle renvoie `TRUE` si un enregistrement a été trouvé, et `FALSE` en cas d'échec.

Le paramètre `type` peut être l'une des valeurs suivantes : `A` (enregistrement classique `IPV6`), `MX` (serveur de courrier électronique), `NS` (serveur de nom), `SOA`, `PTR`, `CNAME` (alias), `AAAA` (adresse `IPV6`), ou `ANY` (composé de tous les autres). La valeur par défaut est `MX`. Le paramètre `host` peut être soit une adresse IP au format numérique, soit un nom d'hôte.

```
<?php
if (checkdnsrr("anaska.com"))
    echo "Le nom de domaine existe";
?>
```

Cette fonction n'est pas disponible sous Microsoft Windows, mais il est possible de la simuler :

```
<?php
function myCheckDNSRR($hostName)
{
    if(!empty($hostName)) {
        exec("nslookup -type=$recType $hostName", $result);
        // On vérifie toutes les lignes pour trouver celle qui commence
        // par le nom de l'hôte
        foreach ($result as $line) {
            if(ereg("^\$hostName",$line)) {
                return true;
            }
        }
    }
}
```

```

        return false;
    }
    return false;
}
echo myCheckDNSRR("dreams4net.com");
?>

```

**Remarque**

La fonction `checkdnsrr()` peut permettre d'optimiser une validation d'adresse électronique en vérifiant que le nom de domaine de l'adresse est bien valide.

**Corrélation IP/DNS**

La fonction `dns_get_record()` lit les données DNS associées à un hôte passé en paramètre :

```
dns_get_record ( hôte)
```

**Note**

La fonction n'est pas implémentée sur les plates-formes Windows. Il est possible d'utiliser la classe PEAR `Net_DNS` en remplacement.

La fonction renvoie un tableau associatif, contenant au minimum les index `host`, `type`, `class` et `ttl`. Leur description est donnée au tableau 7-3.

**Tableau 7-3 Les paramètres renvoyés par `dns_get_record()`**

Attribut	Signification
host	L'enregistrement de l'espace de noms DNS qui est décrit par les autres données.
class	Classe d'enregistrement Internet. En tant que tel, cet index vaudra toujours IN.
type	Chaîne de caractères contenant le type d'enregistrement.
ttl	<i>Time To Live</i> : durée avant expiration de l'enregistrement.

Le résultat du script suivant est donné en figure 7-9.

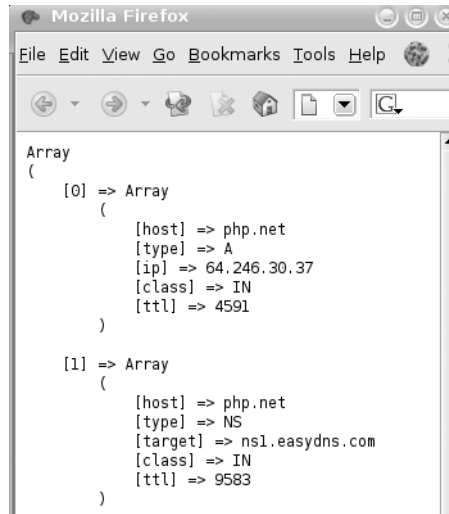
```

<?php
$result = dns_get_record("php.net");
echo '<pre>' ;
print_r($result);
echo '</pre>' ;
?>

```

Figure 7-9

Utilisation de  
`dns_get_record()`



La fonction inverse est la fonction `gethostbyaddr()`, qui renvoie le nom d'hôte correspondant à une adresse IP.

```
<?php
$hote = gethostbyaddr("217.174.203.51");
print $hote;
?>
```

## Fonctions de chiffrement

Toute application produisant des flux de données confidentielles implique une sécurisation de ceux-ci pour éviter qu'un tiers puisse les consulter. La solution optimale est de chiffrer les données à sécuriser empêchant ainsi leur utilisation en cas d'interception.

PHP met à votre disposition dans ce but un module implémentant la plupart des méthodes de codage existantes. Pour l'installer, il vous faudra compiler PHP avec `--with-mcrypt` ou décommenter l'extension correspondante dans votre `php.ini` si vous utilisez Windows.

Nous nous contenterons de décrire ici quelques fonctions qui sont accessibles par défaut, sans l'extension.

### *Quelques définitions : chiffrement, hachage, codage/décodage*

Tout d'abord, il est nécessaire de préciser les termes. Sous le terme de chiffrement, nous distinguerons en fait deux termes plus précis et plus corrects : hachage et codage (et décodage).

Le procédé de hachage est une opération à sens unique qui permet d'obtenir une chaîne non prédictible et qui semble aléatoire à partir d'un texte ou d'une donnée. La chaîne résultante est généralement assez courte. Comme il s'agit d'une opération à sens unique, personne ne pourra décoder la chaîne résultante et revenir à la donnée d'origine, pas même vous qui avez fait la première conversion. Nous verrons plus loin l'utilité d'une telle technique.

Les procédés de codage et décodage permettent, eux, d'avoir une opération à double sens. Une fois la chaîne transformée, il est possible de revenir à la donnée d'origine, que ce soit avec un mot de passe ou grâce à un système plus complexe.

## *Fonctions de hachage*

### **Buts et utilité**

Les fonctions de hachage fonctionnent à sens unique. Il vous sera impossible de revenir en arrière, sauf à essayer toutes les possibilités une à une. Il y a trois applications classiques à ces fonctions.

L'utilisation la plus classique, et la plus fréquente, concerne les mots de passe. Une pratique malheureusement répandue consiste à stocker les mots de passe en clair ou de manière décodable. La première faille de sécurité de votre application entraînerait la divulgation de tous les accès. Plutôt que de stocker un mot de passe, il est possible de stocker un hachage du mot de passe à la place de celui-ci. Pour identifier une personne, il suffit de refaire le même traitement de hachage sur le mot de passe fourni et de vérifier que le résultat est le même que celui qu'on avait stocké auparavant. On arrive alors à identifier quelqu'un sans avoir stocké son mot de passe réel.

Une autre application concerne l'identification. Les chaînes hachées sont généralement courtes (32 caractères pour le hachage MD5 par exemple). Elles servent donc souvent d'identifiant.

Enfin, on peut utiliser le hachage pour vérifier l'intégrité d'un fichier transféré. Pour cela, l'auteur peut lui adjoindre un hachage court. À la réception, il suffit de vérifier la concordance entre la chaîne de hachage et le fichier ; si les deux ne correspondent pas, c'est que l'un des deux a été mal téléchargé.

### **CRC32**

La méthode de hachage CRC32 est un procédé très simple et très rapide. Son résultat peut toutefois être facilement prévisible, c'est-à-dire qu'il est facile de générer une donnée à partir d'un code CRC. Cette donnée ne sera pas celle d'origine, mais elle renverra le même code de hachage.

En conséquence, cette méthode peut être utilisée pour vérifier la présence d'erreur dans une transmission, mais sera bien trop sensible pour des fonctionnalités liées à la sécurité.

Ce hachage est particulièrement utilisé dans les systèmes de fichiers pour détecter les erreurs d'écriture sur le disque.

#### Utilisation de CRC32 dans les systèmes de fichiers

Les écritures sur le disque sont sujettes aux erreurs. Le système cherche donc en permanence, quand il relit une information, à savoir si elle a été correctement relue (et s'il peut s'y fier) ou si elle comporte un défaut (et le cas échéant essayer de le corriger). En pratique, le système va stocker avec chaque donnée une série de bits représentant un contrôle de parité des quelques bits précédents.

La fonction PHP se nomme `crc32()`. Il suffit de lui donner en argument une chaîne de caractères ou des données binaires et elle renvoie un entier (voir figure 7-10).

```
<?php
$fichier = 'test2.php';
$crc = crc32( file_get_contents($fichier) );
echo $crc;
?>
```

Figure 7-10

Exemple de CRC  
calculé



L'entier renvoyé est normalement un entier non signé. PHP gère, lui, des entiers signés. Si vous comptez échanger le résultat avec un autre programme ou l'afficher, il vous faudra donc le convertir. Vous pouvez par exemple le faire avec la fonction `sprintf()` et le paramètre `%u`.

```
<?php
$fichier = 'phpinfo.php';
$crc_PHP = crc32( file_get_contents($fichier) );
echo sprintf("%u", $crc_PHP);
?>
```

#### MD5

Le hachage MD5 est nettement plus complexe que le CRC32. Son résultat est jugé imprédictible : il est impossible de construire volontairement une donnée qui donne un MD5 spécifié (sauf si on veut tester toutes les possibilités une à une). De plus, la chaîne

résultante est composée de 32 caractères, ce qui commence à être assez long pour garantir une probabilité d'unicité assez importante (avoir à un instant T deux données avec le même résultat MD5 dans une application est très hautement improbable, assez pour ne pas en tenir compte sur des volumes raisonnables).

Un système basé sur cet algorithme est désormais souvent utilisé pour gérer les mots de passe système (c'est le cas dans les Unix et Linux récents). On le trouve aussi souvent pour vérifier l'intégrité des téléchargements.

Vous pouvez obtenir un MD5 en passant une donnée en paramètre à la fonction PHP `md5()`. Un exemple de rendu est donné à la figure 7-11.

```
<?php
$donnee = 'secretXzB';
$md5 = md5( $donnee );
echo $md5 ;
?>
```

**Figure 7-11**

*Exemple de MD5  
calculé*



Le résultat est une chaîne hexadécimale (donc facilement manipulable et affichable) de 32 caractères. En passant la valeur `TRUE` comme second paramètre, PHP renvoie une valeur binaire de 16 octets à la place (mais la gestion par des humains de ce hachage est alors délicate puisqu'il n'est pas forcément lisible).

Comme MD5 est souvent utilisé pour vérifier l'intégrité de fichiers, PHP implémente une fonction `md5_file()`, qui calcule directement la somme md5 d'un fichier :

```
$somme_md5 = md5_file($fichier) ;
```

## SHA1

Le SHA1 est similaire au MD5 mais renvoie une chaîne légèrement plus longue (40 caractères hexadécimaux, ou 20 octets). La probabilité d'une collision (deux valeurs avec le même résultat) est donc plus rare ; plus exactement, pour deux données précisées, il y a une chance sur  $2^{20}$  qu'elles aient le même résultat. En conséquence, ce procédé de hachage est aussi légèrement plus lent.

Les fonctions `sha1()` et `sha1_file()` sont par ailleurs en tout point similaires à `md5()` et `md5_file()`.

```
■ $sha1 = sha1_file($fichier) ;
```

## Crypt

`crypt` est le nom de la fonction classique sur Unix qui permet d'authentifier quelqu'un avec un hachage dérivé du codage DES. Ce procédé est de moins en moins utilisé sur les systèmes récents car l'algorithme permet de trouver dans des temps désormais accessibles un mot de passe à partir de son hachage. Il reste toutefois largement employé, par exemple pour gérer les authentifications dans les configurations Apache.

```
■ $hash = crypt( $donnee ) ;
```

La fonction `crypt()` n'est pas une application directe du hachage ; elle implémente aussi une partie aléatoire, pour rendre la prédiction plus difficile à faire. Il est possible de fournir la donnée aléatoire via un second paramètre. Elle est classiquement de deux caractères, les caractères surnuméraires sont ignorés. Ces deux caractères sont utilisés pour produire un résultat. Ainsi, si on fait deux conversions d'une même donnée avec `crypt()` en utilisant des données aléatoires différentes, on obtiendra des résultats différents.

```
■ $hash = crypt( $donnee , $aleatoire ) ;
```

Si vous ne spécifiez pas la partie aléatoire, PHP la calcule lui-même. Il est important de noter que PHP ne calcule lui-même la partie aléatoire qu'une seule fois par exécution ; les autres appels à `crypt()` utilisent la même valeur, ce qui peut se révéler gênant pour la sécurité. Si vous traitez des données en masse, vous vous devez donc de fournir vous-même la partie aléatoire.

### Utilisation pour une authentification

Il a été dit que `crypt()` servait surtout à vérifier des mots de passe. Pour faire cette vérification, il faut pouvoir connaître la valeur aléatoire utilisée (sinon on obtiendrait à chaque fois un résultat différent).

#### Note

En fait, la fonction `crypt()` renvoie toujours la chaîne aléatoire dans la valeur retournée, en premiers caractères. Pour vérifier un mot de passe, il suffit alors de spécifier le hachage sauvegardé comme valeur aléatoire.

La première phase est de chiffrer le mot de passe soumis par l'utilisateur lors de son inscription :

```
■ // Stockage du mot de passe initial
  $hachage_stocke = crypt($pass_initial) ;
```

Lorsque l'utilisateur revient et souhaite s'identifier, on récupère le *hash* stocké. Cette ancienne chaîne cryptée sert de paramètre pour chiffrer le nouveau mot de passe (afin que



les deux chaînes aléatoires soient les mêmes). Il suffit alors de comparer l'ancienne chaîne cryptée et la nouvelle. Si elles sont identiques, c'est que le mot de passe est bon :

```
// Récupération du mot de passe fourni par l'utilisateur
$pass_fourni = $_REQUEST['password'] ;
// Vérification plus tard avec un mot de passe fourni
if ( $hachage_stocke == crypt($pass_fourni, $hachage_stocke) ) {
    echo "mot de passe valide" ;
} else {
    echo "mot de passe invalide" ;
}
```

### Utiliser différents algorithmes

La fonction `crypt()` utilise classiquement un algorithme dérivé de DES. Sur les systèmes récents, `crypt()` sait aussi se servir d'une base MD5 ou Blowfish, pour plus de sécurité.

Sur les systèmes qui le permettent, vous pouvez utiliser `md5` comme algorithme interne en spécifiant une valeur aléatoire de douze caractères préfixée par `$1$`. Si votre système accepte cette méthode, la constante `CRYPT_MD5` contiendra la valeur `TRUE`.

Vous pouvez aussi utiliser Blowfish en spécifiant une valeur aléatoire de seize caractères préfixée par `$2$`. Si votre système accepte ce type de hachage, alors la constante `CRYPT_BLOWFISH` aura une valeur vraie.

## Fonctions de codage et décodage

Les fonctions de codage et décodage se séparent elles-mêmes en deux groupes : les codages à clé publique et les codages à clé privée. Les premiers désignent les codages qui emploient la même clé (ou le même mot de passe) pour coder et décoder la donnée. Les autres permettent d'avoir un mot-clé pour coder et un pour décoder. Généralement, on en réserve un pour soi et on diffuse l'autre.

Décrire les fonctions de codage et décodage demanderait de s'étendre longuement sur les algorithmes et les procédés. Nous nous contenterons de vous signaler la présence du module `mcrypt` dans PHP (documentation à l'adresse <http://fr.php.net/mcrypt>), qui gère pratiquement tous les algorithmes et les méthodes de codage qui sont rencontrées le plus souvent.

Les principales applications de codage que vous risquez de rencontrer dans un contexte classique sont celles des transmissions sécurisées par SSL. Ces flux sont automatiquement gérés par PHP quand on utilise le préfixe `https://` pour des URL ou le préfixe `ssl://` pour les *sockets* réseau. Les communications sont alors totalement transparentes et vous n'aurez pas à vous en préoccuper.

#### Note

Quand un visiteur accède à vos pages par une connexion sécurisée (adresse commençant par `https://`), c'est Apache qui gère la communication. PHP n'intervient pas dans le codage. Il vous suffit de gérer vos scripts comme pour une page normale.

## Exécution de code

### Fonction à l'arrêt du script

PHP exécute automatiquement certaines actions à la clôture du script ; la fermeture des fichiers encore ouverts est l'exemple le plus connu. Vous pouvez vous-même enregistrer du code pour qu'il soit exécuté juste avant l'arrêt du script.

Si vous fournissez un nom de fonction à `register_shutdown_function()`, PHP l'enregistre et l'exécute juste avant de terminer le script. Vous pouvez faire autant d'enregistrements que vous souhaitez, PHP les traite dans l'ordre. La fonction n'est appelée à la fin de l'exécution que si elle se termine normalement. Une erreur fatale, un appel à `exit()` ou `die()` arrête brutalement le script, sans appeler les fonctions enregistrées.

Cette fonctionnalité est prévue pour vous permettre des opérations de nettoyage ou de statistiques, un peu comme la fermeture des fichiers ouverts. Elle n'est en particulier pas faite pour afficher quelque chose sur la sortie standard (vers le navigateur). Les envois de textes vers la sortie ne fonctionnent pas.

#### Note

En fait, sur certaines installations, les fonctions ainsi exécutées peuvent tout à fait renvoyer des données vers la sortie. Il s'agit cependant d'un paramètre dépendant de votre configuration et de votre plate-forme. Il ne doit pas être utilisé car il n'est pas portable et peut changer dans une future version de PHP.

On peut par exemple imaginer faire des statistiques sur vos scripts et leur temps d'exécution :

```
<?php
// On retient l'heure de démarrage du script
$time = time() ;
// On définit une fonction qui enregistrera le temps écoulé
// dans un fichier de statistiques
function statistiques() {
    global $time ;
    $fp = fopen('stats.txt', 'a') ;
    flock($fp, LOCK_EX) ;
    $secondes = time() - $time ;
    fputs($fp, "exécution de $secondes secondes\n" ;
    fclose($fp) ;
}
// Enregistrement de la fonction pour qu'elle s'exécute à la fin
register_shutdown_function('statistiques') ;

/* Mettez ici votre script normal
   son temps d'exécution sera calculé
   et enregistré dans stats.txt      */
```

**Utilisation de la programmation orientée objet**

Comme à chaque fois que PHP vous demande un nom de fonction à exécuter, vous pouvez utiliser à la place une méthode d'objet. Il vous faut alors fournir un tableau qui contient un objet en premier argument et un nom de méthode en second.

## Exécution d'une chaîne de code PHP

Normalement, le code PHP est une partie fixe de votre applicatif, il ne devrait pas changer en cours d'exécution. Seules les données utilisées et échangées devraient constituer la partie « dynamique ».

Il peut toutefois être utile de pouvoir produire dynamiquement du code PHP et l'exécuter dans la foulée. La fonction `eval()` peut alors vous être utile. Elle prend en paramètre une chaîne de caractères qui contient du code PHP et l'exécute. Si votre code contient une instruction `return`, l'exécution du code inséré s'arrête et `eval()` retourne la valeur en question.

```
<?php
$code = " \$a++ ; \$a .= \"texte\" ; return \$a" ;
$a = 1 ;
$a = eval($code)
echo $a ; // Affiche 2texte Cas d'application
```

**Note**

Le texte à exécuter doit être dans une chaîne de caractères classique. Si vous définissez cette chaîne avec PHP, il est probable que vous deviez échapper les caractères comme les préfixes de variable ou les guillemets afin qu'ils ne soient pas interprétés par PHP lors de la définition de la chaîne.

## Login/mot de passe sécurisés

**Contexte**

Votre application gère des comptes utilisateurs pour vos clients afin qu'ils puissent accéder à certaines informations via le site web public. Dernièrement, une faille a été exploitée sur votre site web. Aucune information confidentielle n'a été dévoilée ou corrompue, mais le visiteur a semblé-t-il pu lire le fichier de mots de passe. Faire changer le mot de passe de vos différents clients s'est avéré délicat et vous voudriez éviter cette situation embarrassante à l'avenir.

**Solution retenue**

Pour qu'une éventuelle faille sur votre applicatif ne devienne pas exagérément gênante, il vous faut protéger l'accès à vos données sensibles : dans notre cas, aux mots de passe. Pour éviter tout problème, il a été décidé que les mots de passe ne seront plus stockés ni en clair ni sous une forme décodable.

La fonction employée pour brouiller les mots de passe sera `crypt()`. Elle est standard sur les systèmes Unix et garantit l'interopérabilité de la base des mots de passe avec des futurs logiciels.

Les accès au système de mots de passe sont basés sur deux fonctions qu'il vous appartient d'implémenter :

- `getPassword($utilisateur)` : récupère le mot de passe (brouillé) associé à un utilisateur ;
- `setPassword($utilisateur, $passe)` : définit un mot de passe pour l'utilisateur.

### Réalisation

Ces fonctions peuvent par exemple être implémentées avec le code suivant (basé sur un *backend* avec SGBD MySQL) :

```
function getPassword($utilisateur) {
    $uid = addslashes( $utilisateur );
    $requete = "SELECT password FROM utilisateurs WHERE uid='$uid'";
    $res = mysql_query($requete);
    if (mysql_num_rows($res)) {
        return mysql_result($res, 0, 0);
    } else {
        return FALSE;
    }
}

function setPassword($utilisateur, $pass) {
    $uid = addslashes( $utilisateur );
    $pass = addslashes( $pass );
    $req = "REPLACE utilisateurs "
        . "SET uid='$uid', password = '$pass'";
    return mysql_query($req);
}
```

Les accès bas niveau faits, il reste à écrire les deux fonctions qui vont nous permettre d'ajouter ou de modifier un mot de passe et de vérifier l'accès d'un utilisateur.

La première fonction est celle qui modifie ou ajoute un mot de passe. Elle se contente de prendre en paramètre un identifiant utilisateur et un mot de passe, de chiffrer le mot de passe puis le passer en paramètre à `setPassword()` afin qu'il soit enregistré dans la base de données.

```
function newPassword($utilisateur, $pass) {
    $crypt = crypt($pass);
    return setPassword($utilisateur, $crypt);
}
```

La deuxième fonction est celle qui permet de vérifier un mot de passe en le comparant à celui stocké dans la base de données :

```
function checkPassword($utilisateur, $pass) {
    if (empty($pass)) return FALSE; // Pas de mot de passe vide
    $interne = getPassword($utilisateur); // On récupère l'ancien
    $crypt = crypt($pass, $interne); // On crypte le nouveau
```

```
// L'ancien sert de valeur aléatoire
// car la valeur aléatoire utilisée la première fois par PHP
// est donnée dans les deux premiers caractères
// (les deux seuls utilisés)
return ($interne === $crypt) ; // on vérifie la correspondance
}
```

Plus aucun mot de passe n'est maintenant stocké en clair. En cas de problèmes, il suffira de corriger la faille utilisée et de restaurer les données corrompues. Le pirate ne pourra pas utiliser les mots de passe lus pour revenir ; il n'y aura donc plus besoin de faire changer ces mots de passe par vos clients.

# Formulaires et superglobales

---

Jusqu'ici, nous n'avons utilisé que des variables que nous avons définies nous-même, ou des données retournées par des fonctions de PHP. Nous avons pu voir comment les affecter, les modifier et agir en fonction de leur valeur, mais nous n'avons pas encore utilisé de données envoyées par l'utilisateur.

Dans un contexte Web, ces données sont généralement envoyées via des formulaires. C'est l'un des points importants dans la réalisation d'un site web dynamique, mais c'est également l'un des aspects nécessitant le plus de vigilance. C'est dans ce contexte qu'ont été introduites les superglobales. Ces supervariables permettent, entre autres, de manipuler les données transmises par les formulaires. Nous verrons donc comment gérer la transmission d'informations textuelles et de fichiers du client vers le serveur. Nous traiterons ensuite des différents points auxquels il faut prêter attention pour assurer et sécuriser son application.

## Formulaires HTML

Les formulaires HTML sont la méthode la plus simple pour avoir des interactions avancées avec l'utilisateur. Ils vous permettront par exemple de :

- créer un espace sécurisé ;
- donner la possibilité à vos clients de modifier eux-mêmes le contenu de leur site ;
- interagir avec le visiteur en lui demandant des informations complémentaires ;
- réaliser une page de recherche.

**Attention**

Les formulaires doivent cependant être maniés avec de grandes précautions. En acceptant et en utilisant des données venant de l'utilisateur, vous ouvrez une porte par laquelle des gens mal intentionnés pourraient essayer d'exploiter des failles de sécurité. Vous vous exposez de plus à de nombreux comportements bogués si vous n'avez pas prévu ce que l'utilisateur vous envoie.

Nous allons dans un premier temps aborder les bases de la gestion des formulaires en HTML, puisqu'ils constituent la méthode la plus courante d'envoi d'informations par l'utilisateur. Nous présenterons ensuite les deux méthodes permettant de faire circuler ces informations. Nous verrons pour chacune d'elles ses avantages et ses inconvénients. Nous rentrerons alors dans le vif du sujet en voyant comment PHP peut, via les super-globales de formulaires, récupérer ces valeurs et les traiter.

### ***Nouveautés depuis PHP 4.0***

Avant la version PHP 4.2.0, la valeur par défaut du paramètre `register_globals`, dans le fichier de configuration `php.ini`, était à `0n`. On pouvait alors référencer la valeur `login` reçue par un formulaire avec la variable globale `$login`.

Mal employée, cette fonctionnalité facilitait l'apparition de problèmes de sécurité, puisque le visiteur pouvait injecter directement des variables dans l'espace du développeur.

Depuis PHP 4.2.0, cette fonctionnalité est désactivée par défaut. Avec `register_globals` désactivé, une variable envoyée par un formulaire sous le nom `login` ne sera plus accessible directement avec `$login`. Pour relire ce champ, il faudra alors utiliser la variable `$_REQUEST['login']` (`$_GET['login']` ou `$_POST['login']` renverra le même résultat suivant la méthode employée pour envoyer le formulaire).

Désormais, votre code sera plus sûr, car vous savez d'où provient chacune de ses variables. Vous pouvez trouver plus d'informations sur le site officiel de PHP, à l'adresse <http://www.php.net/manual/fr/security.registerglobals.php>.

## **Caractères spéciaux et HTML**

Le HTML contient quelques caractères spéciaux avec une signification particulière. Les caractères d'ouverture et de fermeture des balises (`<` et `>`) ainsi que le caractère de début d'entité (`&`) sont les trois plus importants. Si vous envoyez dans votre page HTML des chaînes de texte sans transformer ces caractères, vous pourriez rendre votre page invalide ou la rendre illisible pour le visiteur.

De la même manière, si vous remplissez un attribut délimité par des guillemets (ou apostrophes) et si la chaîne que vous y insérez en contient aussi, vous risquez de rendre votre donnée illisible par le navigateur.

Pour gérer ces cas, vous devrez faire appel à la fonction `htmlspecialchars()` pour toute donnée que vous comptez envoyer vers la page HTML. Elle transformera les caractères interprétables de façon qu'ils soient affichés au lieu d'être analysés par le navigateur.

```
// Sans htmlspecialchars, l'affichage ne marcherait pas
echo htmlspecialchars("L'affirmation 1<2 est vraie, pas 1>2");
```

Si vous utilisez une chaîne dans un attribut HTML, vous devrez passer `ENT_QUOTES` en second paramètre pour que les guillemets et apostrophes soient aussi convertis.

```
$default = "c'est moi" ;
$default = htmlspecialchars($default, ENT_QUOTES) ;
echo "<input type='text' value='$default'" ;
```

Vous trouverez plus d'informations à ce sujet au chapitre 27, consacré à la sécurité.

#### Note sur les exemples de ce chapitre

Pour ne pas alourdir systématiquement les exemples et se concentrer à chaque fois sur un point précis, nous n'utiliserons pas systématiquement l'échappement. Ces exemples ne sont toutefois là que pour votre compréhension ; sur une application réelle, l'utilisation de `htmlspecialchars()` est nécessaire.

## Création du formulaire

Dans cette sous-partie, nous allons détailler brièvement les différentes notions de HTML qui vous seront nécessaires pour utiliser les formulaires. Si HTML est pour vous un terrain largement connu, vous pouvez lire rapidement ces quelques pages en ne vous arrêtant que sur les astuces PHP et passer à la description de l'utilisation des formulaires à la réception.

### Déclaration d'un formulaire

Un formulaire HTML est une zone dans laquelle vous insérez des champs et des contrôles que le visiteur manipulera via son navigateur. On peut y définir des champs de saisie de texte, des cases à cocher, des listes d'options, des champs cachés, des mots de passe, des fichiers à envoyer, etc.

#### Balise FORM

Les balises `<form>` et `</form>` permettent d'indiquer le début et la fin d'un formulaire. Ainsi, dans votre page HTML, vous aurez le code suivant :

```
<form>
<!--
Différents champs et balises de données.
-->
</form>
```



Cela ne suffit cependant pas à indiquer à qui envoyer les informations, ni par quelle méthode les traiter. L'attribut `action` désigne la page vers laquelle seront envoyées les informations rentrées par l'utilisateur une fois le bouton d'envoi actionné. C'est cette page de destination qui traitera les données du formulaire envoyées par le visiteur.

Dans le cadre d'une utilisation avec PHP, on aurait le code suivant :

```
<form action='reception_formulaire.php'>
<!--
Différents champs et balises de données.
-->
</form>
```

Quand nous traiterons de l'envoi de fichiers, nous verrons qu'il est parfois nécessaire de modifier l'attribut `enctype` désignant le type de codage utilisé.

## Méthode d'envoi du formulaire

L'attribut `method` de la balise `<form>` définit le mode d'envoi des informations au serveur. Deux méthodes existent : la méthode GET et la méthode POST.

### Méthode GET (transmission par URL)

Une des façons les plus simples pour faire transiter des données de page en page est l'insertion de ces données dans l'URL (ce qui est affiché dans la barre d'adresse du navigateur). Ce type d'envoi passe normalement par une requête HTTP de type GET ; on nomme donc communément les données ajoutées dans l'URL sous le nom de « chaîne de GET ».

### Création manuelle des URL

Des adresses contenant des paramètres GET peuvent être créées manuellement assez facilement. Le lien suivant insère par exemple des valeurs de nom et prénom. La syntaxe à utiliser consiste à séparer le fichier à appeler des variables à transmettre par un point d'interrogation. Ensuite, on définit pour chaque variable à envoyer un couple composé du nom de la variable et d'un signe égal = suivi du contenu de la variable. On sépare chaque couple par le symbole &. Le contenu de la donnée doit lui-même être converti avec la fonction `urlencode()` pour éviter la présence de caractères spéciaux.

```
<?php
$nom = urlencode("daspet") ;
$prenom = urlencode("éric") ;
$lien = "script.php?nom=$nom&prenom=$prenom";
echo "<a href='$lien'>Lien</a>";
// Affiche <a href='script.php?nom=daspet&prenom=%E9ric'>Lien</a>
?>
```

Dans le cas de formulaires, une URL de ce genre est automatiquement construite par le navigateur en fonction des données à envoyer quand vous définissez le type GET comme valeur du champ `method` de votre formulaire.

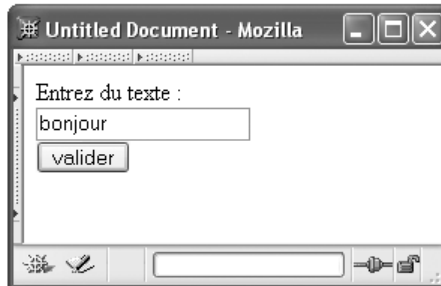
## Méthode POST

L'autre méthode possible pour l'envoi d'un formulaire HTML est la méthode POST :

```
<form action="script.php" method="POST"><p>
Entrez du texte :<BR>
<input name="variable" type="text"><br>
<input type="submit" value="valider">
</p></form>
```

Figure 8-1

Exemple de  
formulaire



Quand un utilisateur clique sur le bouton Valider, son navigateur soumet le formulaire au script `script.php`. On peut récupérer les valeurs via PHP dans ce dernier script.

### Quelle méthode utiliser ?

Le plus souvent, vous pouvez choisir indifféremment l'une ou l'autre méthode d'envoi. Toutefois, la méthode POST s'impose dans les cas suivants :

- envoi d'un fichier ;
- envoi de données importantes en taille ;
- envoi de données confidentielles (un mot de passe par exemple) ;
- si le formulaire déclenche une action spécifique qui doit être renouvelée à chaque fois (modération d'un article par exemple).

La méthode GET est recommandée pour tous les autres cas. Elle est adaptée à un comportement d'affichage uniquement lorsque le résultat peut être mis en cache (recherche sur un site, affichage d'un article défini, etc.) et ne subit pas certaines limitations de la méthode POST, notamment le fait de ne pouvoir revenir en arrière sans re-soumettre le formulaire. Cette méthode est aussi souvent utilisée de manière statique, sans recours aux formulaires, pour faire passer un paramètre à un script via l'URL.

## Champ de texte

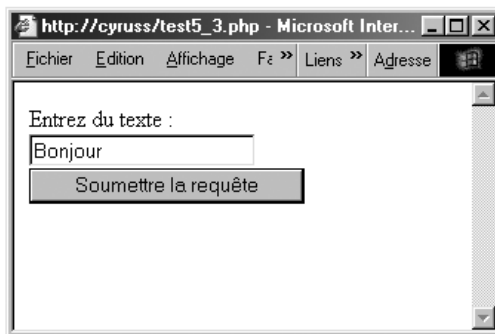
Les champs de saisie de texte sont probablement les balises les plus utilisées dans des formulaires. Ils sont créés à partir de la balise `<input>` en définissant l'attribut `type`

comme étant du texte (`type="text"`). On utilise ce type de champ quand on a besoin de recueillir de la part de l'utilisateur des mots ou des textes de taille limitée.

L'exemple suivant illustre l'utilisation de ce genre de champ ; une capture d'écran du rendu est présentée à la figure 8-2.

```
<form action="form.php" method="POST"><p>
Entrez du texte : <br>
<input type="text" name="champ1"><br>
<input type="Submit" value="Soumettre la requête">
</p></form>
```

**Figure 8-2**  
*Champ texte*



### Valeur par défaut

Il est possible de définir une valeur par défaut grâce à l'attribut `value`. Si par exemple vous disposez d'un espace d'administration où il est possible de modifier le titre d'un article, vous afficherez la valeur actuelle avant de laisser l'utilisateur la changer éventuellement.

```
Votre nom:<br>
<input type="text" name="nom" value="Daspet">
```

#### Attention

Comme à chaque fois que vous utilisez du texte dans une page HTML, s'il peut contenir des caractères spéciaux (délimitation de balise HTML, d'attribut, caractère `&`, etc.) vous devez l'échapper avec la fonction `htmlspecialchars()`.

### Taille du champ

La taille du champ de saisie peut aussi être contrôlée. On distingue alors deux cas : on contrôle la taille de la donnée ou celle de la boîte de saisie.

La taille de la boîte de saisie peut être définie via l'attribut `size`. On peut par exemple avoir une boîte qui s'adapte exactement à la taille de son contenu par défaut :

```
<?php
$t = 'Nouvelle hausse du pétrole !';
```

```
// En général ce titre est récupéré dans une base de données
$s = strlen($t);
//Ici, on détermine dynamiquement la taille de la chaîne
echo '<input type="text" name="t" value="', $t, '" size="', $s, '">';
?>
```

Il est aussi possible de limiter la taille de la donnée via l'attribut `maxlength`. Vous éviterez ainsi des erreurs d'utilisateur. Limiter à cinq caractères la saisie d'un code postal peut aider l'utilisateur à repérer une faute de frappe. Limiter un champ de mot de passe peut éviter à l'utilisateur qui n'a pas lu les instructions d'en fournir un trop long qui ne sera pas pris en compte.

```
<input type="text" name="login" maxlength="8" value="eda">
```

#### Attention

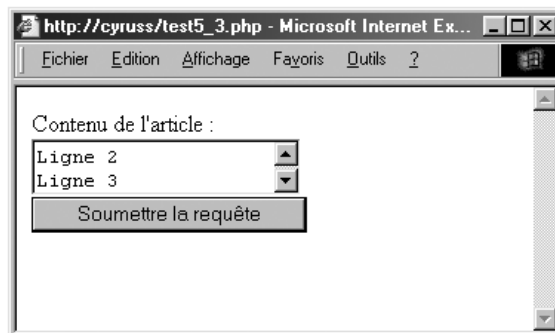
La taille de la donnée n'est qu'une valeur informative qui sert d'aide pour le navigateur et le visiteur. Elle n'empêche pas quelqu'un de malveillant d'envoyer volontairement une donnée trop grande et ne vous dispense pas d'un tel contrôle dans vos scripts de gestion.

## Zone de texte

Nous venons de voir comment gérer un champ texte. Celui-ci est suffisant pour de petites phrases contenant au maximum une vingtaine de mots. Il est souvent nécessaire de disposer d'un affichage permettant de caler une plus grande zone de texte. Pour cela, on utilise des zones de texte délimitées par les balises `<textarea></textarea>`. Un exemple de zone de texte telle qu'affichée par un navigateur est donné à la figure 8-3.

```
<form action="./form.php" method="POST"><p>
Contenu de l'article :<br>
<textarea name="texte_long"></textarea>
<input type="submit" value="Soumettre la requête">
</p></form>
```

Figure 8-3  
Zone de texte



### Taille de la zone de texte

Il est possible de définir la largeur et la hauteur de la zone de texte avec les attributs `rows` et `cols` (respectivement le nombre de lignes et le nombre de colonnes à afficher).

```
Contenu de l'article :<br>
<textarea name="t" cols="50" rows="10"></textarea>
```

### Valeur par défaut

De la même façon que pour les champs de texte, il est possible de définir une valeur par défaut. Il suffit de la placer entre la balise d'ouverture et la balise de fermeture :

```
Contenu de l'article :<br>
<textarea name="t" cols="50" rows="10">
PHP GTK ?
Une combinaison entre l'outil le plus utilisé pour l'Internet
dynamique et GTK, une bibliothèque graphique très connue
chez les utilisateurs de Linux, pourquoi faire ?
</textarea>
```

### Cases à cocher

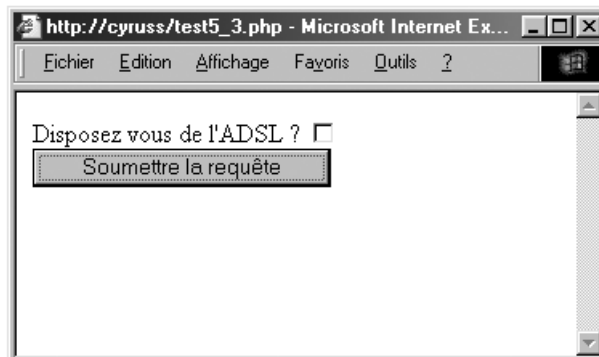
Les cases permettent d'interagir avec vos utilisateurs en définissant vous-même les réponses possibles. L'interaction avec le visiteur est moindre mais permet de faciliter le traitement informatique des réponses.

Ces cases à cocher, de la même façon que les champs de texte, sont créées à l'aide de la balise `<input>`. C'est l'attribut `type` qui, au lieu de prendre la valeur `text`, prend la valeur `checkbox`. Un exemple de rendu dans un navigateur est donné à la figure 8-4.

```
<form action="./form.php" method="POST"></p>
Disposez vous de l'ADSL ?
<input type="checkbox" name="qcm1"><br>
<input type="Submit" value="Soumettre la requête">
</form>
```

Figure 8-4

Cases à cocher



### Valeur associée à une case

Par défaut, les cases à cocher renvoient la valeur 0n quand elles sont cochées. Si ce n'est pas le cas, aucune valeur n'est envoyée (comme si la case n'était pas présente dans le formulaire). On peut cependant changer la valeur associée à une case à cocher via l'attribut value.

```
<input type="checkbox" name="qcm1" value="oui"><br>
```

### État par défaut

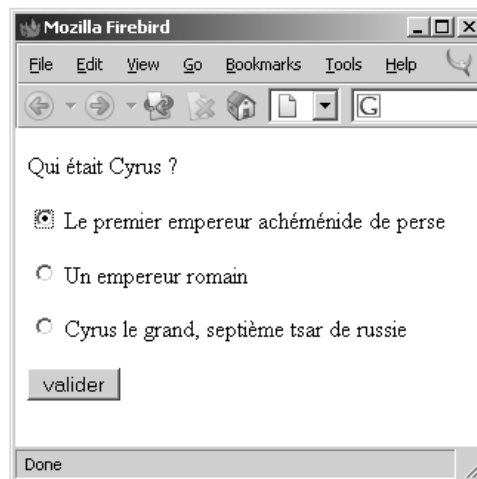
Il est possible de définir qu'une case soit cochée par défaut en ajoutant le paramètre checked dans la balise input :

```
<input type="checkbox" name="qcm1" checked>
```

## Bouton radio

Les boutons radio sont assez semblables aux cases à cocher, si ce n'est qu'ils ne permettent qu'un seul choix sur un ensemble. L'application la plus directe est la création d'un questionnaire à choix multiples (voir figure 8-5).

Figure 8-5  
Boutons radio



Les boutons radio sont créés avec la balise `<input>` en définissant le type radio :

```
<input type="radio" name="radio1">
```

Comme les cases à cocher, les boutons radio possèdent l'attribut checked permettant de les cocher par défaut.

```
<input type="radio" name="radio1" checked>
```

Le nom identique des différentes balises radio indique au serveur web que les trois boutons sont liés. Pour réaliser l'exemple de la figure 8-5, nous aurions le code suivant :

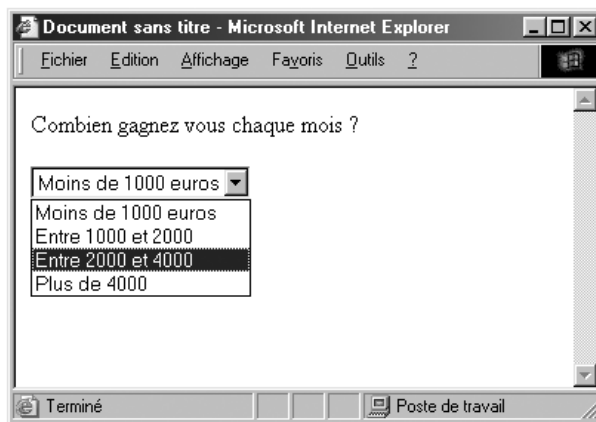
```
<form action="cases.php" method="POST"><p>
Qui était Cyrus ?<br>
<input type="Radio" name="radio1" value=1> Le premier ...
<input type="Radio" name="radio1" value=2> Un empereur ...
<input type="Radio" name="radio1" value=3> Cyrus le ...
<input type="submit" value="valider">
</p></form>
```

### Liste de sélections et liste déroulante

La liste déroulante permet d'afficher en un espace réduit une liste d'éléments. Un exemple vous est donné à la figure 8-6.

Figure 8-6

Liste déroulante



En HTML, on indique que la liste déroulante commence avec la balise `<select>` :

```
<select name= "salaire">
<!--Ici les différentes options possibles-->
</select>
```

Entre les balises d'ouverture et de fermeture, on indiquera les différentes options possibles. Pour cela, on utilisera la balise `<option>` en lui donnant éventuellement une valeur avec l'attribut `value` (sinon le résultat sera le texte contenu entre les balises d'ouverture et de fermeture).

L'exemple de la figure 8-6 peut être généré par le code suivant :

```
<select name= "salaire">
<!--Ici les différentes options possibles-->
<option value="1">Moins de 1000 euros</option>
<option value="2">Entre 1000 et 2000</option>
```

```
<option value="3">Entre 2000 et 4000</option>
<option value="4">Plus de 4000</option>
</select>
```

### Choix par défaut

Par défaut, aucun choix n'est sélectionné, ou bien c'est le premier. On peut bien sûr remédier à cela en ajoutant `selected` dans la balise `<option>` qui est sélectionnée par défaut. Comme pour les autres champs de formulaire, cela nous permettra de cocher dynamiquement l'option sélectionnée.

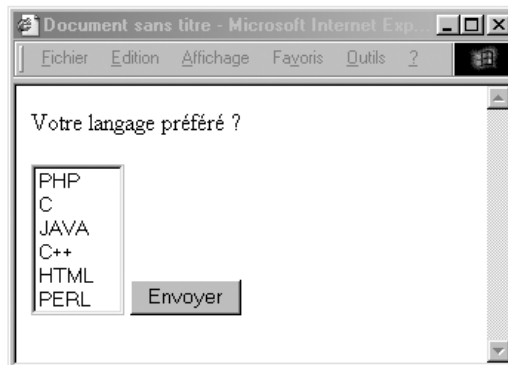
```
<select name="salaire">
<!--Ici les différentes options possibles-->
  <option value="1">Moins de 1000 euros</option>
  <option value="2"
  <?php
  if ($selection == 2)
    echo 'selected';
  ?>
  >Entre 1000 et 2000</option>
  <option value="3">Entre 2000 et 4000</option>
  <option value="4">Plus de 4000</option>
</select>
```

### Taille de la liste

Par défaut, le navigateur proposera une liste déroulante, comme à la figure 8-6. Ce type de liste n'est pas forcément adapté lorsque les choix sont nombreux. Vous pouvez demander une liste dans un cadre plus classique tel qu'à la figure 8-7, en spécifiant un nombre de lignes à afficher dans l'attribut `size`.

Figure 8-7

Liste d'options



```
<select name="id_langage" size="6">
  <option value="1">PHP</option>
  <option value="2">C</option>
```



```
<option value="3">JAVA</option>
<option value="4">C++</option>
<option value="5">HTML</option>
<option value="6">PERL</option>
</select>
<input type="submit" name="Submit" value="Envoyer">
```

### Liste pour sélections multiples

La liste déroulante que nous venons de voir a un gros défaut : elle ne permet de choisir qu'un seul élément. Si vous avez besoin que l'utilisateur fasse plusieurs choix dans votre liste, il pourrait être possible de spécifier arbitrairement cinq listes, ou faire un script JavaScript qui ajoute des listes sur demande de l'utilisateur. Aucune de ces deux solutions n'est toutefois pratique.

Pour gérer les listes à sélections multiples HTML, il est possible d'ajouter un attribut `multiple` à la balise `<select>`. Sur la plupart des navigateurs, le visiteur pourra alors faire plusieurs sélections s'il laisse appuyée la touche Ctrl quand il fait son choix. Chaque nouvelle sélection s'ajoutera alors aux précédentes.

```
<select name="id_langage[]" size="6" multiple>
  <option value="1">PHP</option>
  <option value="2">C</option>
  <option value="3">JAVA</option>
  <option value="4">C++</option>
  <option value="5">HTML</option>
  <option value="6">PERL</option>
</select>
<input type="submit" name="Submit" value="Envoyer">
```

## Champs cachés

Les champs cachés servent à envoyer une valeur sans que l'utilisateur ne le voie et ne puisse la modifier. Ils sont placés dans le formulaire HTML sous la forme :

```
<input type="hidden" name="identifiant" value="52">
```

L'utilisation la plus fréquente est faite dans les formulaires en plusieurs pages. On évite généralement de faire le traitement à la fin de chaque page car si l'utilisateur ne va pas jusqu'au bout, les données peuvent présenter des incohérences. On stocke alors dans des champs cachés les données de la page précédente et on les traite toutes lors de la soumission de la dernière page du formulaire.

#### Attention

Ne mettez jamais de données sensibles dans un champ caché. Elles ne sont cachées que dans le rendu. Le visiteur peut les lire dans le code source de la page HTML produite. Il pourrait aussi assez facilement les modifier et vous envoyer ce qu'il veut à la place.

## Les champs pour mot de passe

Il est difficile d'écrire votre mot de passe dans un formulaire en le laissant visible aux personnes passant derrière vous pendant l'opération. Pour cela, il existe une balise similaire dans ses attributs au champ de texte mais dont la valeur de l'attribut `type` est `password`.

```
<input type="password" name="mot_de_passe" size="8">
```

Notez cependant que si vous utilisez la méthode `GET` pour faire transiter vos variables, votre mot de passe sera affiché dans l'URL et ne sera donc plus caché. Si vous utilisez une valeur par défaut, il sera aussi possible à l'utilisateur de la lire en ouvrant le code source de la page HTML sur son navigateur.

## Image cliquable

Il est possible de créer un bouton de validation graphique. Pour cela, on définit le type `image` dans une balise `input`. Les champs additionnels `src` et `alt` définissent respectivement l'adresse de l'image à afficher et un texte alternatif pour ceux qui n'affichent pas les images (les deux sont obligatoires).

```
<input type="image" name="img" src="images/logo.gif" alt="logo">
```

Quand le visiteur clique sur l'image pour soumettre le formulaire, le navigateur récupère la position du clic et envoie les coordonnées avec le formulaire.

## Envoi d'images et de fichiers

Il est tout à fait possible dans un formulaire de permettre à un utilisateur d'envoyer des images ou des fichiers dont il dispose sur son poste. Par exemple, l'administrateur d'un site d'e-commerce pourra envoyer la nouvelle image d'un produit ou le fichier PDF le concernant. Il vous appartient ensuite de traiter ce fichier (redimensionner, copier, etc.) dans le script destinataire.

On utilise dans ce but la balise `<input type="file">`. Contrairement aux autres balises, il faut spécifier dans la balise `<form>` que nous enverrons des données autres que textuelles. Pour cela, on se sert de l'attribut `enctype` en lui attribuant la valeur `multipart/form-data`.

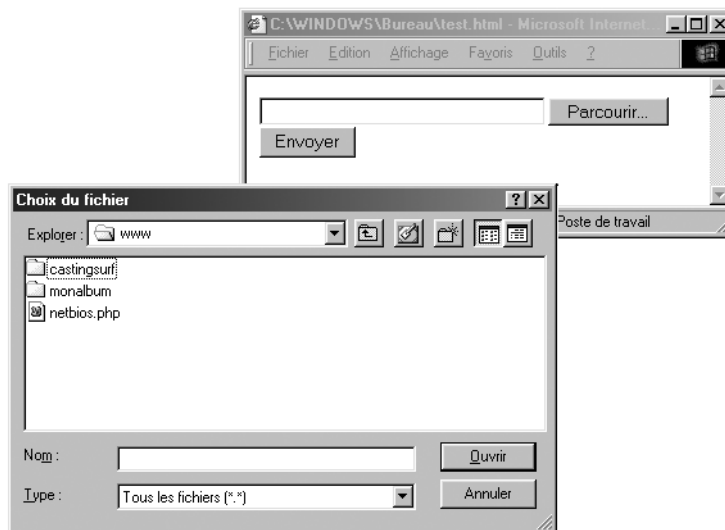
### Note

Seule la méthode `POST` est possible si vous comptez envoyer un fichier via votre formulaire.

Le code exemple suivant enverrait un formulaire tel qu'illustré par la capture d'écran 8-8.

```
<form action="up.php" method="POST" enctype="multipart/form-data">
<P>
<input type="file" name="fichier" size="40">
<input type="submit" value="Envoyer">
</P>
</form>
```

Figure 8-8

*Envoi de fichier*

L'utilisateur sera alors invité à choisir un fichier sur son disque. En soumettant le formulaire, le fichier sera envoyé à votre script (up.php dans notre exemple).

## Réception des données en PHP

Quand un formulaire est envoyé vers un script PHP (l'adresse dans l'attribut `action` est celle d'un de vos scripts), le moteur lit les différentes informations du formulaire et les présente de façon simple au développeur.

### Utilisation des superglobales

Toutes les données envoyées se retrouvent dans ce qu'on appelle des superglobales. Il s'agit de tableaux qui ont une portée globale où qu'ils soient utilisés. Vous pouvez y accéder depuis un contexte global, depuis une fonction ou une méthode de classe sans avoir besoin de vous soucier de la portée des variables.

Dans ce chapitre, nous utiliserons quatre superglobales :

- Le tableau `$_GET[]` contient toutes les données envoyées via l'URL.
- Le tableau `$_POST[]` contient les données envoyées via un formulaire en POST (attribut `method="post"`).
- Le tableau `$_FILES[]` contient les informations sur les fichiers envoyés par le visiteur.
- Le tableau `$_REQUEST[]` est quant à lui une fusion des deux premiers avec la superglobale `$_COOKIE[]` que vous découvrirez au chapitre 10. Si des données avec le même

nom sont envoyées via plusieurs méthodes, la priorité est donnée aux cookies, puis à la méthode POST.

#### Utilisation de \$\_REQUEST ou du triplet \$\_GET, \$\_POST et \$\_COOKIE

L'utilisation de \$\_REQUEST est préconisée par certains mais ne fait pas l'unanimité.

La première école consiste à utiliser la superglobale \$\_REQUEST[] qui permet d'avoir une meilleure abstraction de la provenance sans compromettre la sécurité : tous les paramètres en provenance de l'utilisateur se retrouvent ainsi dans le même espace, quel que soit le mode de transmission. Il est alors possible de changer ou d'accepter plusieurs modes de transmission sans changer le code PHP.

La seconde école consiste en l'utilisation de \$\_GET[], \$\_POST[] et \$\_COOKIE[] afin d'avoir un contrôle plus fin et d'éviter d'éventuelles collisions entre variables si vous utilisez des données différentes sous le même nom dans un formulaire et dans un cookie.

Si vous avez conçu votre application de façon à éviter les conflits de noms, vous pouvez utiliser indifféremment les deux méthodes.

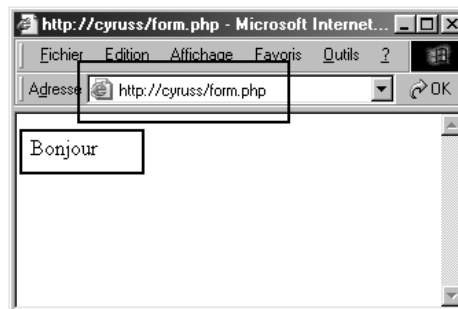
### Récupération d'une donnée simple

Pour récupérer le contenu d'un champ de formulaire du nom de X, il suffit de lire l'index X de la superglobale.

```
echo $_REQUEST['montexte'] ;
```

Figure 8-9

Récupération d'une donnée simple de formulaire



Le code précédent pourrait être l'affichage du champ texte fourni par le formulaire suivant :

```
<form action="form.php" method="POST"><p>
Entrez votre texte : <br>
<input type="text" name="montexte"><br>
<input type="Submit">
</p></form>
```

On aurait pu aussi utiliser directement la superglobale \$\_POST[], puisque notre formulaire est envoyé avec cette méthode :

```
echo $_POST['montexte'] ;
```

## Gestion des `magic_quotes`

Si la directive de configuration `magic_quotes_gpc` (`gpc` correspond à `get post cookies`) est activée dans votre configuration, PHP modifie toutes les données reçues avant que vous n'y accédiez. Il procède de la même façon que la fonction `addslashes()`, en ajoutant une barre oblique inverse devant certains caractères comme les apostrophes ou les barres obliques inverses elles-mêmes. Pour vous donner un exemple, si la chaîne de caractères « L'idée est bonne » est envoyée via un formulaire, elle devient « L\'idée est bonne » si la directive `magic_quotes_gpc` est activée. Cette conversion est faite pour permettre d'utiliser sans risque les données dans des requêtes SQL (vous êtes alors protégé contre les injections SQL les plus courantes). Si vous retrouvez souvent des barres obliques inverses devant vos apostrophes, c'est probablement la directive `magic_quotes_gpc` qui a transformé votre chaîne automatiquement.

L'utilisation des `magic_quotes_gpc` est généralement assez gênante. Elle impose de faire une double conversion si les données sont destinées à l'affichage ou à toute autre utilisation que des requêtes SQL. Elle empêche aussi de gérer facilement les données puisqu'il faut à tout moment savoir d'où vient une donnée avant de l'utiliser pour savoir si elle a été transformée ou pas. Depuis quelques temps, l'équipe PHP conseille de désactiver cette option, et c'est aussi ce que nous vous recommandons. Soyez vigilant tout de même car il ne faudra alors pas oublier de préparer manuellement vos données lors de leur utilisation dans des requêtes SQL.

```
magic_quotes_gpc = Off
```

Toutefois, si vous souhaitez diffuser votre script, il vous faudra tenir compte de cette valeur, car de nombreuses installations continuent à activer cette directive. Pour vous aider, vous pouvez faire appel à la fonction `get_magic_quotes_gpc()`. Elle retourne `TRUE` si la directive de configuration est activée, `FALSE` sinon ; à vous d'en tenir compte et d'opérer les transformations qui s'imposent.

```
if (get_magic_quotes_gpc()) {  
    $donneePourSQL = $_REQUEST['donnee'] ;  
} else {  
    $donneePourSQL = addslashes( $_REQUEST['donnee'] ) ;  
}
```

## Retours à la ligne et zones de texte

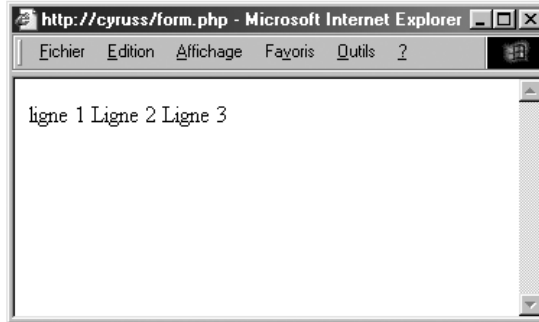
Il arrive qu'un visiteur saute des lignes ou insère des caractères de fin de ligne dans un champ. C'est normalement le cas dans les zones de texte `<textarea>` qui sont faites pour accueillir des textes complets, potentiellement en plusieurs paragraphes.

```
<textarea name="t" cols="50" rows="10">  
ligne 1  
Ligne 2  
Ligne 3  
</textarea>
```

Si on affiche directement dans la page HTML le contenu du champ de formulaire précédent avec `echo $_REQUEST['t']`, l'affichage se fera sur une seule ligne (voir figure 8-10).

**Figure 8-10**

*Affichage d'une zone de texte avec plusieurs lignes*



Le texte de l'exemple précédent est bien affiché mais pas forcément comme on pourrait s'y attendre : les retours à la ligne ont disparu. En effet, nous avons affiché le texte tel quel, les retours à la ligne sont donc faits en texte pur (vous pouvez voir que dans la source de la page HTML résultante on a bien trois lignes séparées) et les espaces blancs (espaces surnuméraires, fins de ligne, tabulations, etc.) sont ignorés en HTML. Si vous destinez le texte à un affichage HTML, il faudra transformer les changements de ligne en balises `<br>`. En PHP, vous pouvez faire cette opération via la fonction `n12br()` (voir résultat à la figure 8-11).

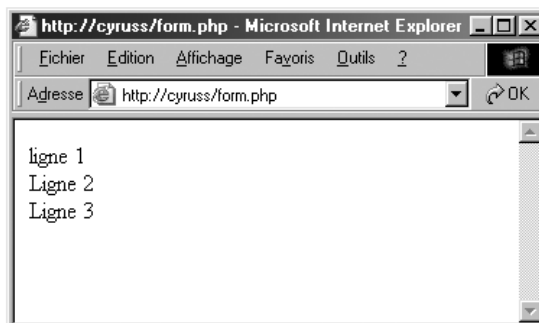
#### Astuce mnémotechnique

Prononcé en anglais : n1 pour *new line* ; 2 pour *to* ; br pour indiquer la balise HTML `<br />`. La traduction logique est donc « nouvelle ligne vers balise `<br />` ».

```
<?php
echo n12br($_REQUEST['texte_long']);
?>
```

**Figure 8-11**

*Utilisation de n12br()*



Contrairement à l'exemple précédent, le code source correspondant serait le suivant :

```
ligne 1<br />
Ligne 2<br />
Ligne 3
```

### Gestion des tags XHTML et HTML

Vous pouvez remarquer que PHP produit des balises `<br />` et non des balises `<br>`. Il s'agit de la syntaxe XML de XHTML, alors que nous utilisons dans nos exemples du code HTML. PHP n'est malheureusement pas homogène à ce niveau et certaines fonctions sont faites pour produire du XHTML, d'autres pour produire du HTML. Si vous souhaitez avoir du HTML valide, vous pouvez utiliser le code suivant à la place de `nl2br()`.

```
<?php
$n1 = array("\r\n", "\r", "\n");
echo str_replace($n1, "<br>\n", $_REQUEST['texte_long']);
?>
```

### Utilisation des cases à cocher

La valeur par défaut (modifiable lors de la création du formulaire) d'une case à cocher est 0n. Ce texte sera transmis uniquement si la case est cochée, sinon rien ne sera envoyé.

Vous pouvez donc tester l'état d'une case à cocher avec la fonction `isset()`. Si l'élément cherché existe (quelle que soit sa valeur), c'est que la case a été cochée, sinon c'est qu'elle est décochée.

```
if ( isset($_REQUEST['case_a_cocher']) ) {
    echo 'La case est cochée ' ;
    echo 'sa valeur est ', $_REQUEST['case_a_cocher'] ;
} else {
    echo 'La case n'est pas cochée' ;
}
```

### Validation de données avec l'extension Filter

Une solution de filtrage simple est arrivée avec la version 5.2 de PHP : il s'agit de l'extension Filter. Cette extension permet de valider et de filtrer les données venant habituellement de sources non sécurisées comme les entrées utilisateur.

Au lieu de récupérer directement les données utilisateur dans les variables super-globales (`$_GET[]`, `$_POST[]`, `$_REQUEST[]...`), on passe par des fonctions qui font immédiatement un filtrage suivant le type de donnée qui est attendue. Il n'y a donc plus d'erreur possible sur le type ou le format de données. Le code s'en trouve simplifié car le développeur n'a plus à redévelopper lui-même ces filtres avec le risque d'en oublier.

**Note**

Une fois les filtres maîtrisés, vous avez tout intérêt à ne plus jamais récupérer vos données directement dans les superglobales mais toujours passer par les fonctions de filtrage. Vous évitez ainsi les classiques oublis de vérification et vous améliorez la sécurité de votre code.

L'extension est composée de quatre composants :

- une liste de fonctions qui permettent d'agir sur les données en entrée et d'y appliquer éventuellement des filtres ;
- une liste de constantes représentant les sources de données (GET, POST, COOKIE, etc.) sur lesquelles appliquer les fonctions précédentes ;
- une liste de filtres à appliquer, ou non, aux données ;
- une liste d'options permettant de gérer finement l'application des filtres.

La combinaison de ces composants permet, par exemple, de tester l'existence d'une variable adresse provenant d'un formulaire, puis de la récupérer tout en vérifiant qu'il s'agit bien d'une adresse e-mail.

**Validation d'une adresse e-mail**

Afin de vous permettre de voir d'un seul coup d'œil les possibilités de cette extension, prenons un exemple classique : la réception d'un formulaire contenant une adresse e-mail. L'extension Filter va nous permettre de valider qu'il s'agit là d'une adresse e-mail bien formée.

Le formulaire (classique)

```
<form method='POST' action='filter.php'>
<input type=text name=mel>
<input type="Submit" value="Soumettre la requête">
</form>
```

Le script recevant le formulaire

```
<?php
// Validation
$mail = filter_input(INPUT_POST, 'mel', FILTER_VALIDATE_EMAIL) ;

if ($mail === FALSE){
    echo 'Le mail fourni n est pas valide' ;
}elseif($mail === NULL){
    echo 'mel n était pas définie.' ;
}else{
    echo "La variable est une adresse e-mail valide : $mail";
}
?>
```



## Filtrer une variable

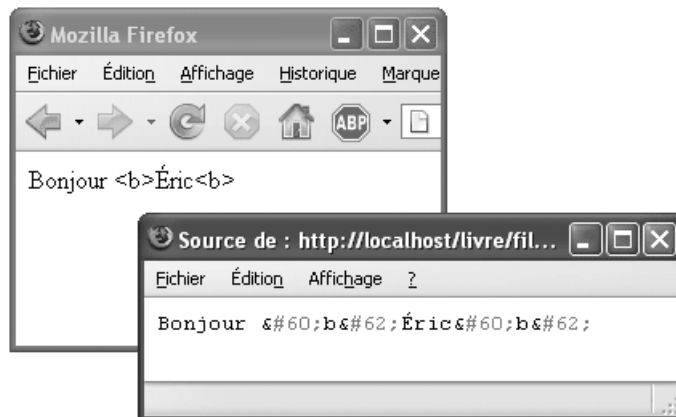
On filtre une variable avec la fonction `filter_var()`. Celle-ci prend comme arguments une variable et un type de filtre. Il est également possible de lui ajouter un tableau d'options en troisième argument. La fonction retourne la variable filtrée, ou `FALSE` en cas d'échec.

Dans l'exemple suivant, nous utilisons `FILTER_SANITIZE_SPECIAL_CHARS` qui filtre tous les caractères spéciaux en HTML et XML pour les convertir en entité (voir à ce sujet la fonction `htmlspecialchars()`).

```
<?php
$adresse = "Bonjour <b>Éric<b>" ;
$message = filter_var($adresse, FILTER_SANITIZE_SPECIAL_CHARS) ;
echo $message ;
?>
```

Figure 8-12

Utilisation de  
`filter_var()`



La fonction `filter_var_array()` se comporte de manière identique mais prend en paramètre un tableau de données et retourne ce dernier avec les données filtrées. Cela permet d'effectuer davantage de traitements en une seule passe.

## Les filtres les plus courants

Les filtres sont représentés par des constantes. On les passera en paramètres des fonctions que nous utiliserons. On peut distinguer deux types de filtres : les filtres de validation et ceux dits de conversion.

La liste de tous les filtres disponibles peut être obtenue via la fonction `filter_list()`.

Seuls deux filtres n'appartiennent à aucune de ces deux catégories :

- Le filtre `FILTER_UNSAFE_RAW` se contente de retourner les données sans rien vérifier ni modifier. Comme son nom l'indique, cette fonction retourne la donnée brute et n'ajoute donc aucune sécurité.

- Le filtre `FILTER_CALLBACK` permet d'appeler une fonction utilisateur qui validera les données. Elle ne fait donc rien par elle-même. Le nom de la fonction de rappel est à passer en dernier paramètre, à la place des options.

### Les filtres de validation

Ces filtres servent à valider une donnée sans la modifier. Si la donnée est invalide, la valeur `false` est renvoyée. Ils contiennent généralement le mot-clé `VALIDATE` :

- `FILTER_VALIDATE_EMAIL` s'assure que la donnée représente une adresse e-mail (mais pas que l'adresse existe).
- `FILTER_VALIDATE_URL` s'assure que la donnée représente une URL (adresse web).
- `FILTER_VALIDATE_REGEXP` s'assure que la donnée correspond à une expression rationnelle compatible Perl (PCRE) passée en paramètre avec les options.
- `FILTER_VALIDATE_INT` s'assure que la donnée représente un nombre entier.
- `FILTER_VALIDATE_FLOAT` s'assure que la donnée représente un nombre réel.
- `FILTER_VALIDATE_BOOLEAN` s'assure que la donnée représente un booléen et retourne `NULL` en cas d'échec au lieu de `FALSE`.
- `FILTER_VALIDATE_IP` s'assure que la donnée représente une adresse IP.

### Les filtres de conversion

Ces filtres servent à convertir la donnée de façon à s'assurer qu'elle respecte toujours le format attendu. Ces filtres contiennent généralement le mot-clé `SANITIZE` :

- `FILTER_SANITIZE_STRIPPED` retire les balises HTML.
- `FILTER_SANITIZE_EMAIL` retire tous les caractères qui ne devraient pas se retrouver dans une adresse e-mail, mais ne s'assure pas que le résultat retourné est une adresse valide.
- `FILTER_SANITIZE_SPECIAL_CHARS` code les caractères spéciaux en HTML et XML sous forme d'entités.
- `FILTER_SANITIZE_NUMBER_INT` retire tous les caractères qui ne composent pas un nombre entier.
- `FILTER_SANITIZE_NUMBER_FLOAT` retire tous les caractères qui ne composent pas un nombre à virgule flottante.
- `FILTER_SANITIZE_URL` retire tous les caractères qui ne composent pas une URL, mais ne s'assure pas que le résultat retourné est une adresse valide.
- `FILTER_SANITIZE_ENCODED` convertit tous les caractères étendus d'une URL dans leurs équivalents `%xx`.
- `FILTER_SANITIZE_MAGIC_QUOTES` applique la fonction `addslashes()` et opère une action identique à la directive de configuration `magic_quotes_gpc`.

## Configuration

Le filtre par défaut appliqué à toutes les variables utilisateur (GET, POST et COOKIE) est déterminé par la directive de configuration `filter.default` du `php.ini`. Nous vous recommandons toutefois de ne pas la modifier sans raison précise car vous auriez alors une installation incompatible avec les installations standards.

## Utiliser les filtres sur les superglobales

Plutôt que de lire les données dans les variables superglobales puis de les passer à `filter_var()`, il est possible de demander au filtrage de lire directement les données utilisateur.

Cette procédure a l'avantage de permettre une vérification rapide du code : si nous filtrons bien toutes nos variables en entrées, nous n'aurons jamais à lire les variables superglobales. L'extension Filter le fait pour nous et nous retourne les résultats déjà filtrés.

À la place de `filter_var()` et `filter_var_array()`, nous utilisons donc `filter_input()` et `filter_input_array()`. Au lieu de prendre une valeur directement en paramètre, on donne à ces variables une source et un nom pour la donnée à récupérer.

Ainsi, dans notre exemple, nous cherchons une donnée nommée "adr" dans un formulaire soumis avec une requête POST et nous lui appliquons le même filtre que précédemment :

```
<?php
$m = filter_input(INPUT_POST, "adr", FILTER_SANITIZE_SPECIAL_CHARS) ;
echo $m ;
?>
```

Afin de ne pas tenter de filtrer une donnée inexistante, la fonction `filter_has_var()` prend en paramètres une source de données et un nom de champ. Elle retourne une valeur vraie ou fausse en fonction de l'existence ou non de la valeur recherchée.

```
<?php
if ( filter_has_var(INPUT_POST, "adr") ) {
    $m = filter_input(INPUT_POST, "adr", FILTER_SANITIZE_SPECIAL_CHARS) ;
    echo $m ;
}
?>
```

## Sources de données

Les sources de données sont représentées par différentes constantes :

- `INPUT_GET` pour les données dans `$_GET` ;
- `INPUT_POST` pour les données dans `$_POST` ;
- `INPUT_COOKIE` pour les données dans `$_COOKIE` ;
- `INPUT_REQUEST` pour les données dans `$_REQUEST` ;
- `INPUT_SESSION` pour les données dans `$_SESSION` ;
- `INPUT_ENV` pour les données dans `$_ENV` ;
- `INPUT_SERVER` pour les données dans `$_SERVER`.

## Affiner les filtres avec des options

Lors de l'appel à une fonction de filtrage, le dernier paramètre est réservé aux options. Ces options permettent de configurer finement l'application du filtre.

Chaque option est représentée par une constante. On peut les associer en les séparant par le caractère « | » pour cumuler plusieurs options :

- `FILTER_NULL_ON_FAILURE` permet de retourner `NULL` en cas d'échec plutôt que `FALSE`.
- `FILTER_FLAG_ALLOW_HEX_` et `_OCTAL` autorise l'écriture des nombres filtrés en hexadécimal (en les faisant débiter par `0x`) et en octal (en les faisant débiter par `0`).
- `FILTER_FLAG_STRIP_LOW` et `_HIGH`, lors du filtrage de `HTML` ou d'`URL`, retire respectivement tous les caractères avec un code `ASCII` inférieur à `32` et/ou supérieur à `127`.
- `FILTER_FLAG_ENCODE_LOW`, `_HIGH` et `_AMP`, lors du filtrage de `HTML` ou d'`URL`, impose de coder respectivement tous les caractères avec un code `ASCII` inférieur à `32`, supérieur à `127`, et/ou les esperluettes (caractère « `&` »).
- `FILTER_FLAG_NO_ENCODE_QUOTES`, lors du filtrage de `HTML`, impose de ne pas coder les apostrophes et guillemets sous forme d'entités.
- `FILTER_FLAG_ALLOW_FRATION`, `_THOUSAND` et `_SCIENTIFIC`, lors du filtrage des nombres à virgule flottante, autorise le signe de fraction, le séparateur de milliers et/ou l'exposant avec notation scientifique.
- `FILTER_FLAG_SCHEME_REQUIRED`, `_HOST_REQUIRED`, `_PATH_REQUIRED`, et `_QUERY_REQUIRED`, dans le filtrage d'`URL`, impose la présence du protocole (`http://` par exemple), du domaine (adresse `IP` ou nom de domaine), du chemin (partie juste après le nom de domaine, qui peut être réduite à « `/` »), et/ou d'une chaîne de requêtes (variables qui suivent le point d'interrogation).
- `FILTER_FLAG_IPV4`, `_IPV6`, `_NO_RES_RANGE`, `_NO_PRIV_RANGE`, lors du filtrage d'une adresse `IP`, impose une adresse `IPv4` ou `IPv6`, interdit les adresses réservées et/ou interdit les adresses privées.

Si le filtre attend une option telle qu'une fonction de rappel ou une expression rationnelle, et que vous souhaitez en plus préciser une option, il faut alors passer un tableau associatif.

## Exemples d'utilisation

### Validation d'un entier dans un intervalle

Toutes les fonctions de l'extension `Filter` acceptent en paramètre un tableau d'options permettant de préciser leur fonctionnement. L'exemple suivant montre la validation d'un nombre dans un intervalle donné.

```
<?php
// Données à valider
$entier1 = 12;
$entier2 = 21;
```

```
// Intervalle de validation
$min = 20;
$max = 30;

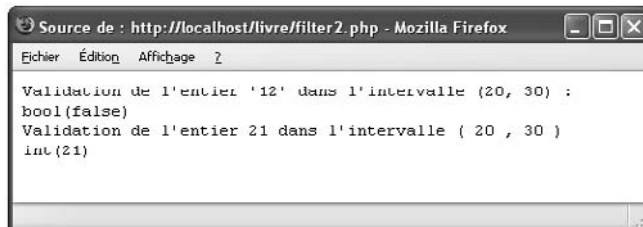
// Filtrage
echo "Validation de l'entier '$entier1' ";
echo "dans l'intervalle ($min, $max) : \n";

var_dump(filter_var($entier1, FILTER_VALIDATE_INT,
    array('options' =>
        array('min_range' => $min, 'max_range' => $max))
    )
);
echo "Validation de l'entier $entier2 dans l'intervalle " ;
echo "( $min , $max )\n";

var_dump(filter_var($entier2, FILTER_VALIDATE_INT,
    array('options' =>
        array('min_range' => $min, 'max_range' => $max))
    )
);
?>
```

**Figure 8-13**

*Validation  
d'un intervalle*



### Validation avec une expression régulière

Le tableau d'options peut servir à passer une expression régulière comme le montre notre exemple ci-dessous :

```
<?php
// Données à valider
$chaîne1 = 'baobab';
$chaîne2 = 'babo';

// Expression régulière
$regex = '/^(b(a|o))*$/';

// Validation
if(filter_var($chaîne1, FILTER_VALIDATE_REGEXP,
    array("options"=>array("regex" => $regex))) !== false){
    echo "La chaîne $chaîne1 est valide.\n";
}
```

```
if(filter_var($chaine2, FILTER_VALIDATE_REGEXP,
    array("options">array("regexp" => $regexp))) !== false){
    echo "La chaine $chaine2 est valide.\n";
}
?>
```

### Validation d'une adresse IP

Avec l'extension Filter il vous est possible de valider les adresses ipv4 et ipv6, les adresses locales, les filtres...

```
<?php
// Données à valider
$ip = '192.168.0.1';

// Validation
if(filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4) !== false)
    echo "L'adresse $ip est une adresse IPV4 valide.\n";

if(filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE) !== false)
    echo "L'adresse n'est pas locale.\n";
else echo "L'adresse $ip est locale.\n";
?>
```

### Listes à sélections multiples

La balise `<select>` nous permet d'avoir des listes à sélections multiples. Quand on fait plusieurs sélections, le navigateur envoie plusieurs fois le même paramètre avec les différentes valeurs. PHP interprète mal cette syntaxe et la dernière écrase les autres. Vous ne relirez donc qu'une seule des sélections du visiteur.

Pour pallier ce problème, vous pouvez utiliser une syntaxe de tableau. En mettant `[]` à la fin du nom du champ, PHP comprendra qu'il s'agit d'une liste, avec plusieurs valeurs.

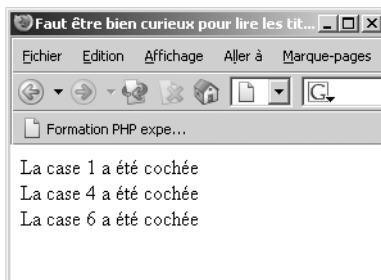
```
<select name="id_langage[]" size="6" multiple>
  <option value=1>PHP</option>
  <option value=2>C</option>
  <option value=3>JAVA</option>
  <option value=4>C++</option>
  <option value=5>HTML</option>
  <option value=6>PERL</option>
</select>
<input type="submit" name="Submit" value="Envoyer">
```

La variable récupérée par PHP est alors un tableau contenant les différentes valeurs sélectionnées (voir code suivant et figure 8-14).

```
foreach( $_REQUEST['id_langage'] as $option ) {
    echo 'La case ', $option, ' a été cochée<br \>';
}
```

Figure 8-14

Affichage du script



## Gestion des images cliquables

```
<input type="image" name="img" src="images/logo.gif" alt="logo">
```

Quand une image est employée comme bouton de soumission du formulaire, le navigateur enregistre la position du clic de souris dans l'image et transmet les coordonnées lors de l'envoi.

Vous pouvez par la suite y accéder via PHP comme à toutes les autres données. L'abscisse du point de clic est donnée par un index qui porte le même nom que le bouton cliqué avec `_x` comme suffixe. L'ordonnée, elle, utilise `_y` comme suffixe. Le point de coordonnées (0,0) est le point en haut à gauche de l'image.

```
<?php
echo $_REQUEST['img_x'];
echo $_REQUEST['img_y'];
```

## Téléchargements d'images et de fichiers

```
<form action="up.php" method="POST" enctype="multipart/form-data">
<p>
<input type="file" name="fichier" size="40">
<input type="submit" value="Envoyer">
</p>
</form>
```

Les fichiers envoyés avec un formulaire se traitent différemment des autres données. La différence la plus visible est l'utilisation de la superglobale `$_FILES[]`. C'est avec cette dernière que vous pourrez accéder aux fichiers téléchargés.

Il s'agit d'un tableau associatif à deux niveaux. Le premier champ doit être le nom du champ de formulaire (fichier dans notre exemple), le second paramètre concerne plusieurs informations sur le fichier transmis :

- `$_FILES['fichier']['name']` : nom et adresse originels du fichier sur le disque de l'utilisateur ;
- `$_FILES['fichier']['type']` : type mime du fichier ;
- `$_FILES['fichier']['size']` : taille du fichier en octets ;

- `$_FILES['fichier']['tmp_name']` : nom et adresse du fichier temporaire stocké sur le serveur ;
- `$_FILES['fichier']['error']` : code erreur associé au téléchargement.

On pourrait donc utiliser le code suivant pour traiter notre exemple :

```
<?php
$fichier = $_FILES['fichier']['name'];
$taille = $_FILES['fichier']['size'];
$tmp = $_FILES['fichier']['tmp_name'];
$type = $_FILES['fichier']['type'];
$erreur = $_FILES['fichier']['error'];

echo "Nom d'origine => $fichier <br />";
echo "Taille => $taille <br />";
echo "Adresse temporaire sur le serveur => $tmp <br />";
echo "Type de fichier => $type <br />";
echo "Code erreur => $erreur. <br />";
?>
```

### Utilisation et traitement du fichier

Recevoir des fichiers peut avoir de multiples applications. On peut citer l'échange de fichiers, l'envoi d'illustrations pour modifier le site via une interface d'administration ou la modification et le redimensionnement d'images. Ce dernier exemple est illustré dans un cas d'application du chapitre 25, « Gestion des images ».

PHP crée un fichier temporaire sur le serveur pour chaque fichier envoyé. Ce fichier étant détruit automatiquement à la fin de l'exécution du script, vous avez généralement besoin de le sauvegarder à un emplacement définitif. Cet enregistrement peut se faire par l'intermédiaire de la fonction `move_uploaded_file()`. PHP déplace alors le fichier depuis son emplacement temporaire sur le serveur vers la destination finale que vous lui aurez allouée. Cette fonction prend deux paramètres : le nom du fichier téléchargé et l'adresse de destination.

```
<?php
$nom_fichier = $_FILES['img']['tmp_name'];
$nom_destination = './img/725.jpg';
move_uploaded_file($nom_fichier, $nom_destination);
?>
```

### Gérer les noms de fichiers

Quand vous enregistrez un fichier, pensez bien à vérifier le nom ou à en changer. Vous éviterez ainsi les problèmes dans le cas très fréquent où deux utilisateurs soumettent des fichiers différents avec le même nom.

Faites aussi attention aux extensions de fichiers si ces derniers sont sauvegardés dans un emplacement accessible. Un visiteur mal intentionné pourrait vous envoyer un fichier PHP. Si vous l'enregistrez tel quel dans un répertoire sans autre configuration, il suffira à



cette personne de taper la bonne adresse pour exécuter son script chez vous. En général, les fichiers téléchargés doivent soit avoir une extension neutre (type .txt), soit être déposés dans un répertoire qui n'exécute pas les scripts et CGI.

### Détection d'erreurs

PHP retourne un code d'erreur dans la superglobale `$_FILES[]`. Le cas dans lequel on se trouve est indiqué dans cette variable de la manière suivante :

```
$_FILES['fichier']['error']
```

Voici une liste des valeurs que vous pourrez rencontrer :

- `UPLOAD_ERR_OK` - le téléchargement est correct, cette constante représente la valeur nulle.
- `UPLOAD_ERR_INI_SIZE` - le fichier téléchargé excède la taille maximale définie dans la configuration de PHP par la directive `upload_max_filesize`.
- `UPLOAD_ERR_FORM_SIZE` - le fichier téléchargé excède la taille maximale définie dans le formulaire HTML.
- `UPLOAD_ERR_PARTIAL` - le fichier n'a été que partiellement chargé, probablement arrêté par l'utilisateur.
- `UPLOAD_ERR_NO_FILE` - aucun fichier n'a été téléchargé.

On peut utiliser le code suivant pour afficher l'état du téléchargement :

```
if ($err = $_FILES['fichier']['error']) {  
    echo "il y a eu une erreur<br>" ;  
    if ($err == UPLOAD_ERR_INI_SIZE)  
        echo "Le fichier est plus gros que le max autorisé par PHP";  
    elseif ($err == UPLOAD_ERR_FORM_SIZE)  
        echo "Le fichier est plus gros qu'indiqué dans le formulaire";  
    elseif ($err == UPLOAD_ERR_PARTIAL)  
        echo "Le fichier n'a été que partiellement téléchargé";  
    elseif ($err == UPLOAD_ERR_NO_FILE)  
        echo "Aucun fichier n'a été téléchargé." ;  
} else echo "fichier correctement téléchargé" ;
```

Globalement, on conseille de toujours vérifier la présence d'erreurs avant d'utiliser le fichier.

## Formulaire dynamique et tableaux

PHP peut envoyer au navigateur des balises de formulaires comme n'importe quelle autre balise HTML. On peut donc gérer un formulaire dynamiquement et insérer ou non des champs selon des paramètres internes. Une des possibilités offertes est de créer un nombre variable de champs et de pouvoir récupérer toutes les données avec PHP par la suite.

Il existe deux méthodes pour gérer ce cas. La première consiste à utiliser la même syntaxe que les listes à sélections multiples. En ajoutant [] après chaque nom de champ, PHP crée des tableaux avec chaque valeur, comme pour une sélection multiple.

```
for( $i = 0 ; $i < 10 ; $i++ ) {  
    echo "<input type='text' name='montexte[]'>" ;  
}
```

Le problème est alors qu'il nous sera impossible de gérer des listes à sélections multiples ainsi. Il nous faudrait soit utiliser [][] (ce que PHP ne comprend pas), soit fusionner les résultats de toutes ces listes en une seule. L'astuce est de pré-crée l'index lors de la création du formulaire :

```
<?php  
for( $i = 0 ; $i < 10 ; $i++ ) {  
    echo 'select name="id_langage[$i][]" size="6" multiple' ;  
    echo 'option value=1>PHP</option>' ;  
    echo 'option value=2>C</option>' ;  
    echo 'option value=3>JAVA</option>' ;  
    echo 'option value=4>C++</option>' ;  
    echo 'option value=5>HTML</option>' ;  
    echo 'option value=6>PERL</option>' ;  
    echo '</select>' ;  
}
```

PHP vous permet en effet de construire des tableaux de la manière que vous voulez, tant que seul le dernier index est indéfini.

## Autres problématiques

### *Gestion du temps*

Il peut arriver que l'envoi d'un fichier trop volumineux engendre un message d'erreur indiquant que le temps maximal d'exécution du script a été dépassé.

Pour éviter cela, on peut faire appel à la fonction `set_time_limit()`, qui permet de fixer le temps maximal d'exécution d'un script.

`set_time_limit()` fixe ce délai d'expiration en secondes. Si la limite est atteinte, le script s'interrompt et renvoie une erreur fatale. La valeur par défaut est de 30 secondes, mais peut être changée avec la directive `max_execution_time` définie dans le fichier de configuration. Si la valeur est zéro, il n'y a alors aucune limite imposée.

Lorsqu'elle est appelée, la fonction `set_time_limit()` remet le compteur à zéro. En d'autres termes, si la limite par défaut est à 30 secondes et si après 25 secondes d'exécution du script l'appel `set_time_limit(20)` est fait, alors le script tournera pendant un total de 45 secondes avant de finir.

Un autre cas à gérer est l'arrêt du téléchargement par l'utilisateur. Celui-ci peut à tout moment cliquer sur le bouton arrêter de son navigateur. Pour permettre au script de poursuivre son exécution, même dans le cas où l'utilisateur se déconnecte, on utilise la fonction `ignore_user_abort()`.

**Note**

La fonction `set_time_limit()` n'a pas d'effet si PHP fonctionne avec une configuration de type « safe mode ». Il n'y a pas d'autre solution que de changer de mode, ou de modifier la durée maximale d'exécution dans le fichier de configuration global.

## Gestion de la taille des données

Pour des raisons de sécurité, PHP limite la taille des fichiers via la directive `post_max_size` contenue dans le fichier de configuration `php.ini`.

Par défaut, PHP accepte des fichiers de taille inférieure ou égale à 8 Mo.

```
; Maximum size of POST data that PHP will accept.  
post_max_size = 8M
```

Il est bien entendu possible de modifier ce paramètre pour accepter le téléchargement de fichiers plus lourds.

## Stockage des fichiers temporaires

Les données téléchargées via un formulaire sont stockées dans le répertoire temporaire (par défaut `/tmp` sous Linux). Potentiellement, cela peut être considéré comme une faille de sécurité, car dans le cas d'un hébergement mutualisé, tout le monde a accès à ce répertoire. N'importe qui peut alors lire le nom et le contenu des fichiers téléchargés.

## Sécurité et données reçues

La sécurité sera abordée dans un chapitre dédié, plus loin dans ce livre, mais il est important de rappeler qu'il faut toujours analyser le contenu des données reçues de l'utilisateur :

- Recevoir du HTML peut amener à des défaçages (changement d'apparence) de site.
- Recevoir du HTML avec JavaScript peut amener à des problèmes de Cross Site Scripting.
- Recevoir des données avec des apostrophes peut permettre des failles dites d'injection SQL.
- Recevoir des noms de fichiers peut amener à une faille de divulgation, etc.

Pensez à analyser toutes les données provenant de l'utilisateur, vérifiez qu'elles ne contiennent que des caractères autorisés dont la valeur fait bien partie de la liste des valeurs possibles, etc.

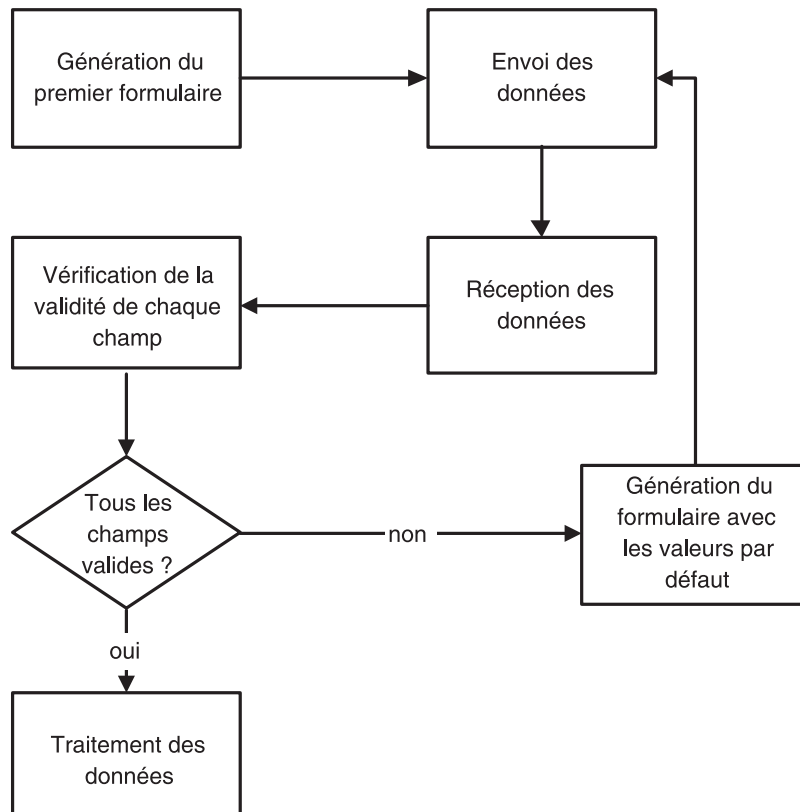
Si la donnée est un contenu libre, utilisez une fonction d'échappement. Il en existe une pour pratiquement tout type d'utilisation : échappement des guillemets pour le SQL en bases de données, échappement des tags HTML contre le JavaScript, etc.

Pour plus de renseignements sur ces questions de sécurité, vous pouvez consulter le chapitre 27, dédié à la sécurité.

## Procédure de gestion des formulaires

Il est fréquent qu'un visiteur ne remplisse pas correctement un formulaire dès la première tentative, soit de son fait (oubli de champs), soit du vôtre (*login* demandé déjà existant). Dans ce cas, on réaffiche le formulaire, mais on ne peut décemment demander à l'utilisateur de tout retaper. Donc, on va réafficher le formulaire en lui donnant en valeurs par défaut les champs qui étaient corrects et en surlignant les champs à réécrire avec, éventuellement, un message explicatif. Le schéma de cette procédure est donné à la figure 8-15.

**Figure 8-15**  
*Gestion des formulaires*



### Génération dynamique du formulaire

Imaginons qu'un utilisateur veuille modifier des informations sur son compte ou que certaines informations qu'il avait saisies soient erronées. Dans ce cas, on va créer dynamiquement un formulaire avec des valeurs pré-remplies.

La première étape consiste à récupérer toutes les valeurs correctes fournies par l'utilisateur. Dans le cas d'une modification, on utilisera les valeurs contenues dans la base de données. Dans l'autre cas, on récupérera les informations valides que nous avait précédemment transmises l'utilisateur.

La seconde étape consiste à réécrire le formulaire en utilisant les attributs `value` des balises d'entrée et en leur indiquant les valeurs déjà spécifiées et correctes. Une utilisation optimale consistera à indiquer de façon visible (en rouge par exemple) les champs manquants ou inexacts.

On notera qu'il n'est pas sécurisé de réafficher les champs secrets ; il vaut mieux demander leur réécriture (ou enregistrer le résultat et ne pas afficher le champ du tout).

# 9

## Environnement web et superglobales

---

Il est fréquent d'avoir besoin d'informations en rapport avec le contexte d'exécution de PHP. Dans un environnement Web, ce peut être connaître le nom du serveur, savoir d'où vient le visiteur, quelle est la configuration actuelle ou récupérer l'adresse IP du client. Nous avons vu au chapitre précédent comment utiliser certaines de ces informations avec les formulaires web. Voici maintenant le reste des informations que vous donne PHP concernant son contexte d'exécution. Nous traiterons donc entre autres de l'authentification HTTP et nous aborderons des informations plus générales sur le fonctionnement des requêtes et de PHP lui-même.

### Descriptif du contexte Web

Dans un environnement Web, nous pouvons distinguer deux principaux acteurs : le serveur, qui met du contenu à disposition, et le navigateur, qui demande du contenu. Le navigateur et son environnement (machine, utilisateur et logiciel) sont souvent appelés par la dénomination « client ».

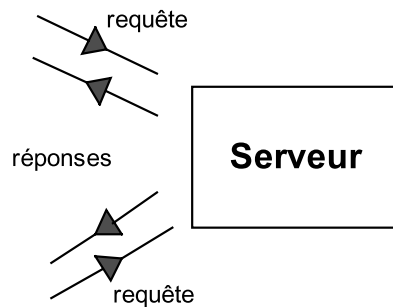
#### *Client-serveur*

Par opposition au modèle de client lourd où tous les programmes fonctionnent et sont exécutés en local, le modèle client-serveur centralise les informations sur un serveur distant et tous les clients s'y connectent pour utiliser ses services.

Le fonctionnement des applications Internet repose sur le principe de base suivant. Un programme client, doté généralement d'une interface conviviale (un navigateur Internet par exemple), est installé sur votre ordinateur. En faisant un simple appel à une page web, le client déclenche l'exécution d'une action complexe sur le serveur. Celui-ci exécute la commande et retourne l'information demandée au programme client, qui l'affiche sous une forme exploitable. La figure 9-1 illustre ce principe.

Ce système permet principalement de centraliser les ressources (au niveau du serveur) et de se soustraire aux problèmes de portage sur différentes architectures, le logiciel client utilisant généralement des protocoles standards.

**Figure 9-1**  
*Vue client-serveur*



## **En-tête et contenu**

Lorsque vous consultez une page web sur un serveur Internet, vous utilisez le protocole HTTP pour rapatrier le contenu de la page sur votre ordinateur. Pour arriver à cela, votre poste client envoie une requête au serveur, qui lui répond en renvoyant simplement la page souhaitée.

Dans le dialogue qui existe entre le navigateur et le serveur web, il y a deux informations différentes qui transitent l'une après l'autre.

Tout d'abord, le bloc d'en-têtes, qui correspond aux messages techniques que s'adressent le serveur et le navigateur. Il s'agit d'un ensemble de lignes permettant de fournir des informations supplémentaires à l'interlocuteur (nom du navigateur, page demandée, etc.). C'est par exemple dans l'en-tête que le serveur précise au navigateur qu'il doit créer un nouveau cookie.

Chaque ligne de l'en-tête est composée d'un nom décrivant le type d'en-tête suivi du caractère deux-points et d'une valeur, par exemple : « Location : index2.html ».

Ensuite vient le contenu qui va constituer la page visualisée ; en d'autres termes le code HTML renfermé dans les fichiers demandés. Ce contenu est séparé du bloc d'en-têtes par une ligne vide. Une fois cette ligne envoyée, tout ce qui suit est le contenu de la page ; il est par la suite impossible de définir d'autres en-têtes.

**PHP et la fin des en-têtes**

PHP insère le délimiteur de fin d'en-têtes à la réception du premier caractère destiné à l'affichage. Ainsi, si vous avez fait appel à `echo`, `print` ou à toute autre fonction d'affichage, le délimiteur est envoyé. Il vous sera alors impossible dans la suite de faire appel à une fonction agissant sur les en-têtes comme `header()`, `set_cookie()` ou `session_start()`.

Une erreur classique est de laisser un espace ou une ligne vide entre le début du fichier et l'ouverture de bloc PHP. Ce caractère blanc est alors envoyé à l'affichage avant exécution du code PHP et utiliser une des fonctions précitées renverra une erreur ou un avertissement.

## Variables superglobales

Nous venons de voir, au travers du chapitre précédent sur les formulaires, quatre superglobales (`$_REQUEST`, `$_GET[]`, `$_POST[]` et `$_FILES[]`). Leur nom obéit à deux règles :

- il est constitué uniquement de majuscules ;
- il est préfixé par un signe de soulignement (*underscore*, « `_` »), la variable `$GLOBALS[]` exceptée.

Il existe deux autres tableaux superglobaux dont nous n'avons pas parlé : `$_SERVER[]` et `$_ENV[]`. Ces variables recueillent des informations sur le contexte de l'exécution :

- `$_SERVER[]` contient tout ce qui concerne la réception de la requête (informations sur le serveur, sur la connexion, paramètres, etc.) ;
- `$_ENV[]` contient les variables d'environnement du système dans lequel tourne PHP.

Le contenu de ces tableaux est fortement dépendant du système d'exploitation, du serveur web et de la configuration utilisés. Nous allons décrire les valeurs que vous retrouverez partout ou dans les configurations les plus répandues.

Si vous souhaitez utiliser une valeur non décrite dans ce livre, assurez-vous qu'elle est présente sur tous les types de serveur où vous pourriez être amené à utiliser vos scripts. Pour connaître les différentes valeurs présentes sur votre configuration, il vous suffit de parcourir ces tableaux. Le code suivant peut vous y aider :

```
<?php
foreach($_SERVER as $key => $value) {
    echo "\$_SERVER['$key'] = $value <br />" ;
}
foreach($_ENV as $key => $value) {
    echo "\$_ENV['$key'] = $value <br />" ;
}
?>
```

## Informations sur le serveur

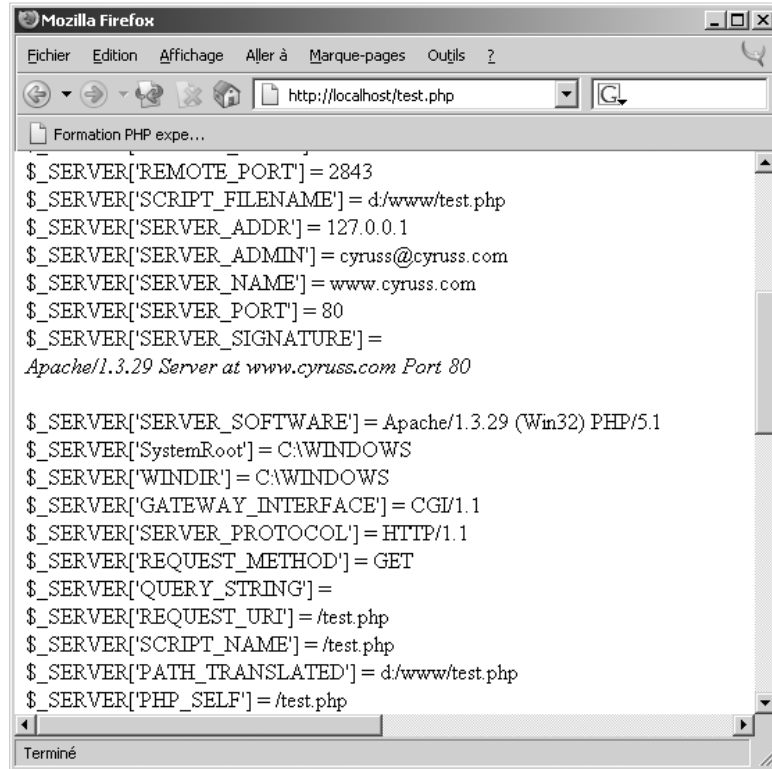
Dans le cas d'une connexion à un serveur web, vous pouvez récupérer plusieurs informations concernant le serveur web lui-même. Les valeurs disponibles dépendent du logiciel



utilisé ; nous nous contentons de décrire celles du serveur le plus répandu, Apache. La figure 9-2 montre une partie des informations accessibles en rapport avec un serveur Apache sous Microsoft Windows.

Figure 9-2

Contenu de  
`$_SERVER[]`



```
$_SERVER['REMOTE_PORT'] = 2843
$_SERVER['SCRIPT_FILENAME'] = d:/www/test.php
$_SERVER['SERVER_ADDR'] = 127.0.0.1
$_SERVER['SERVER_ADMIN'] = cyruss@cyruss.com
$_SERVER['SERVER_NAME'] = www.cyruss.com
$_SERVER['SERVER_PORT'] = 80
$_SERVER['SERVER_SIGNATURE'] =
Apache/1.3.29 Server at www.cyruss.com Port 80

$_SERVER['SERVER_SOFTWARE'] = Apache/1.3.29 (Win32) PHP/5.1
$_SERVER['SystemRoot'] = C:\WINDOWS
$_SERVER['WINDIR'] = C:\WINDOWS
$_SERVER['GATEWAY_INTERFACE'] = CGI/1.1
$_SERVER['SERVER_PROTOCOL'] = HTTP/1.1
$_SERVER['REQUEST_METHOD'] = GET
$_SERVER['QUERY_STRING'] =
$_SERVER['REQUEST_URI'] = /test.php
$_SERVER['SCRIPT_NAME'] = /test.php
$_SERVER['PATH_TRANSLATED'] = d:/www/test.php
$_SERVER['PHP_SELF'] = /test.php
```

## Nom du serveur

Le nom du serveur web est disponible dans la variable `$_SERVER['SERVER_NAME']`. Si votre serveur peut avoir plusieurs noms ou alias, il s'agit du nom canonique du serveur ou hôte virtuel et pas forcément de celui utilisé par le navigateur pour vous joindre.

## Racine du serveur

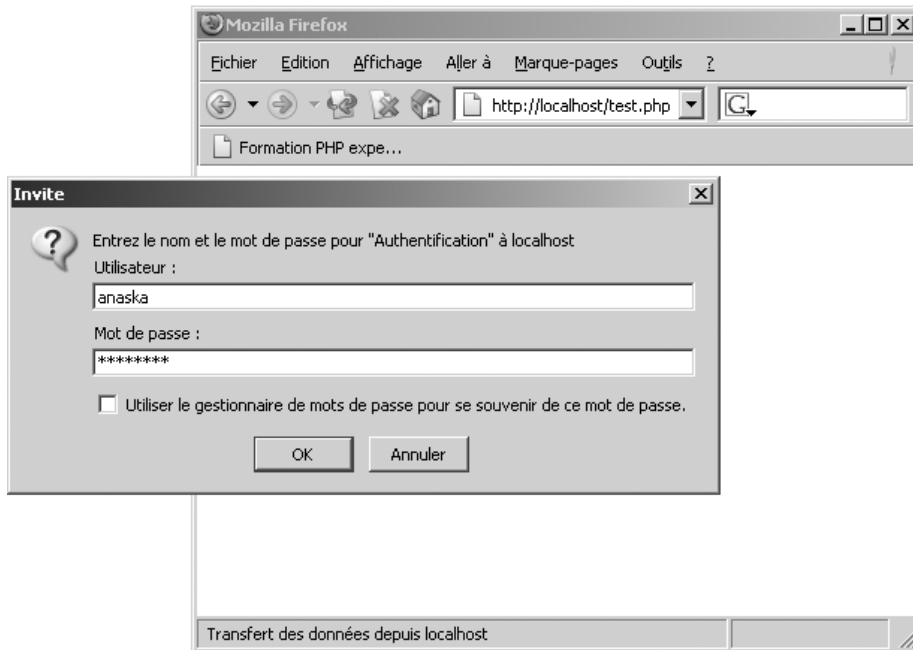
La racine du serveur web est le répertoire où sont stockées vos pages. Connaître cette valeur permet de connaître l'adresse de certains de vos fichiers et donc de les inclure facilement. De plus, en utilisant cette variable, vous restez indépendant face à l'installation : si vous changez de serveur, vos scripts utiliseront le nouveau chemin sans avoir besoin de modification. L'adresse de ce répertoire racine est disponible dans la variable `$_SERVER['DOCUMENT_ROOT']`.

## Autres informations sur le serveur

D'autres informations sur le serveur, moins utiles, sont aussi accessibles, telles que l'adresse électronique de l'administrateur défini dans la configuration Apache (`$_SERVER['SERVER_ADMIN']`) ou le numéro de version du serveur web, le nom et la version des différents modules utilisés (`$_SERVER['SERVER_SOFTWARE']`).

## Authentification HTTP

Si vous vous servez de PHP en tant que module Apache (et non en CGI), vous pouvez utiliser les systèmes d'authentification HTTP. Cette procédure vous est probablement connue ; il s'agit des boîtes de dialogue du navigateur vous demandant un nom d'utilisateur et un mot de passe pour accéder à une page (voir figure 9-3).



**Figure 9-3**  
*Authentification HTTP*

Ces demandes d'authentification sont gérées de manière transparente entre le navigateur et le serveur web. Votre serveur peut d'ores et déjà, sans PHP, utiliser de telles authentifications. Les gérer avec PHP vous permettra toutefois de gérer plus finement les accès ou d'utiliser d'autres supports de stockage pour les mots de passe que celui qu'offre votre serveur web.

## Principes du protocole HTTP

Le fonctionnement des authentifications HTTP est simple, il se repose entièrement sur les codes de retour dans les en-têtes. Le code classique 200 indique que le navigateur recevra la page normalement. Si le serveur renvoie le code 401, c'est que les informations fournies par le navigateur sont insuffisantes pour lui donner accès.

Le navigateur demandera alors à l'utilisateur les nom et mot de passe de la page. Une fois ceux-ci fournis, le navigateur redemande la page au serveur web en fournissant ces informations dans les en-têtes.

Si l'accès est toujours refusé, le navigateur redemande nom et mot de passe jusqu'à l'abandon de l'utilisateur. Si l'accès est autorisé, les nom et mot de passe sont retenus en interne par le navigateur et renvoyés avec chaque demande de page.

### Défauts et avantages

Utiliser les authentifications HTTP a un avantage majeur : le protocole à utiliser est standardisé. La plupart des applicatifs qui sont amenés à utiliser le protocole HTTP permettent de définir un mot de passe par ce biais. Si vous aviez un système personnel reposant sur des cookies ou des sessions, il vous faudrait modifier toutes vos applications pour qu'elles sachent utiliser votre système. Vous avez un avantage de compatibilité et de pérennité de la solution.

Le défaut tient au modèle des connexions HTTP. Comme il n'y a pas de persistance des connexions ou d'informations de contexte (relier une requête aux précédentes requêtes de l'utilisateur), le navigateur renvoie le mot de passe à chaque demande de page. Il vous faudra le vérifier à chaque page, sans pouvoir retenir si l'utilisateur a déjà été authentifié ou pas.

### Authentification dans l'URI

Vous rencontrerez peut-être parfois des adresses de pages contenant `nom:motdepasse@` avant le nom de domaine. Il s'agit d'une syntaxe pour prédéfinir les nom et mot de passe à utiliser par le navigateur sans qu'il nous les demande. La partie précédant les deux points est le nom d'utilisateur, la partie entre les deux points et l'arobase est le mot de passe. En voici un exemple :

■ `http://cyruss:barfoo@www.phpteam.net/test.php`

## Gestion avec PHP

La gestion des authentifications HTTP avec PHP se fait directement via la fonction `header()`, permettant d'envoyer des en-têtes HTTP. Les informations venant du navigateur sont, elles, stockées dans le tableau superglobale `$_SERVER[]` :

- `$_SERVER['PHP_AUTH_USER']` représente le nom d'utilisateur.
- `$_SERVER['PHP_AUTH_PW']` représente le mot de passe.

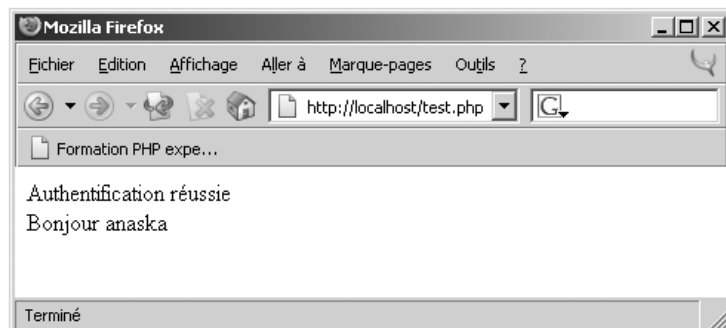
Il suffit alors de vérifier si ces deux informations sont présentes et valides. Si ce n'est pas le cas, on envoie les en-têtes HTTP de demande d'authentification.

```
<?php
function verifieMotDePasse($login, $pass){
    // Vérifie que le couple login/pass est correct
    return 1;
}

if ( !isset( $_SERVER['PHP_AUTH_USER'] )
    || !isset( $_SERVER['PHP_AUTH_PW'] )
    || !verifieMotDePasse($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW'])
    ) {
    // Soit les informations ne sont pas présentes,
    // soit elles sont erronées
    // On demande donc plus d'informations au navigateur
    $titre = 'Authentification';
    header('WWW-Authenticate: Basic realm="'. $titre. '"');
    header('Unauthorized', TRUE, 401);
    // Texte à envoyer si l'utilisateur refuse de s'authentifier
    echo "vous n'avez pas accès à cette page" ;
    // On arrête l'exécution
    exit() ;
}
// L'authentification a réussi
// On peut mettre le reste de la page comme habituellement
echo 'Authentification réussie<br>' ;
echo 'Bonjour ', $_SERVER['PHP_AUTH_USER'] ;
?>
```

Une fois l'authentification réussie, vous aurez accès aux nom d'utilisateur et mot de passe sur toutes les pages (figure 9-4).

**Figure 9-4**  
*Authentification  
HTTP*



### Durée de l'authentification et déconnexion

Par défaut, la plupart des navigateurs réutilisent les nom et mot de passe fonctionnels sur toutes les pages d'un même domaine tant qu'on ne ferme pas complètement toutes les

fenêtres de navigation. Ce comportement est géré par le navigateur et nous ne pouvons pas intervenir avec PHP.

Si toutefois vous souhaitez fournir un moyen à l'utilisateur de perdre ses droits d'accès (par exemple, quand il quitte son poste et le laisse accessible à d'autres), il vous suffit de renvoyer les en-têtes d'authentification au navigateur. Sur la plupart des logiciels (mais pas forcément tous), le mot de passe sera alors effacé et l'utilisateur devra le ressaisir.

```
function deconnecteHTTP() {  
    $titre = 'Authentification';  
    header('WWW-Authenticate: Basic realm="'. $titre. '"');  
    header('Unauthorized', TRUE, 401);  
    exit ;  
}
```

## *Authentification par le serveur web*

Vous pouvez souhaiter passer par les systèmes de contrôle d'accès classiques de votre serveur web (par exemple la directive `Require valid-user` d'Apache) plutôt que gérer les authentifications avec PHP. Dans ce cas, si PHP tourne en mode CGI ou que la directive de configuration `safe_mode` soit activée dans votre `php.ini`, la variable `$_SERVER['PHP_AUTH_PW']` ne contiendra pas le mot de passe donné par le visiteur. PHP le masque afin qu'un administrateur puisse gérer les mots de passe lui-même sans que le développeur en ait connaissance.

Dans ce cas, si l'authentification est réussie (donc si votre script est exécuté), vous pouvez toutefois lire le nom de l'utilisateur via la variable `$_SERVER['REMOTE_USER']`.

## Paramètres de la connexion

### *Adresse IP et port du client*

Dans le cas d'une connexion HTTP, PHP remplit automatiquement dans `$_SERVER[]` les adresses IP et ports utilisés. Ces informations sont dans `$_SERVER['REMOTE_ADDR']` et `$_SERVER['REMOTE_PORT']`.

```
<?php  
echo 'Adresse du serveur :', $_SERVER['REMOTE_ADDR'],'<br>';  
echo 'Port utilisé par le serveur :', $_SERVER['REMOTE_PORT'];  
?>
```

Les deux usages principaux de cette information sont :

- les fichiers de logs, pour pouvoir identifier un utilisateur malveillant et porter plainte en cas de fraude ou d'activité illégale ;
- la redirection sur un miroir géographiquement proche du client.

Cette dernière fonctionnalité est par exemple utilisée sur le site officiel de PHP. Grâce à l'IP, le site choisit automatiquement le miroir le plus proche et vous y redirige afin de soulager le site central et le réseau.

**Note**

Si vous utilisez une adresse IP dans votre application, pensez que dans le futur, il sera probablement possible de voir apparaître des IPv6, donc de la forme 2001:7a8:2f99:0:0:0:0:1 (chaîne de 8 parties de 4 chiffres hexadécimaux, séparés par le caractère deux points).

**Présence de proxy**

Toutefois, il faut bien se rendre compte que ce sont les adresses visibles du serveur. Si le client passe par une passerelle ou un *proxy*, c'est l'adresse et le port utilisés par le proxy qui seront visibles et uniquement ceux-ci.

Certains proxies informent cependant votre serveur sur l'adresse IP réelle du client. Vous pouvez alors la lire, quand elle est fournie, dans `$_SERVER['X_FORWARDED_FOR']`. Cette adresse réelle sera le plus souvent une adresse privée (par exemple 192.168.0.1) qui ne vous permettra pas de joindre directement la personne et qui ne sera pas unique.

**Sécurité de l'identification par adresse IP**

Il est important de noter que ces valeurs sont toutes des valeurs fournies par le client. Il est techniquement facile pour quelqu'un d'affirmer être un proxy qui redirige une autre adresse, même si ce n'est pas vrai. Il est aussi techniquement possible de se faire passer pour quelqu'un d'autre. On ne doit donc jamais authentifier quelqu'un uniquement par son adresse IP.

De même, rien ne garantit l'unicité d'une adresse IP récupérée ainsi. Certains fournisseurs d'accès Internet réutilisent très rapidement les adresses une fois qu'un client se déconnecte. Il est techniquement possible de voir deux clients différents avec la même IP se connecter sur votre site à moins de cinq minutes d'intervalles. Il est encore plus fréquent de voir deux clients différents simultanément avec la même adresse. Généralement deux personnes d'une même société, école ou bibliothèque qui passent par un proxy (qui ne transmettra pas forcément l'IP réelle) peuvent avoir la même adresse IP. Ces cas sont très fréquents car les adresses de sites web se partagent souvent entre collègues de bureau, engendrant des visites presque simultanées.

Pour règle générale, vous ne pouvez pas donner foi à cette information pour identifier un utilisateur ou réaliser un contrôle d'accès.

**Nom d'hôte**

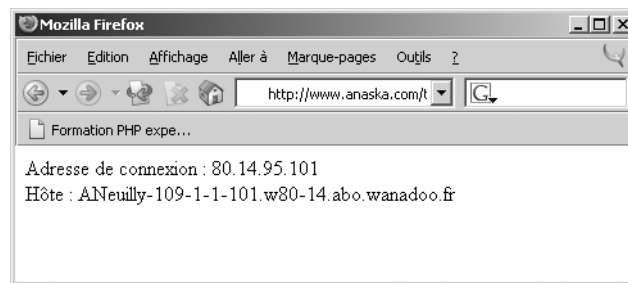
Selon la configuration du serveur, il est aussi possible que vous ayez le nom associé à l'adresse -IP du client dans `$_SERVER['REMOTE_HOST']`. Dans le cas contraire, il est de toute façon possible de le connaître en utilisant la fonction `gethostbyaddr()` et l'adresse IP (cette fonction fait une requête complète vers le serveur DNS et peut donc être longue).

Le résultat du code exemple suivant est donné à la figure 9-5.

```
<?php
$ip = $_SERVER['REMOTE_ADDR'] ;
if (isset($_SERVER['REMOTE_HOST'])) {
    $host = $_SERVER['REMOTE_HOST'] ;
} else {
    $host = gethostbyaddr($ip) ;
}
echo "Adresse de connexion : $ip <br>" ;
echo "Hôte : $host" ;
?>
```

Figure 9-5

*Adresse IP et hôte*



## Adresse IP et port du serveur

Contrairement à l'adresse du navigateur client, l'adresse IP du serveur est une information fiable et non falsifiable. L'adresse IP est accessible dans `$_SERVER['SERVER_ADDR']` et le port (probablement 80 pour une connexion HTTP ou 443 pour une connexion sécurisée HTTPS) dans `$_SERVER['SERVER_PORT']`.

Dans le cas où votre serveur aurait plusieurs adresses disponibles, c'est celle utilisée par le navigateur pour vous joindre qui serait renvoyée. En particulier, cela sous-entend que si vous vous connectez en local à votre serveur, cette variable ne vous permettra pas de connaître votre adresse IP publique.

## Description de la requête HTTP

Les informations les plus importantes sont probablement celles sur la requête en cours sur le client.

Les différentes explications définissent des moyens d'accès directs à certaines informations. Si vous n'y retrouvez pas tout ce que vous cherchez ou si vous voulez traiter ces informations à la main, utilisez une fonction qui renvoie tous les en-têtes reçus : `apache_request_headers()`.

### Note

Cette fonction ne marche qu'avec le moteur PHP compilé en module Apache.

## Paramètres de la requête

### Méthode d'accès invoquée

La superglobale `$_SERVER['REQUEST_METHOD']` permet de connaître la méthode invoquée pour accéder à la page. Généralement, la méthode utilisée sera la méthode GET. Si une page réceptionne un formulaire, il sera alors indiqué soit GET, soit POST selon la valeur de l'attribut `method` de la balise `<form>`.

```
■ echo $_SERVER['REQUEST_METHOD'];
```

### Serveur demandé

Les serveurs web sont généralement configurés pour pouvoir servir plusieurs sites (on parle alors d'hôtes virtuels) et plusieurs domaines ou sous-domaines.

Plus haut, nous avons vu que le nom de l'hôte virtuel utilisé était donné par `$_SERVER['SERVER_NAME']`. Le nom du serveur utilisé par le navigateur (celui qui apparaît dans sa barre d'adresse) est, lui, donné par `$_SERVER['HTTP_HOST']`.

### Protocole utilisé

Le protocole utilisé pour accéder au serveur est disponible dans la variable `$_SERVER['SERVER_PROTOCOL']`. Le plus souvent, cette valeur est HTTP/1.1.

Cette information est particulièrement utile si vous envoyez vous-même certains en-têtes, par exemple des directives de cache (voir le chapitre sur les caches, section sur les caches HTTP). Dans ce cas, il faut pouvoir faire la différence entre les versions 1.0 et 1.1 du protocole HTTP.

## L'adresse demandée (URL)

L'adresse de la page demandée est disponible dans la variable `$_SERVER['REQUEST_URI']`. Elle contiendra tout ce qui est après le nom de domaine (répertoire, page, extension et paramètres).

Il est possible d'envoyer des paramètres à une page via l'URL (adresse de la page). Dans ce cas, ils sont visibles après un point d'interrogation. C'est notamment la technique utilisée dans les formulaires qui utilisent la méthode GET. Une autre méthode est de remplir directement les informations après le nom du script, c'est une méthode souvent utilisée pour avoir des adresses virtuelles.

### Chaînes de GET

Les paramètres envoyés dans une chaîne de GET (ce qui est après le point d'interrogation suivant le nom de la page dans l'adresse) sont automatiquement décodés dans la superglobale `$_GET[]`. Si toutefois vous souhaitez récupérer manuellement ces paramètres pour les interpréter ou les réutiliser, vous pouvez y accéder via `$_SERVER['QUERY_STRING']`.



Par exemple, une adresse `http://php.net/manual-lookup.php?pattern=string&lang=fr` renverra `?pattern=string&lang=fr` comme chaîne de paramètres.

### Codage des paramètres

Les différents paramètres sont du type `nom=valeur`, séparés par le « et commercial » (caractère `&`). Les caractères autres que les caractères alphanumériques peuvent être codés avec une syntaxe `%xx` où `xx` est le code hexadécimal du caractère.

La fonction `rawurlencode()` permet de convertir les caractères spéciaux d'une chaîne vers cette syntaxe, la fonction `rawurldecode()` fait l'opération inverse. Les fonctions `urlencode()` et `urldecode()` sont similaires mais codent l'espace en le remplaçant par le signe plus (+). C'est cette dernière syntaxe qui est normalement à utiliser dans le contexte Web.

Ainsi, une fonction pour décoder les chaînes de paramètres pourrait être :

```
$results = array() ;
$query = $_SERVER['QUERY_STRING'] ;
$params = explode('&', $query) ;
foreach($params as $param) {
    $param = explode('=', $param) ;
    $key = urldecode($param[0]) ;
    $value = urldecode(@$param[1]) ;
    $results[$key] = $value ;
}
```

PHP met à disposition la fonction `parse_str()`, qui fait ces opérations automatiquement :

```
// URL : http://php.net/manual-lookup.php?pattern=string&lang=fr
$query = $_SERVER['QUERY_STRING'] ;
$result = array() ;
parse_str($query, $result) ;
echo $result['pattern'] ; // affiche: string
echo $result['lang'] ; // affiche: fr
```

### Chemin d'accès

Selon votre configuration, il vous sera peut-être possible d'avoir des adresses du type `script.php/chemin/page?parametres` voire `script/chemin/page?parametres` (pas d'extension `.php` dans l'adresse). Un tel système permet d'avoir une architecture virtuelle, et de dissocier l'emplacement et le nom des scripts sur le serveur des adresses utilisées.

Quand une telle méthode est utilisée, vous pourrez retrouver la partie virtuelle (ici `/chemin/page?parametres`) dans la variable superglobale `$_SERVER['PATH_INFO']`.

## Informations fournies par le client

Les navigateurs envoient avec les en-têtes de la requête plusieurs informations. Ces informations sont dites « non fiables » car rien ne garantit que le client qui a fait la requête ne mente pas. Pourtant, elles permettent souvent de récolter quelques statistiques ou de servir un contenu plus adapté au navigateur.

## Page référente

Le plus souvent, le navigateur envoie avec sa demande l'adresse de la dernière page où il est allé. Cette information est enregistrée par PHP dans `$_SERVER['HTTP_REFERER']`.

Connaître cette valeur peut être très intéressant à des fins statistiques, pour savoir qui vous référence, ou par quels mots-clés on trouve votre site sur les moteurs de recherche.

Il est aussi possible de restreindre certaines pages, images ou certains fichiers sur votre serveur à l'aide de cette variable, pour être sûr que personne ne les référence directement sans faire passer par votre site. Cette démarche est toutefois déconseillée car tous les navigateurs n'envoient pas dans toutes les situations l'adresse de la page référente (notamment pour des raisons de sécurité ou de vie privée) ; vos ressources seraient alors indisponibles pour ces gens-là. De plus, c'est une information très simple à falsifier pour le client, elle n'offre donc aucune garantie.

## Négociation de contenu

Depuis HTTP 1.1, les clients web envoient diverses informations sur les contenus qu'ils savent gérer ou qu'ils préfèrent recevoir. La réception des en-têtes concernés permet au serveur de choisir le bon contenu à renvoyer. Parmi ces en-têtes, on trouve :

- Une négociation de format (valeur présente dans la variable `$_SERVER['HTTP_ACCEPT']`). Elle permet de déclarer les différents formats acceptés et de faire sélectionner automatiquement le plus adapté par le serveur. Ainsi, un navigateur pourrait envoyer la chaîne `text/xml,text/html;q=0.8,*/*;q=0.1`, ce qui signifierait qu'il préfère recevoir du XML (priorité 1.0), sinon du HTML (priorité 0.8) et à défaut, il acceptera ce qui est disponible (priorité 0.1). Récupérer cette chaîne permet de servir un format différent selon les choix de l'utilisateur (par exemple choisir automatiquement entre XHTML et HTML).
- Une négociation de langue (grâce à la valeur présente dans `$_SERVER['HTTP_ACCEPT_LANGUAGE']`). Cette donnée permet de choisir la langue à utiliser sur la page et de pouvoir distribuer une page dans la langue du visiteur. C'est par exemple l'information utilisée dans la documentation en ligne du site officiel PHP. Selon la configuration de votre navigateur, le manuel sera automatiquement affiché en français, en anglais, ou dans une autre langue disponible.
- Une négociation de jeux de caractères (via la variable `$_SERVER['HTTP_ACCEPT_CHARSET']`). Le format de la chaîne reçue, comme les suivantes, est similaire à la précédente.
- Une négociation de compression (via la valeur `$_SERVER['HTTP_ACCEPT_ENCODING']`). Cette information permet de savoir si le navigateur accepte les envois de pages compressées ou non.

## Nom et version du navigateur

Les navigateurs envoient presque tous leurs nom et numéro de version dans la requête. La chaîne envoyée par le navigateur est disponible dans `$_SERVER['USER_AGENT']`.

Ces informations sont très utiles à des fins statistiques (par exemple, remarquer quelques organisateurs de poche peut amener à faire une version spécifique plus adaptée). De telles valeurs peuvent aussi servir à reconnaître les aspirateurs web et limiter l'impact qu'aurait leur passage.

Certains webmasters les utilisent aussi pour servir des versions de la page différentes selon le navigateur. Cette procédure est toutefois largement déconseillée, car elle est généralement très imparfaite : le nombre de navigateurs différents rend impossible une détection exhaustive et beaucoup de navigateurs mentent sur cette information. Même le portail MSN s'est fait épingler plusieurs fois pour servir des pages illisibles à certains navigateurs en les détectant mal alors que la page classique aurait été parfaitement lue. De plus, cette optique oblige à maintenir une grande partie des pages en plusieurs exemplaires. Si vous n'avez pas d'importantes ressources à consacrer uniquement à cette tâche et à la maintenance d'un tel système, ce procédé est à éviter.

## Environnement système

Les variables d'environnement du système sont accessibles à travers la superglobale `$_ENV[]`. Les valeurs accessibles dépendent entièrement de votre système, nous ne pouvons donc pas vous donner de liste des valeurs intéressantes. De plus, sur un système en `safe_mode`, les variables d'environnement accessibles en lecture et en écriture sont habituellement restreintes à un jeu très limité.

D'autres variables sont toutefois accessibles par d'autres moyens. Ainsi, vous pouvez connaître la variable d'environnement `PATH` (qui définit les chemins de recherche lors d'une exécution) avec `$_SERVER['PATH']`.

De même, vous avez accès aux informations sur le processus ou l'utilisateur courant via les fonctions POSIX : `posix_getuid()` et `posix_getgid()` donnent respectivement l'identifiant de l'utilisateur et celui du groupe en cours, `posix_uname()` donne le nom du système d'exploitation.

## Nom du script exécuté

La superglobale `$_SERVER[]` nous donne aussi quelques informations sur le script appelé.

La variable `$_SERVER['SCRIPT_NAME']` donne par exemple l'adresse du script relative à la racine du serveur web. La variable `$_SERVER['PATH_TRANSLATED']` donne la même information, mais avec une adresse absolue sur le système de fichiers.

### Note

Ces informations concernent le script exécuté en premier et non les scripts inclus avec `include()` ou `require()`.

Il existe de plus des mots-clés pour accès rapide. Ainsi `__FILE__` définit le chemin complet sur le système de fichiers, `__LINE__` la ligne en cours d'exécution, `__CLASS__`, `__FUNCTION__`

et `__METHOD__` les éventuels noms de classe, de fonction et de méthode dans lesquelles se trouve la ligne en cours d'exécution. Ces mots-clés ont une syntaxe de constante mais n'en sont pas puisque la valeur change au cours de l'exécution.

## Interactions PHP/JavaScript

Une des recherches les plus souvent faites dans le développement web par ceux qui n'ont pas l'habitude de ce contexte est l'intégration de PHP et JavaScript ou PHP et Flash, par exemple utiliser une variable PHP en JavaScript, faire appel à une fonction PHP en JavaScript, ou le contraire.

Ce type d'interaction est malheureusement impossible de par la manière dont fonctionne PHP. Les deux langages ne s'exécutent pas pendant la même étape de l'échange HTTP : PHP s'exécute sur le serveur avant d'envoyer la page, JavaScript s'exécute chez le client une fois la page téléchargée. Ainsi, quand PHP fait ses opérations, le concept même de JavaScript n'existe pas, il n'y a que du texte brut sans sens renvoyé à Apache. Inversement, quand JavaScript s'exécute, PHP a complètement fini son travail en envoyant la page et il n'y a plus moyen d'accéder à un quelconque objet du langage PHP.

Pourtant, des interactions sont parfois utiles, et certaines sont possibles. De la même façon qu'on fait du HTML dynamique, il est possible de faire du JavaScript dynamique. Ainsi, PHP produit du texte envoyé au navigateur. Il lui est possible de modifier le JavaScript créé pour changer le contenu de certaines fonctions ou initialiser certaines valeurs. Le texte créé par le PHP et envoyé au serveur contient habituellement du HTML, mais peut également contenir du JavaScript ou toute autre donnée interprétable.

De même, JavaScript peut très bien faire une requête complète vers le serveur avec des paramètres, et lire la réponse donnée par PHP. Il s'agit cependant là d'une action coûteuse en temps, car il y a un aller et retour jusqu'au serveur.

## Ligne de commande

Comme nous l'avons vu en introduction, PHP ne permet pas uniquement de travailler en mode client-serveur. On peut également l'utiliser comme un langage de programmation en ligne de commande. Dans ce cadre, deux superglobales d'environnement nous permettent de connaître les informations passées en paramètres.

### *Lecture des arguments*

La superglobale `$_SERVER['argv']` trouvera principalement son utilité dans le cadre de l'utilisation de PHP en tant que langage en ligne de commande. Il s'agit d'un tableau des arguments passés au script.

Une exécution par :

```
php script.php nom=rasmus prenom= lerdorf
```

avec le script suivant :

```
<?php
print_r($_SERVER[argv]);
?>
```

Renverra :

```
Array
(
    [0] => script.php
    [1] => nom=rasmus
    [2] => prenom=lerdorf
)
```

### ***Nombre d'arguments***

La superglobale `$_SERVER['argc']` indique le nombre de paramètres passés au script dans le cas d'une utilisation en ligne de commande. L'appel de l'exemple précédent nous renverra 3.

# 10

## Les cookies

---

Les *cookies* permettent de retenir des informations sur un utilisateur : vous pouvez enregistrer des données qui seront associées à un visiteur particulier. Les utilisations les plus fréquentes des cookies sont :

- se souvenir du nom d'un utilisateur pour lui éviter de le ressaisir lors de sa prochaine authentification ;
- se souvenir des informations saisies dans un formulaire pour éviter de les redemander ou pour pré-remplir le formulaire la prochaine fois ;
- identifier chaque utilisateur de façon unique lors de ses visites à des fins statistiques.

### Présentation

Les cookies ont été conçus par la société Netscape afin d'étendre les fonctionnalités du protocole HTTP et de lui ajouter la possibilité d'établir un lien entre les différentes requêtes. Ils ont été par la suite intégrés au protocole ; tous les navigateurs actuels prennent en charge les cookies.

Les cookies sont des fichiers texte courts stockés par le navigateur sur l'ordinateur de l'utilisateur, à la demande du serveur web. Pour faire une analogie, le cookie ressemble à une carte d'identité : l'administration vous donne une carte avec des informations vous concernant et vous demande de la représenter régulièrement. Grâce à cette carte, elle peut vous identifier chaque fois qu'il est nécessaire et connaître quelques informations sur vous, votre âge par exemple. Le cookie est l'équivalent de ce principe pour les pages web : le serveur vous envoie une valeur (le cookie) avec la page et vous demande de la renvoyer dans vos prochains échanges. Grâce à cette valeur, le serveur peut retenir des

informations vous concernant. Cette demande se fait dans les en-têtes HTTP, avant l'envoi de la page web. Une illustration de ce fonctionnement se trouve en figure 10-1.

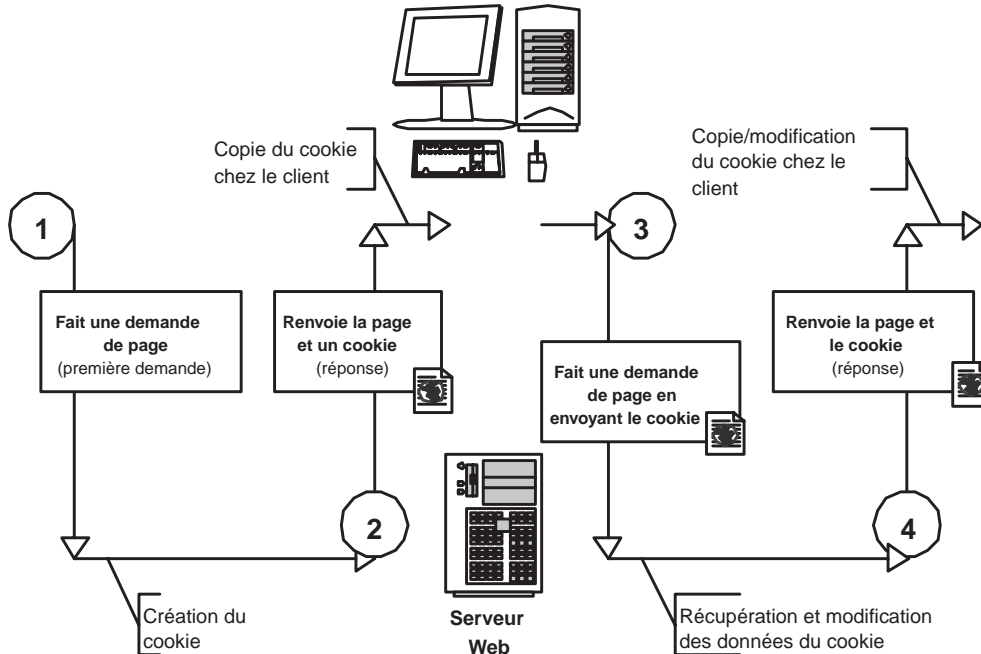


Figure 10-1

*Envoi et réception d'un cookie*

Lorsque vous envoyez un cookie, vous demandez au navigateur de le renvoyer dans ses prochaines requêtes. Il a toutefois la possibilité de refuser et de ne pas le faire. La plupart des navigateurs ont une option qui permet de refuser les cookies. Microsoft Internet Explorer, à partir de sa version 6.0, filtre automatiquement certains cookies. Si vos tests échouent quand vous manipulez des cookies, vous pouvez vérifier si c'est le navigateur qui refuse volontairement votre cookie, par la présence d'une icône de sens interdit dans la barre en bas à droite lors du chargement de la page. Dans ce cas, vous devrez régler la configuration de votre navigateur lors de vos tests.

### Forme du cookie sur votre ordinateur

Comme indiqué plus en amont, les cookies sont stockés sur l'ordinateur du client. Dans le cas d'un cookie sans durée d'expiration, les informations sont stockées dans la mémoire vive de l'ordinateur. En revanche, si vous lui donnez une durée de vie, ce qui est généralement le cas, un fichier est créé sur le disque dur du client contenant ses propres informations.

Microsoft Internet Explorer pour Windows stocke les cookies dans un dossier temporaire Internet (C:\Documents and Settings\ votre\_login \Local Settings\Temporary Internet Files sous Microsoft Windows 2000 et suivants). Il s'agit de simples fichiers texte que vous pouvez lire avec le Bloc-notes.

```
setcookie('PHP', 5, mktime ( 12, 34, 00, 04, 01, 2030), '/php5' );
```

Voici ce que contient par exemple le fichier texte du cookie créé avec la commande précédente :

```
PHP
5
www.example.com/php5
1024
449747968
31538642
3962392528
29605282
```

La première ligne contient le nom du cookie, la deuxième la valeur. Le texte `www.example.com/php5` correspond à la concaténation du domaine et du chemin de validité du cookie. Les lignes numériques suivantes contiennent entre autres les paramètres du cookie (par exemple, s'il doit être envoyé uniquement pour une connexion sécurisée ou non), la date d'expiration, la date de création et la date de modification.

Les systèmes de vos visiteurs n'étant pas forcément bien sécurisés, il faut éviter d'y stocker des informations confidentielles comme des mots de passe : elles pourraient être lues plus ou moins facilement.

#### Note

Sur d'autres systèmes ou d'autres navigateurs, les cookies pourront être stockés sous une toute autre forme. Il n'y a pas de convention et chaque éditeur maintient son propre format de stockage.

## Lecture et écriture d'un cookie

Toute la gestion des cookies se fait avec une unique fonction, `setcookie()`. Son utilisation simple ne nécessite que deux paramètres : le nom du cookie et sa valeur.

```
setcookie(nom, valeur)
```

### Envoi d'un cookie

PHP permet de gérer entièrement l'envoi et la réception des cookies via la fonction `setcookie()`.



Cette facilité d'utilisation ne doit pas vous faire perdre de vue que les cookies se gèrent dans les en-têtes envoyés avant la page web. Le serveur web envoie l'ensemble des en-têtes dès qu'il reçoit du texte à afficher de la part du script. En conséquence, cette fonction ne doit être utilisée qu'en haut de vos pages, avant tout contenu HTML.

#### Remarque

Si vous avez le message d'erreur Warning: Cannot send session cookie - headers already sent, c'est probablement que vous avez oublié cette règle. Peut être qu'une ligne vide ou des espaces se sont glissés entre le début de la page et l'ouverture du bloc PHP.

Voici un exemple simple d'envoi de cookie :

```
<?php
    // Attention :
    // aucun texte HTML ne doit être envoyé avant le cookie.
    setcookie('langage', 'PHP version 5') ;
?>
<html>
  <head><title>titre</title></head>
  <body>
    <p>Un cookie a été envoyé</p>
    <p>Son nom est : langage</p>
    <p>Son contenu est : PHP version 5</p>
  </body>
</html>
```

## Lecture d'un cookie

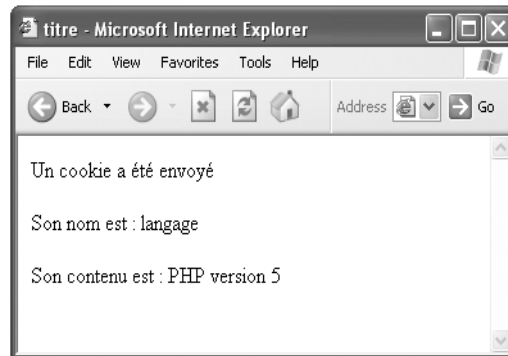
Si vous avez testé l'exemple précédent, la prochaine fois que le navigateur chargera une page sur votre site, il renverra le cookie dont le nom est langage. Il nous reste donc à savoir relire cette information. Encore une fois, tout est déjà fait dans PHP et vous pouvez accéder à tous les cookies envoyés, grâce au tableau `$_COOKIE[]`. Il s'agit d'une des variables superglobales ; vous pouvez donc vous en servir sans vous soucier de sa portée (retournez aux chapitres 8 et 9 pour plus de détails sur les superglobales).

Vous pouvez consulter le résultat de l'exemple suivant à la figure 10-2.

```
<html>
  <head><title>titre</title></head>
  <body>
    <?php
      // On vérifie si le cookie a été reçu
      if ( isset($_COOKIE['langage']) ) {
```

```
// C'est le cas, le cookie existe
echo '<p>Un cookie a été envoyé</p>' ;
echo '<p>Son nom est : langage</p>' ;
echo '<p>Son contenu est : ' ;
// On lit la valeur du cookie et on l'affiche
echo $_COOKIE['langage'] ;
echo '</p>' ;
} else {
// Le cookie n'a pas été reçu
echo '<p>Aucun cookie du nom de langage n'a été reçu</p>' ;
}
?>
</body>
</html>
```

**Figure 10-2**  
*Utilisation  
d'un cookie*



#### Astuce

Si vous cherchez à savoir quels cookies sont utilisés, lisez simplement le tableau `$_COOKIE[]` : ils y sont tous listés.

Le tableau `$_COOKIE[]` est en lecture seule : ajouter un élément n'enverra pas de cookie au navigateur. Pour envoyer un cookie, vous devez impérativement utiliser la fonction `setcookie()` ou envoyer l'en-tête HTTP correspondant à la main.

#### Remarque

Il est à noter que le tableau `$_COOKIE[]` est initialisé avant le début de l'exécution. Lors de l'envoi du cookie, aucune référence n'est créée dans le tableau `$_COOKIE[]`. Celle-ci n'est accessible que sur la page suivante, quand le cookie a été reçu et renvoyé par le navigateur.

## Suppression d'un cookie

Pour effacer un cookie, il suffit d'envoyer un cookie du même nom mais sans valeur, comme dans : `setcookie ('nom_du_cookie')`.

### Attention

Il faut bien dissocier ce que le navigateur garde en mémoire et ce qu'il nous a envoyé : si vous effacez la valeur reçue dans PHP grâce à `unset ( $_COOKIE['nom_du_cookie'] )`, le navigateur, lui, se rappelle toujours le contenu du cookie et le renverra à la prochaine requête. Inversement, si dans un script vous demandez au navigateur d'effacer son cookie, cela ne vous empêchera pas d'accéder à ce qu'il a envoyé, cette fois-ci tant que vous n'aurez pas effacé la variable PHP correspondante. Une bonne habitude est d'effacer les deux en même temps pour éviter les erreurs.

```
<?php
// Si le cookie compteur de visite existe,
if ( isset( $_COOKIE['visites'] ) ) {
    // on demande au navigateur d'effacer son cookie
    setcookie('visites') ;

    // et on en efface la valeur en local pour éviter
    // de l'utiliser par erreur dans la suite de notre script
    unset($_COOKIE['visites'] ) ;
}
?>
```

## Modifier les valeurs d'un cookie

Pour modifier un cookie, il vous suffit de refaire appel à la fonction `setcookie()` avec le nom du cookie à modifier et sa nouvelle valeur. Il remplacera le précédent de même nom.

Comme pour la suppression, pensez bien à dissocier le cookie présent sur le navigateur (que vous souhaitez mettre à jour) et la valeur présente dans `$_COOKIE[]` (qui est celle que le navigateur vous a envoyée).

Voici un exemple vous permettant de voir la modification d'un cookie. À chaque passage sur la page, la valeur du cookie s'incrémente :

```
<?php
$message = array() ;
if ( ! isset($_COOKIE['visites']) ) {
    $message[] = '$_COOKIE[\'visites\'] est vide' ;
    $message[] = 'le cookie n\'a pas été reçu par le serveur' ;
    $message[] = 'on envoie le cookie avec la valeur 1' ;
    setcookie('visites', 1 ) ;
} else {
    $message[] = '$_COOKIE[\'visites\'] n\'est pas vide' ;
    $message[] = 'la valeur reçue est ' . $_COOKIE['visites'] ;
    $message[] = 'on envoie un nouveau cookie avec la valeur '
```

```
. ( $_COOKIE['visites'] +1 ) ;
setcookie('visites', $_COOKIE['visites'] +1 ) ;
$message[] = 'le navigateur va modifier le cookie pour lui'
.' donner la nouvelle valeur ' . ( $_COOKIE['visites'] +1 ) ;
}
$message[] = 'vous pouvez recharger la page pour voir l'évolution';
echo join('<br>', $message) ;
?>
```

## Validité et date d'expiration

Vous avez peut-être remarqué, si vous avez testé les exemples précédents, que la durée de vie par défaut d'un cookie se limite à une session de navigateur. Cela signifie que quand vous fermez tous vos navigateurs et que vous retournez sur la page, le cookie est perdu. Ce comportement est dû à la date d'expiration de votre cookie ; et pour cause, vous n'en avez fourni nulle part. Par défaut, quand le navigateur ne reçoit pas de date d'expiration pour un cookie, il ne le considère comme valide que pour la navigation en cours et l'efface à la fermeture.

### Remarque

Un cookie sans date d'expiration n'est pas créé sous forme de fichier texte sur votre ordinateur : il est stocké dans la mémoire vive de l'ordinateur.

Pour notre compteur, par exemple, il est plus adapté d'avoir un cookie permanent. La permanence n'est pas réellement possible, mais nous pouvons toujours demander au navigateur une date d'expiration éloignée.

Pour définir une date d'expiration du cookie, on spécifie la date sous forme de *timestamp* en troisième paramètre à la fonction `setcookie()`. Dans notre exemple concernant un compteur de visites, nous allons utiliser la fonction `mktime()` qui est décrite au chapitre 7 traitant des fonctions usuelles.

```
<?php
// On vérifie si le cookie est présent
if ( !isset( $_COOKIE['visites'] ) ) {
    // Il n'est encore jamais passé sur la page
    // donc il n'a pas de cookie
    $visites = 1 ;
    $message = 'Vous venez pour la première fois' ;
} else {
    // Il est déjà venu, on incrémente son nombre de visites
    $visites = $_COOKIE['visites'] + 1 ;
    $message = 'Vous êtes venu ' . $_COOKIE['visites'] . ' fois' ;
}
```

```
// On met le cookie à jour avec le nouveau nombre de visites
setcookie('visites', $visites, mktime(0,0,0,12,31,2037) );
?><html><head><title>titre</title></head>
  <body>
    <p> <?php echo $message ; ?> </p>
  </body>
</html>
```

**Remarque**

La date d'expiration d'un cookie est gérée comme un *timestamp* Unix. La valeur maximale de ces dates sur la plupart des systèmes actuels est mi-janvier 2038. Contentez-vous de dates entre les années 2000 et 2037.

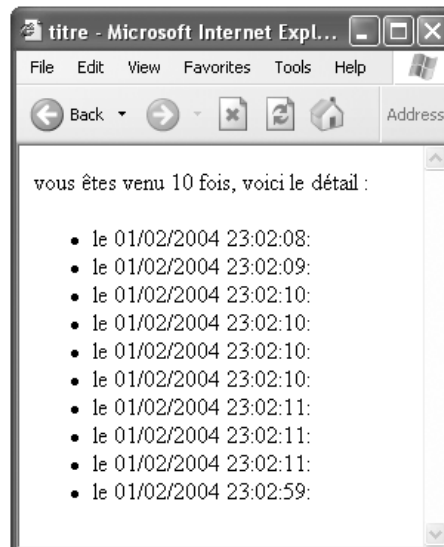
## Tableaux et types complexes

Jusqu'à présent, nous n'avons stocké que des nombres ou des chaînes de caractères dans les cookies. Si vous souhaitez stocker autre chose, par exemple un tableau, il faudra transformer vos données en une chaîne de caractères avant d'envoyer le cookie. De même, une fois votre cookie récupéré à la prochaine page, il sera nécessaire de faire la transformation inverse. Cette transformation, qui s'appelle sérialisation, est gérée par les fonctions PHP `serialize()` et `unserialize()`. La première sert à transformer une variable en une chaîne de caractères, la seconde fait l'opération contraire.

Pour illustrer cette fonctionnalité, voici un script court qui permet de faire stocker les heures de visite. Vous pouvez voir le résultat du script à la figure 10-3.

**Figure 10-3**

*Utilisation d'un cookie pour stocker un tableau*



```
<?php
// On récupère la liste actuelle
if ( isset( $_COOKIE['liste'] ) ) {
    // Si le cookie existe ,
    // on récupère la liste des heures de visite
    $liste_serialisee = $_COOKIE['liste'] ;
    // Il faut maintenant décoder le contenu
    // pour obtenir le tableau
    $liste_tableau = unserialize( $liste_serialisee ) ;
}else{
    // Si le cookie n'existe pas encore,
    // la liste est un tableau vide
    $liste_tableau = array() ;
}

// On ajoute l'heure actuelle
$liste_tableau[] = time() ;

// On renvoie le cookie avec sa nouvelle valeur
// Pour cela, on sérialise le tableau avant
$liste_serialisee = serialize( $liste_tableau ) ;
setcookie('liste', $liste_serialisee) ;

?><html><head><title>titre</title></head>
<body>
  <p> vous êtes venu
    <?php echo count( $liste_tableau ) ; ?>
    fois, voici le détail : </p>
  <ul>
    <?php foreach( $liste_tableau as $heure ) {
      echo '<li>le ',date("d/m/Y H:i:s:", $heure),'</li>';
    } ?>
  </ul>
</body>
</html>
```

## Restriction de portée du cookie

Dorénavant, vous pouvez fermer et rouvrir votre navigateur : le compteur continuera de s'incrémenter à chaque passage. Deux paramètres supplémentaires peuvent être renseignés lors de l'envoi d'un cookie. Ils permettent d'en restreindre la portée.

Le premier paramètre indique un chemin d'accès : par exemple '/manual'. Il demandera au navigateur de n'envoyer le cookie que pour les pages de ce répertoire et ses sous-répertoires. Si vous ne spécifiez pas ce paramètre, le cookie sera envoyé pour toutes les pages du domaine.

Le second paramètre indique un nom de domaine, par exemple 'www.php.net'. Il demandera au navigateur d'envoyer le cookie uniquement pour le domaine spécifié. Dans notre exemple, il l'enverra pour <http://www.php.net>, mais pas pour <http://pear.php.net>. Si vous souhaitez

que le cookie soit accessible pour tous les sous-domaines, il suffit alors d'omettre le sous-domaine (mais laisser le point devant le domaine). Par défaut, le cookie est disponible pour tout le domaine courant.

**Note**

Évidemment, spécifier un répertoire ou un domaine auquel la page actuelle n'appartient pas ne donnera rien : le navigateur refusera le cookie.

Voici quelques exemples, en considérant que le navigateur a appelé l'adresse `http://www.php.net/manual/en/function.setcookie.php` :

```
<?php
// Valide sur http://*.php.net/*
// Non renvoyé sur http://www.asp.net/
setcookie('nom','valeur');

// Identique au précédent
setcookie('nom','valeur',mktime(0,0,0,12,31,2037) , '/' , '.php.net');

// Valide sur http://*.php.net/manual/*
// Non renvoyé sur http://www.php.net/FAQ.php
setcookie('nom','valeur','/manual');

// Valide sur http://www.php.net/*
// Non renvoyé sur http://pear.php.net/
setcookie('nom','val',mktime(0,0,0,0,0,2037) , '/' , 'www.php.net');

// Valide sur http://www.php.net/manual/*
// Non renvoyé sur http://pear.php.net/manual
// Non renvoyé sur http://www.php.net/FAQ.php
setcookie('n','v',mktime(0,0,0,0,0,2037),'/manual','www.php.net');
?>
```

Un dernier paramètre est disponible, qui permet d'éviter de diffuser un cookie avec contenu sensible sur une connexion non sécurisée. Si cet argument était à `TRUE` lors de l'appel à `setcookie()` et si la connexion n'est pas faite via SSL (*Secure Socket Layer*), le navigateur ne renvoie pas le cookie. Les connexions utilisant SSL sont celles qui se font vers des adresses commençant par `https://`. La valeur par défaut est `FALSE` (le cookie est envoyé quelle que soit la connexion).

**Remarque**

Tous les paramètres ci-dessus sont optionnels, le premier mis à part (le nom du cookie). Si vous souhaitez définir un paramètre mais pas les précédents, vous pouvez utiliser le chiffre zéro pour la date d'expiration et le dernier paramètre (transfert HTTPS uniquement) et une chaîne texte vide pour les autres (valeur, chemin d'accès, domaine de validité).

## Limitations et sécurité

Maintenant que vous avez toutes les données pour utiliser les cookies à leur plein potentiel, il est temps de faire quelques remarques concernant ce que vous ne pouvez ou ne devez pas faire avec les cookies.

### *Limitations dues aux navigateurs*

Les spécifications définissent une limite de 20 cookies par domaine et de 4 Ko par cookie, nom du cookie compris. Les navigateurs ont pour la plupart des limites moins restrictives que la norme, mais il est conseillé de ne pas dépasser ces valeurs. Il est donc hors de question d'utiliser les cookies pour stocker de longs textes ou des images. Le chapitre suivant, concernant les sessions, résoudra ce problème.

Les utilisateurs commencent à être sensibles aux intrusions dans la vie privée. En conséquence, les navigateurs intègrent maintenant des mécanismes pour filtrer les cookies. Si vos cookies ont des dates d'expiration trop importantes, sont trop nombreux ou trop volumineux, le navigateur peut très bien décider unilatéralement de ne pas les stocker, ou pas au-delà de la session courante. De plus, de nombreux utilisateurs reconfigurent leur navigateur pour refuser les cookies par défaut. Il ne faut donc pas, si on peut l'éviter, se reposer sur les cookies pour des informations critiques ou des fonctionnalités importantes.

### *Les cookies n'ont aucune sécurité*

Les cookies ont de nombreux avantages, mais il est important de noter que leur utilisation doit se faire de façon réfléchie. Pour reprendre l'analogie de l'introduction concernant la carte d'identité, il faut savoir que notre cookie, lui, ne contient aucune sécurité contre la modification : l'utilisateur peut créer, modifier et supprimer ses cookies très simplement. Cela correspond à une carte d'identité où les informations auraient été écrites au crayon de papier, donc non dignes de confiance.

Ainsi, pour la petite histoire, une régie publicitaire utilisait courant 2000 un système d'administration où elle stockait dans un cookie des valeurs telles que le prix du clic. Quand vous vous rendiez sur son espace d'administration et demandiez à remettre à jour vos paramètres personnels, elle plaçait un cookie chez vous avec toutes les informations qui devaient être contenues dans sa base de données vous concernant, dont le prix qu'elle vous versait par clic. L'utilisateur pouvait alors modifier ces informations en écrivant dans ses cookies. Les nouvelles valeurs étaient intégrées par l'outil de facturation.

Il faut donc être conscient du risque que vous encourez si vous utilisez les cookies pour des données confidentielles ou importantes. Bien sûr, en créant le système, vous vous direz que personne n'ira jusqu'à lire un cookie qui ne reste présent que très peu de temps, mais cela peut arriver, et dans ce genre de cas l'information circule très vite !

Concernant la sécurité, il est vivement déconseillé de mettre une quelconque information confidentielle dans un cookie : lire un cookie utilisé par votre site est simple pour l'utilisateur.



Il est donc par exemple exclu d'y stocker un mot de passe : il pourrait y être relu par n'importe qui ayant accès à la machine, un collègue par exemple.

De même, modifier, créer et supprimer un cookie à destination de votre site est accessible pour l'utilisateur. Donner foi à une quelconque information contenue dans un cookie est une erreur : si, par exemple, vous stockez un identifiant utilisateur dans un cookie, pensez qu'une personne malintentionnée pourra le modifier et se faire passer pour quelqu'un d'autre.

Les cookies sont à réserver pour des utilisations statistiques ou d'aide à l'utilisateur : le visiteur n'aura aucun intérêt à truquer son identifiant s'il ne sert que pour des statistiques (et s'il le fait, l'influence sera faible voire nulle). Un cookie qui ne fait qu'aider à pré-remplir un formulaire ne pose pas de problème non plus puisque cette information n'est à destination que de l'utilisateur (s'il la modifie, lui seul sera concerné).

## Cas d'application

### *Outil de personnalisation d'affichage*

#### Contexte

Pour fidéliser les visiteurs sur votre site, vous proposez régulièrement des brèves d'actualité sur une partie de la page d'accueil. Le rendu de votre page est satisfaisant, mais les nouvelles actualités ne sont pas mises en avant quand elles apparaissent déjà. Vous souhaiteriez donc que votre visiteur sache en un coup d'œil quelles sont les nouvelles actualités depuis sa dernière visite.

Le site actuel utilise une fonction `getActus()` qui retourne un tableau de message d'actualité. Chaque message d'actualité est lui-même un tableau de la forme suivante :

```
$message = array(
    'date' => date, sous le format utilisé par time() ,
    'titre' => titre de la brève ,
    'url' => adresse de la page web avec le contenu
) ;
```

#### Réalisation et solution retenue

Trois types de brèves vont être individualisés :

- les brèves qui sont affichées pour la première fois sur le poste du visiteur,
- les brèves dont les titres ont déjà été affichés, mais qui n'ont jamais été ouvertes par le visiteur,
- les brèves déjà lues.

Les nouvelles actualités auront un fond légèrement jaune de façon à mieux les faire ressortir, celles déjà lues auront un fond gris clair, de façon à les rendre moins attractives pour l'œil, les autres auront le fond blanc classique du reste de la page.

## Gestion des nouveaux messages

La gestion des nouveaux messages est la partie la plus simple. Il s'agit simplement de mettre un cookie à la date du jour pour chaque affichage. Il suffit par la suite de relire ce cookie. Si la date d'un message est plus récente que celle contenue dans le cookie, il s'agit d'un nouveau message. Sinon le message est un ancien message.

Voici une implémentation possible :

```
$messages = getActus() ;
foreach( $messages as &$actu ) {
    if ( $_COOKIE['derniere_visite'] > $actu['date'] ) {
        $actu['nouveau'] = FALSE ;
    } else {
        $actu['nouveau'] = TRUE ;
    }
}
$deux_mois = time() + 3600*24 *60 ;
setcookie('derniere_visite', time(), $deux_mois) ;
```

### Remarque

Nous utilisons dans cet exemple une itération dans un tableau avec `foreach`, comme nous l'avons souvent fait. La différence ici est que nous utilisons des références lors de la déclaration. Ainsi, quand nous modifions `$actu`, nous modifions en fait directement l'élément courant du tableau `$message`. Pour plus d'informations sur les références et la syntaxe de `foreach`, nous vous recommandons de vous reporter aux chapitres 3 et 4.

Avec ce type de schéma, un nouveau visiteur voit tous les messages comme nouveaux, c'est-à-dire en surbrillance. C'est certes un comportement logique, mais ce n'est pas du meilleur effet (imaginez cinquante images animées clignotantes). Une alternative intéressante consiste à considérer qu'à la première visite, tous les messages seront neutres par défaut (ni nouveaux ni lus) :

```
if ( isset($_COOKIE['derniere_visite']) ) {
    $date = $_COOKIE['derniere_visite'] ;
} else {
    $date = time() ;
}
$messages = getActus() ;
foreach( $messages as &$actu ) {
    $actu['nouveau'] = ( $date < $actu['date'] ) ;
}
$deux_mois = time() + 3600*24 *60 ;
setcookie('derniere_visite', time()) ;
```

## Gestion des messages lus

Pour retenir les messages lus ou non lus, nous avons choisi de stocker les différentes adresses des messages lus dans un deuxième cookie. Les adresses visitées sont stockées

dans un tableau, le tableau est sérialisé puis envoyé au navigateur. Voici le code qu'on pourrait mettre au début du script affichant le contenu des brèves :

```
if (isset($_COOKIE['lus'])) {
    // On récupère le tableau s'il existe
    $lus = $_COOKIE['lus'];
} else {
    // sinon on crée un tableau vide
    $lus = array();
}
//On ajoute au tableau la page courante
$lus[] = $_SERVER['REQUEST_URI'];
$cookie = serialize($lus);
//On définit une durée de vie de deux mois
$deux_mois = time() + 3600*24 *60;
setcookie('lus', $cookie, $deux_mois);
```

Il ne reste plus qu'à relire cette information dans la page d'accueil pour savoir quels messages ont été lus ou non :

```
if (isset($_COOKIE['lus'])) {
    $lus = $_COOKIE['lus'];
} else {
    $lus = array();
}
$messages = getActus();
foreach( $messages as &$actu) {
    // Pour chaque brève on regarde
    // si son url est stockée dans le cookie
    $actu['lu'] = in_array($actu['url'], $lus);
}
```

Notre architecture a tout de même un défaut : les adresses s'ajoutent jour après jour et finissent par représenter une taille non négligeable. Il est plus correct de retirer les adresses qui ne sont plus utilisées au fur et à mesure, donc de modifier notre code ainsi :

```
if (isset($_COOKIE['lus'])) {
    $lus = $_COOKIE['lus'];
} else {
    $lus = array();
}
$lus2 = array();
$messages = getActus();
foreach( $messages as &$actu ) {
    if ( in_array($actu['url'], $lus) ) {
        $actu['lu'] = TRUE;
        $lus2[] = $actu['url'];
    } else {
        $actu['lu'] = FALSE;
    }
}
```

```
$cookie = serialize($lus2) ;
$deux_mois = time() + 3600*24 *60 ;
setcookie('lus', $cookie, $deux_mois) ;
```

### Affichage des messages

Dans les deux premières parties, nous nous sommes contentés de mettre une clé `nouveau` et une clé `lu` à vrai ou faux. Il vous suffit maintenant de relire ces propriétés lors de l'affichage et de gérer les couleurs en conséquence.

### Code complet

Voici le code à ajouter en haut du script qui affiche le détail d'une brève :

```
if (isset($_COOKIE['lus'])) {
    $lus = $_COOKIE['lus'] ;
} else {
    $lus = array() ;
}
foreach( $messages as &$actu ) {
    $actu['lu'] = in_array($actu['url'], $lus) ;
}
```

Et voici le code à ajouter en haut du script de la page d'accueil :

```
if (isset($_COOKIE['derniere_visite'])) {
    $date = $_COOKIE['derniere_visite'] ;
} else {
    $date = time() ;
}
if (isset($_COOKIE['lus'])) {
    $lus = $_COOKIE['lus'] ;
} else {
    $lus = array() ;
}
$lus2 = array() ;
$messages = getActus() ;
foreach( $messages as &$actu ) {
    $actu['nouveau'] = ( $date < $actu['date'] ) ;
    if (in_array($actu['url'], $lus) ) {
        $actu['lu'] = TRUE ;
        $lus2[] = $actu['url'] ;
    } else {
        $actu['lu'] = FALSE ;
    }
}
$deux_mois = time() + 3600*24 *60 ;
setcookie('derniere_visite', time()) ;
$cookie = serialize($lus2) ;
setcookie('lus', $cookie, $deux_mois) ;
```



# 11

## Les sessions

---

Au chapitre précédent, nous avons vu comment stocker certaines informations sur le client grâce aux cookies. Ceux-ci ont toutefois deux défauts : leur taille est limitée et le visiteur peut les modifier à loisir. Ce n'est donc pas un bon emplacement pour des données sensibles comme des données d'authentification.

Les sessions sont adaptées à la sauvegarde de données confidentielles ou importantes. On peut citer quelques exemples courants de leur mise en application :

- authentifier un visiteur ;
- garder des informations sur un utilisateur tout au long de sa présence dans votre application ;
- gérer le panier d'achat d'un internaute sur votre site marchand ;
- mettre en place des formulaires en plusieurs parties et donc retenir les informations fournies dans les pages précédentes ;
- effectuer un cache par utilisateur de certaines actions coûteuses en ressources.

En fin de chapitre, retrouvez un cas d'application expliquant comment créer un système d'authentification sécurisé sur votre site.

### Qu'est-ce qu'une session ?

Pour répondre aux limitations des cookies, PHP met à disposition un concept plus évolué : les sessions. Au lieu de stocker vos informations chez le visiteur, vous les stockez sur le serveur. Techniquement, vous attribuez au visiteur un identifiant. À chaque fois

qu'il revient en annonçant cet identifiant, PHP récupérera toutes les informations relatives à ce visiteur qu'il avait sauvegardées.

Il est possible de reprendre une analogie similaire à celle faite lors de l'explication des cookies.

Lorsque l'administration vous écrit, elle inscrit une référence en haut des courriers et vous demande de la mentionner dans chacune de vos réponses. De son côté, elle crée une fiche et y récapitule les informations qu'elle a sur vous comme votre nom ou votre adresse, en plus de cette référence. Quand vous répondez, vous rappelez votre référence, votre contact cherche la fiche correspondante et peut voir toutes les informations écrites.

Dans notre cas, PHP envoie au navigateur un identifiant de session et stocke des données sur le client dans un fichier correspondant à l'identifiant. Quand le navigateur refait une requête sur une de nos pages, l'identifiant est automatiquement renvoyé. PHP ouvre alors le fichier correspondant pour récupérer tout ce qui y avait été sauvegardé.

Les données étant stockées sur le serveur web, vous pouvez y stocker des informations confidentielles sans crainte de modification par l'utilisateur. Contrairement aux cookies, la quantité d'information ne sera pas limitée. En revanche, à la différence des cookies, les sessions ne sont pas faites pour durer : Elles seront perdues après la visite de l'utilisateur. Dès la fermeture du navigateur, la session est perdue, comme pour un cookie sans date d'expiration (nous verrons plus loin pourquoi).

## Lecture et écriture

L'utilisation des sessions est très simple pour le programmeur : la manipulation est presque transparente et il suffit de lire ou d'écrire dans un tableau associatif classique une fois l'initialisation de la session faite.

La session s'initialise avec `session_start()`. PHP essaie alors de lire l'identifiant fourni par l'utilisateur, va chercher le fichier correspondant, et vous met à disposition les informations sauvegardées dans la superglobale `$_SESSION[]`. Si aucun identifiant de session n'est reçu, PHP en crée un unique aléatoirement, l'envoie au visiteur et crée le fichier de données correspondant.

Pour lire, modifier, supprimer ou créer des informations dans la session, il vous suffit de lire, modifier, supprimer ou créer des entrées dans le tableau `$_SESSION[]`.

```
<?php
// initialisation
session_start() ;
```

```
// Tester la présence de la variable 'langage' dans la session
if ( isset( $_SESSION['langage'] ) ) {
    echo 'langage existe dans la session et sa valeur est ' ;
    // Lecture de la variable de session 'langage'
    echo $_SESSION['langage'] ;
} else {
    echo 'langage n\'existe pas dans la session' ;
}
?>
```

Pour résumer, les informations de session se manient exactement comme des variables PHP. Vous pouvez ajouter, enlever ou modifier des éléments à la session simplement en modifiant le tableau de session. Contrairement aux cookies, vous n'avez aucune restriction sur le type de données stockées ou la quantité d'information : vous pouvez stocker indépendamment des tableaux, des nombres, du texte ou des objets sans passer par l'étape de sérialisation. Ce tableau `$_SESSION[]`, comme `$_COOKIE[]`, est une variable superglobale et peut être lu n'importe où dans vos scripts.

La session commence avec un appel à `session_start()` ; son rôle est d'initialiser la gestion. Nous verrons dans la partie suivante que les sessions utilisent parfois les cookies en interne. La conséquence est que l'appel à `session_start()` doit respecter les mêmes règles que la fonction `setcookie()` : être placé en haut du script, avant toute sortie vers la page web. L'initialisation doit se faire dans tous les scripts utilisant les sessions, pas uniquement le premier.

```
<?php
// Initialisation de la session
session_start() ;

// Écrire 'PHP' dans la variable de session 'langage'
$_SESSION['langage'] = 'PHP' ;

$tableau = array('un', 'deux', 'trois', 'quatre');
$_SESSION['tab'] = $tableau;

?>
```

## Utilisation avancée

Vous savez maintenant utiliser une variable de session ; il n'est pas indispensable pour vous d'aller plus loin si votre utilisation est basique. Nous allons voir par la suite comment fonctionnent les sessions, ce qu'on peut modifier dans la configuration, comment créer son propre gestionnaire de sessions et surtout quelles sont les précautions à prendre quand on les utilise.



## Fonctionnement interne des sessions

Pour passer à la suite, il est nécessaire de comprendre le fonctionnement interne des sessions. La figure 11-1 montre le comportement par défaut, un peu simplifié.

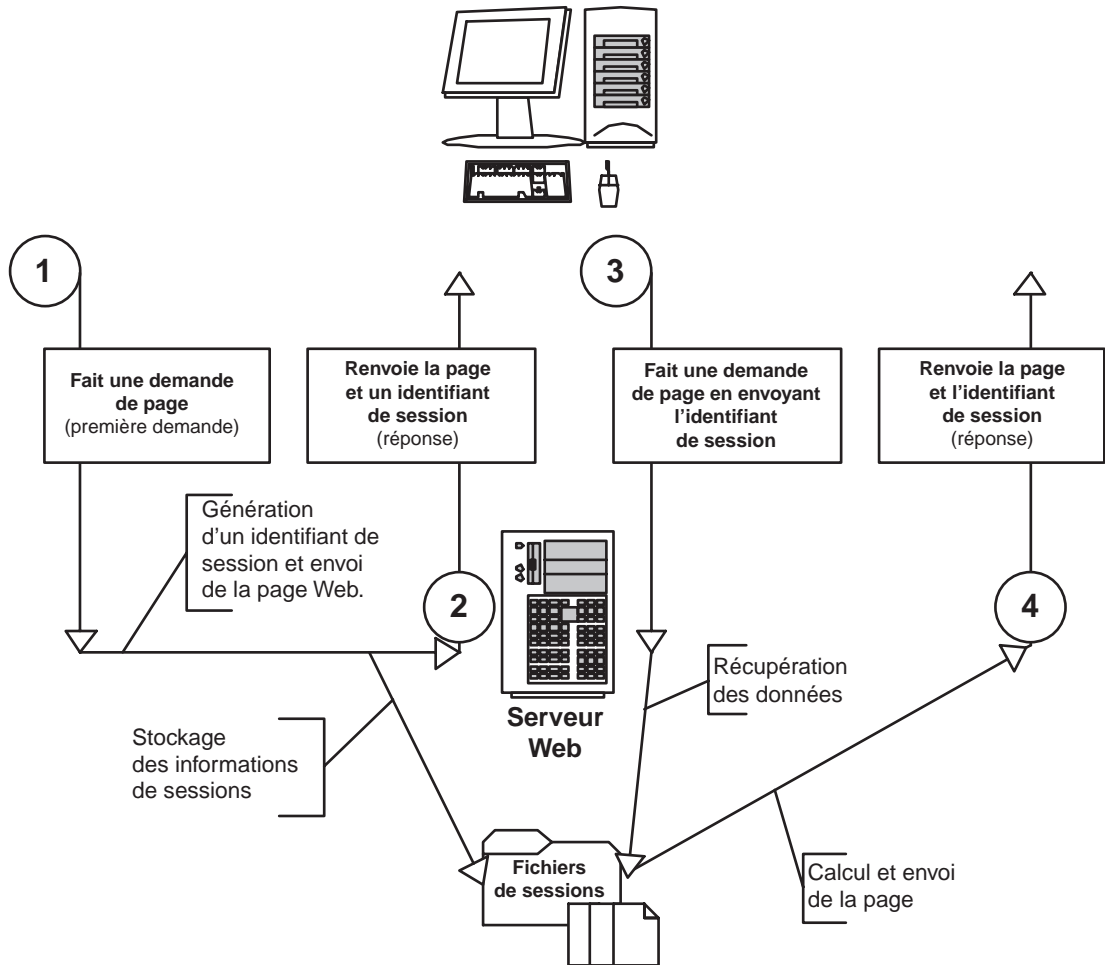


Figure 11-1

Fonctionnement des sessions

Lorsque vous faites appel à `session_start()`, PHP essaie de lire le contenu d'un cookie nommé par défaut `PHPSESSID`. Ce cookie contient l'identifiant assigné à l'utilisateur. Si ce cookie n'existe pas, un nouvel identifiant est créé de manière aléatoire puis est envoyé par cookie au navigateur (sans date d'expiration). C'est à cause de cette partie qu'il vous a été indiqué précédemment de traiter `session_start()` comme la fonction `setcookie()`.

PHP ouvre alors, s'il existe, un fichier sur le serveur, qui a l'identifiant utilisateur comme nom. Le contenu est interprété pour recréer les variables déjà stockées dans la session. Toutes les variables sont mises dans le tableau `$_SESSION[]`.

À la fin du script, PHP relit le tableau `$_SESSION[]` et enregistre le contenu dans le fichier de session sur le serveur. À l'intervalle d'une requête sur cent, en moyenne, PHP efface les fichiers auxquels personne n'a accédé depuis une heure et demie.

## Suppression d'une session

Habituellement, il n'est pas nécessaire de supprimer une session puisque PHP l'efface de lui-même au bout d'un certain temps. Si toutefois vous voulez détruire explicitement le fichier de données, vous pouvez utiliser la fonction `session_destroy()`. Les paramètres de la session devant être initialisés auparavant, il vous faudra tout de même faire un appel à `session_start()` quelques lignes plus haut dans le même script.

### Remarque

Cette fonction ne fait qu'effacer le fichier de données, elle n'efface pas les variables présentes dans `$_SESSION[]` ni ne supprime le cookie. Si vous voulez éviter tout risque de confusion, effacez `$_SESSION[]` aussi.

```
<?php
// On initialise et utilise la session
session_start();
$_SESSION['nom'] = 'Pierre';
echo $_SESSION['nom']; // affiche Pierre

// Divers traitements

// On détruit la session
session_destroy();
unset($_SESSION);
echo $_SESSION['nom'] ; // N'affiche rien
?>
```

## Définition manuelle de l'initialisation

Dans le fonctionnement par défaut, PHP lit un cookie du nom de `PHPSESSID` pour trouver l'identifiant. Vous avez cependant la possibilité de définir vous-même la façon de créer et récupérer les identifiants selon une méthode de votre cru. La fonction `session_id()` vous retourne l'identifiant de session actuel. Quand on lui fournit une chaîne de caractères en argument, elle change l'identifiant pour ce qu'elle a reçu en paramètre. Fournir un identifiant inexistant permet de créer une nouvelle session avec cet identifiant.

De même, `session_name()` vous permet de récupérer le nom utilisé pour la session en cours (`PHPSESSID` par défaut). Si vous lui fournissez un nouveau nom en paramètre, il utilisera ce

nouveau nom pour le script en cours. Il faut garder à l'esprit que cette modification n'est valable que pour le script en cours. Les autres scripts garderont l'ancien nom et si vous n'y faites pas le même changement, il en résultera l'utilisation de deux sessions différentes.

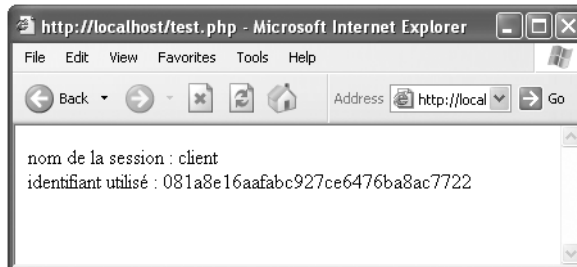
### Important

Si vous décidez de créer vous-même vos identifiants de session, n'oubliez pas de lire la partie sur la sécurité des sessions : fournir des identifiants sans précaution peut permettre à quelqu'un d'usurper la session d'un autre.

Le script suivant permet d'afficher les informations de la session (nom et identifiant) ; son résultat est illustré dans la figure 11-2.

```
<?php
// Définition de 'client' comme nom de session
session_name('client') ;
// et de '/tmp' comme répertoire de stockage des sessions
session_save_path('/tmp');
// Initialisation de la session
session_start() ;
// On récupère les informations de la session :
echo 'nom de la session : ', session_name() , '<br>' ;
echo 'identifiant utilisé : ', session_id() , '<br>' ;
?>
```

**Figure 11-2**  
*Informations  
de session*



Le nom et l'identifiant sont utilisés lors de l'initialisation de la session pour récupérer les données déjà sauvegardées ou envoyer le cookie. Si vous changez ces paramètres, il vous faut le faire avant l'appel à `session_start()`, sinon ils ne pourront pas être pris en compte.

## Stockage des données de session

Par défaut, PHP stocke les données de session dans un fichier du répertoire temporaire de votre système. Vous pouvez récupérer l'adresse du répertoire de stockage grâce à la fonction `session_save_path()`.

```
<?php
```

```
echo session_save_path();  
?>
```

Avant de stocker les données dans un fichier, PHP doit les transformer en une chaîne de caractères, les sérialiser. La façon de gérer cette transformation peut être modifiée. Par défaut, PHP gère ces transformations avec un module nommé `php`, qui utilise les fonctions `session_encode()` et `session_decode()`. Le procédé de ce module est similaire à celui des fonctions `serialize()` et `unserialize()` que nous avons utilisées pour les cookies.

Un autre module, nommé `wddx`, permet de stocker les données sous forme XML standardisée. Il nécessite cependant un PHP compilé avec l'option `--enable-wddx`. La fonction permettant de connaître ou de changer le module utilisé est `session_module_name()`. Le paramètre optionnel est le nom du nouveau module à utiliser.

## Paramètres du cookie

Par défaut, PHP utilise un cookie sans date d'expiration ni restrictions pour envoyer l'identifiant. Il a tout de même été prévu une fonction pour modifier tous les paramètres : son nom est `session_set_cookie_params()`. L'ordre et la signification des paramètres sont les mêmes que pour la fonction `setcookie()`, en considérant que vous n'avez pas à remplir le nom et la valeur du cookie (PHP le fait tout seul selon le nom et l'identifiant de la session).

### Remarque

La date d'expiration du cookie et celle du fichier de données sont indépendantes : définir un cookie permanent n'empêchera pas les données de la session d'être effacées au terme de leur période de validité.

## Accès concurrents aux sessions

Pour éviter que deux scripts ne modifient le fichier de données en même temps (ce qui ferait perdre les modifications d'un des deux scripts), PHP effectue un verrouillage du fichier : seul un script peut y accéder à un instant donné. Dans les cas classiques, le principe pose peu de problèmes puisque le visiteur charge les pages une à une. Il peut pourtant arriver que plusieurs pages soient appelées à la fois, dans le cas d'utilisation de *frames* par exemple. Ce mécanisme peut entraîner une attente indésirable pour l'utilisateur.

Pour résoudre ce problème, deux fonctions sont disponibles :

- La première, `session_readonly()`, ouvre la session en lecture seule. Si vous remplacez `session_start()` par cette fonction, aucune de vos modifications ne sera sauvegardée. En échange, PHP n'aura pas besoin de verrouiller le fichier de données pour cette page. Dans le cas de *frames*, il est fréquent que seule la page centrale ait besoin d'écrire dans la session, les autres parties pouvant se contenter de la lecture des données.

- La deuxième, `session_write_close()`, permet d'enregistrer le contenu de la session avant la fin du script. Les modifications faites après l'appel à cette fonction seront perdues mais cela permet à une autre page d'utiliser le fichier de données sans attendre la fin de l'exécution. Si vous avez une page très longue à exécuter, comme un calcul d'image, il peut être bénéfique de rendre la main plus vite en fermant la session avant la fin.

## Configuration de PHP

Les directives de configuration suivantes se trouvent dans le fichier de configuration `php.ini` au-dessous de l'entrée `[session]`. Presque toutes sont redéfinissables pendant l'exécution du script via la fonction `ini_set()`. Attention cependant à redéfinir ces directives de configuration avant d'initialiser la session via `session_start()`.

### Initialisation des sessions

Le nom de la session par défaut (et donc du cookie envoyé pour la session) peut être changé au niveau de la configuration pour ne pas avoir à faire un appel à `session_name()` en haut de chaque script. La directive est `session.name`. Elle prend en paramètre le nom de la session.

```
session.name = "PHPSESSID"
```

Vous pouvez aussi demander à PHP de démarrer automatiquement la gestion des sessions sur chacune de vos pages, vous évitant de faire un appel à `session_start()`. Pour obtenir ce comportement, il suffit de mettre la directive `session.auto_start` à 1 (la valeur par défaut est 0). Cela peut toutefois entraîner une charge supplémentaire non négligeable si vous n'utilisez pas les sessions dans tous vos scripts.

```
session.auto_start = 0
```

#### Remarque

Si vous utilisez cette fonctionnalité vous ne pourrez plus changer les paramètres de configuration des sessions pendant l'exécution : la session sera déjà initialisée lors de la première ligne de votre script.

### Stockage des données de session

Le répertoire de sauvegarde par défaut et le module utilisé pour la sérialisation sont aussi modifiables via le fichier de configuration. Les directives utilisées sont `session.serialize_handler` et `session.save_path`. Les paramètres sont les mêmes que pour les fonctions `session_module_name()` et `session_save_path()`.

```
session.serialize_handler = "php"  
session.save_path = "/tmp"
```

## Paramètres du cookie

Vous pouvez modifier les paramètres du cookie de session, soit pendant l'exécution de vos scripts, soit une fois pour toutes dans la configuration de PHP. Les paramètres modifiables sont ceux de la fonction `session_set_cookie_params()` :

- la date d'expiration est gérée par `session.cookie_lifetime` ;
- le répertoire de validité par `session.cookie_path` ;
- le domaine de validité par `session.cookie_domain` ;
- la gestion des connexions sécurisées est gérée par le paramètre `session.cookie_secure`.

```
session.cookie_lifetime = "0"  
session.cookie_path = "/"  
session.cookie_domain = ""  
session.cookie_secure = ""
```

## Expiration des sessions

Les fichiers de session stockés sur le serveur n'ont pas une durée de vie illimitée. Pour effacer les fichiers correspondants aux sessions expirées, PHP utilise un ramasse-miettes (*garbage collector* en anglais). Celui-ci fonctionne de la façon suivante.

À chaque lancement de session, un nombre est tiré au hasard entre 0 et 99. Si ce numéro est inférieur à la valeur de la directive de configuration `session.gc_probability`, alors PHP lit le répertoire de stockage et efface tous les fichiers de session qui sont expirés. Un fichier est considéré comme ayant expiré si son âge en secondes est plus important que la valeur de `session.gc_maxlifetime`. Définir des valeurs trop importantes pour ces deux directives aura un impact négatif sur les performances du système, le ramasse-miettes ayant à analyser trop de fichiers ou trop souvent.

```
session.gc_probability = "1"  
session.gc_maxlifetime = "1440"
```

### Remarque

S'il y a peu de visiteurs sur une page, il se peut très bien que le ramasse-miettes ne tire pas rapidement un nombre inférieur à `session.gc_probability` et qu'une session reste active beaucoup plus longtemps que spécifié par `session.gc_maxlifetime`.

## Gestion du cache

Le protocole HTTP utilisé pour les pages web permet de spécifier divers niveaux de cache. La valeur de la directive peut être :

- `public` - Les proxies et navigateurs pourront sauvegarder la page et la resservir à tous les utilisateurs. C'est généralement un comportement peu souhaitable dans notre cas, car au final tous les utilisateurs risqueront de mélanger leurs sessions.

- `private_no_expire` - La page peut être sauvegardée et resservir plus tard, mais pour cet utilisateur seulement. Ceci implique que, quand l'utilisateur va revenir sur la même page, il ne fera pas de requête à votre serveur, vous ne pourrez donc pas faire les traitements que vous voudrez. Il est rare que ce cas soit souhaitable si vos pages sont dynamiques.
- `private` - Cette valeur a une signification similaire à la précédente mais le cache perd sa validité au bout d'un moment. Pendant la période de validité, si l'utilisateur revient sur la même page, il ne déclenche aucune exécution sur le serveur.
- `nocache` - C'est la valeur par défaut pour les sessions. Les proxies n'enregistreront pas la page et les navigateurs ne s'en resserviront que dans de très rares cas (bouton précédent du navigateur par exemple).
- `none` - Il est explicitement demandé de ne pas sauvegarder la page. Le navigateur fera une nouvelle requête pour tout passage sur la page (même quand l'utilisateur revient sur la page précédente).

La valeur envoyée par PHP lors d'une session est spécifiée par la directive `session.cache_limiter`. Vous ne devriez modifier cette valeur que si vous comprenez parfaitement les implications qu'elle aura.

```
session.cache_limiter = "nocache"
```

Si vous avez choisi un cache avec les valeurs `private` ou `public`, vous pouvez définir la période de validité de la page en spécifiant un âge maximal en secondes dans la directive `session.cache_expire`.

```
session.cache_expire = 180
```

## Transmission de l'identifiant

Le problème des cookies est qu'ils peuvent être filtrés par l'utilisateur. Pour les néophytes n'ayant pas configuré eux-mêmes cette caractéristique sur leur navigateur, on peut envisager de mettre un message d'explication contenant la procédure pour réactiver la prise en charge des cookies. Cependant, cette méthode découragera probablement certains visiteurs.

Pour résoudre ce problème, PHP a implémenté une méthode de rechange pour la transmission de l'identifiant de session : il s'agit de modifier les adresses des liens de votre page pour y accoler automatiquement l'identifiant. Quand le visiteur sans cookie clique sur un lien, il voit l'adresse de la page se terminer par `?PHPSESSID=xxxxxxxxxx`. PHP reçoit cette information lors de la requête et sait relire l'identifiant (marqué par des x dans l'exemple).

PHP essaie de détecter seul si l'utilisation du cookie fonctionne ou non : il modifie les liens de votre page pour y inclure l'identifiant de session uniquement si l'identifiant n'a pas été reçu via un cookie.

**Remarque**

Dans la première page créant la session, PHP n'aura reçu aucun identifiant via un cookie. Les liens seront donc modifiés, que les cookies marchent avec ce visiteur ou pas.

La liste des éléments qui sont modifiés pour faire apparaître l'identifiant dans les liens est disponible dans la directive `url_rewriter.tags` du fichier `php.ini`. Pour chaque élément, on spécifie le nom de la balise HTML et le nom de l'attribut contenant l'URL ; les deux sont séparés par le symbole `=`. La balise `<form>` a une mise en forme spécifique (`form=fakeentry`) pour demander à PHP d'insérer une entrée cachée dans le formulaire.

```
url_rewriter.tags =  
    "a:href,area:href,frame:src,input:src,form=fakeentry"
```

Si vous comptez réécrire vous-même certains liens, PHP met à disposition la constante `SID` (pour *session id*), qui contient la chaîne à ajouter après le point d'interrogation, ceci vous évitant d'avoir à vérifier vous-même les noms et identifiants de la session.

La fonctionnalité de réécriture des liens est désactivée par défaut. Elle est activable en mettant à 1 la directive `session.use_trans_sid`. Bien entendu, si elle est activée, l'analyse de la page et la modification des liens auront un impact sur les performances. Cette directive est la seule des sessions à ne pas pouvoir être modifiée dans un script et doit être définie dans le fichier de configuration PHP.

**Important**

L'activation de cette fonctionnalité diminue la sécurité de vos sessions. Voir à ce sujet la section « Sécurité » de ce chapitre.

```
session.use_trans_sid = 0
```

Si vous choisissez d'utiliser la réécriture des liens, vous pouvez si vous le souhaitez désactiver complètement l'envoi des cookies pour la gestion des sessions. La directive `session.use_cookies` définit ce comportement : à 1, les sessions essaieront d'utiliser les cookies, à 0, elles ne le feront jamais.

```
session.use_cookies = 1
```

## Gestionnaires de sessions

PHP utilise le système de fichiers pour stocker les sessions et fait tout automatiquement. Pourtant, il peut être utile dans certains cas d'utiliser une solution tierce pour le stockage des sessions, par exemple se servir d'une base de données centralisée afin d'avoir accès aux mêmes sessions sur plusieurs serveurs. PHP utilise la directive `session.save_handler` pour connaître le comportement à adopter.



Par défaut, la valeur `files` est utilisée : une fois sérialisées, les données sont enregistrées dans des fichiers sur le serveur. C'est le comportement qui a été décrit jusqu'à présent.

Une deuxième possibilité est le module `mm`. Si votre PHP est compilé avec l'option `—with-mm`, il permet de sauvegarder les sessions en mémoire plutôt que sur le système de fichier. Ce module permet en général un gain réel côté performance mais a un côté négatif : dans ce cas, PHP ne gère pas le verrouillage lors de l'accès aux données. Il peut donc en résulter des pertes d'informations si plusieurs scripts accèdent à la session en même temps. Voyez à ce sujet le paragraphe sur les accès concurrents aux fichiers de session légèrement plus haut dans ce chapitre.

Une autre variante est le module `msession`. Il nécessite l'installation d'un serveur de session dédié et une compilation avec `—with-msession`. Les sessions ne seront plus stockées sur le système de fichiers mais seront gérées par un serveur distant. Cela peut être très utile dans une architecture avec plusieurs serveurs traitant les requêtes : les sessions peuvent être centralisées sur un serveur indépendant, toutes les machines utilisent alors les mêmes données de session.

Si les modules de gestion de sessions fournis avec PHP ne vous suffisent pas, vous pouvez vous procurer des scripts ou extensions qui implémentent des gestionnaires supplémentaires. Parmi eux se trouve `session_pgsql`, qui permet l'utilisation des sessions à travers une base de données PostgreSQL. Vous pourrez le trouver par l'adresse <http://pear.php.net/manual/en/pecl.session-pgsql.php>, sur le site de Pear.

## Définir un gestionnaire personnalisé

Si vous voulez gérer vous-même le stockage des données, vous pouvez définir la valeur `user`. Dans ce cas, PHP attendra que vous utilisiez dans chaque script la fonction `session_set_save_handler()` pour lui indiquer comment traiter les informations. Cette fonction prend en paramètres les noms de six fonctions, qui vont respectivement initialiser le gestionnaire, clôturer la gestion, lire les données correspondant à un identifiant, écrire les données correspondant à un identifiant, détruire les données correspondant à un identifiant et effacer les sessions expirées. Sauf exception, ces fonctions doivent être définies et retourner la valeur `TRUE` quand il n'y a pas eu d'erreur.

```
session_set_save_handler(  
    'init','ferme','lit','ecrit','efface','nettoie'  
);
```

La première fonction initialise le gestionnaire. C'est par exemple là que vous vous connecteriez au serveur lors d'une gestion par base de données. Ce n'est donc pas à vous d'aller chercher le répertoire de sauvegarde ou le nom de la session via les fonctions `session_save_path()` et `session_name()` : PHP vous fournit tous les arguments utiles en paramètres. Si pour vous le répertoire de stockage n'a aucune signification, vous pouvez vous servir de cette valeur comme nom de table pour une base de données, ou tout simplement l'ignorer.

```
// Premier paramètre : répertoire de stockage
// Deuxième paramètre : nom de la session
function init( $session_save_path, $session_name ) {

    // Nous réutiliserons ces informations par la suite
    // donc nous voulons utiliser des variables globales
    global $repertoire_stockage , $nom_de_session ;
    $repertoire_stockage = $session_save_path ;
    $nom_de_session = $session_name ;
    return TRUE ;
}
```

La deuxième fonction est la complémentaire de la précédente : elle ferme la session en cours. C'est là que vous fermez l'accès à la base de données par exemple.

```
function ferme() {
    return TRUE ;
}
```

Vient ensuite la fonction de lecture. Elle prend l'identifiant en paramètre et doit renvoyer le contenu de la session. Il ne vous est pas demandé d'interpréter le contenu pour créer les variables. C'est fait de manière transparente par PHP en fonction de sa configuration. Vous devez traiter le contenu comme une chaîne de caractères quelconque sans vous préoccuper de son sens.

```
function lit( $identifiant ) {

    // On réutilise les variables initialisées
    global $repertoire_stockage , $nom_de_session ;
    // On construit l'adresse du fichier qui contient les données
    $fichier = $repertoire_stockage.'/'.$nom_de_session.$identifiant;
    // On lit les données
    if ( $fp = @fopen($fichier, 'r') ) {
        $donnees = fread($fp, filesize($fichier) ) ;
        fclose($fp) ;
        return $donnees ;
    } else {
        return '' ;
    }
}
```

La quatrième fonction permet d'écrire les données. Elle prend aussi l'identifiant de session en premier paramètre, le deuxième étant la chaîne de caractères à enregistrer dans la session. Vous n'avez pas besoin de sérialiser les données ou d'essayer de comprendre la donnée à stocker : PHP s'occupe de tout.

```
function ecrit( $identifiant, $donnees ) {

    // On réutilise les variables initialisées
    global $repertoire_stockage , $nom_de_session ;
```

```
// On construit l'adresse du fichier qui contient les données
$fichier = $repertoire_stockage.'/'.$nom_de_session.$identifiant;

$fp = fopen($fichier, 'w') ;
fwrite( $fp, $donnees, strlen($donnees) ) ;
fclose($fp) ;
return TRUE ;
}
```

La fonction suivante est appelée pour effacer les données d'une session existante. Encore une fois, l'identifiant de session est passé en paramètre :

```
function efface($identifiant) {

    // On réutilise les variables initialisées
    global $repertoire_stockage , $nom_de_session ;

    // On construit l'adresse du fichier qui contient les données
    $fichier = $repertoire_stockage.'/'.$nom_de_session.$identifiant;

    return @unlink($fichier) ;
}
```

La dernière fonction de la série permet de gérer le ramasse-miettes. PHP vous fournit en paramètre l'âge maximal des sessions en secondes. Toutes les sessions plus anciennes doivent être effacées.

```
function nettoie($age) {
    // on effacera ici les fichiers de plus de $age secondes
    // voir à ce sujet le chapitre sur la gestion des fichiers
}
```

#### Remarque

Pour la clarté de l'exemple, nous n'avons pas fait de gestion d'erreur, mais il faudra bien sûr l'implémenter dans votre gestionnaire, par exemple afin de traiter le cas de fichiers inexistantes. De même, aucun verrouillage des données n'a été fait et il est de votre responsabilité d'installer ou non un tel système.

Mises bout à bout, ces fonctions définissent un gestionnaire similaire à celui utilisé par défaut par PHP. Comme vous le voyez, vous n'avez pas à vous préoccuper de la gestion des identifiants, de la sérialisation ou des autres détails : PHP va lui-même remplir ces tâches. Une fois défini un gestionnaire personnalisé, l'utilisation des fonctions se fait exactement comme auparavant.

Pour ceux qui préfèrent la programmation orientée objet, il est possible de spécifier une méthode d'un objet plutôt qu'une fonction. Au lieu de fournir une chaîne de caractères, il faut alors lui fournir un tableau contenant l'objet utilisé en premier élément et le nom de la méthode à appeler en second.

## Limitations et sécurité

Les sessions sont un sujet critique pour la sécurité. Si quelqu'un a accès à vos sessions, il peut potentiellement usurper les droits d'un de vos utilisateurs et, pourquoi pas, de l'administrateur. Plusieurs choses sont à savoir afin de limiter les problèmes.

### *Cachez les sessions*

Si vous partagez un serveur avec d'autres utilisateurs, définissez un répertoire de stockage pour les sessions auquel vous seul avez accès. Dans le cas contraire, ils pourront lire les identifiants de sessions et les fournir à vos pages pour « voler » une session. De plus, le contenu des fichiers n'est pas crypté ; ils pourront donc lire les données confidentielles qui y sont contenues.

```
session.save_path = "/home/utilisateur/repertoire/personnel"
```

### *N'utilisez pas la réécriture des liens*

Si vous le pouvez, n'utilisez pas la fonction de réécriture des liens. Quelqu'un pourrait avoir accès à la session d'un autre. Un cas classique est l'échange de liens : l'utilisateur copie ce qu'il a dans sa barre d'adresse et l'envoie à un deuxième utilisateur. Si ce lien contient l'identifiant de session, le deuxième utilisateur profitera de la session du premier et de ses droits d'accès. Un deuxième cas habituel est l'exploitation du fait que, quand votre navigateur va sur une page, il envoie au serveur l'adresse de la page précédente. Un serveur malveillant pourra voir votre identifiant de session et le réutiliser pour avoir accès à votre session.

Si vous n'avez pas activé la réécriture des liens, vous pouvez aussi refuser tout identifiant de session qui n'aurait pas été envoyé via un cookie. Si chacun peut très bien créer un cookie chez lui et le falsifier, certaines méthodes de vol de session nécessitent l'utilisation d'identifiants passés dans l'adresse de la page. Vous évitez de fait toute cette catégorie d'attaques. PHP gère cette possibilité de restriction via la directive de configuration `session.use_only_cookies`, qu'il faut mettre à 1 pour l'activer.

```
session.use_trans_sid = 0
session.use_cookie = 1
session.use_only_cookies = 1
```

### *Les identifiants par défaut suffisent*

Évitez de définir vous-même les identifiants de session. Si vous utilisez une méthode totalement aléatoire, vous risquez de retomber deux fois sur la même session. Le cas est peu probable, mais possible. Si vous utilisez une méthode non aléatoire, vous risquez de rendre prédictible l'allure des identifiants.

PHP utilise une méthode intermédiaire qui est un bon compromis. Les développeurs de PHP savent ce qu'ils font ; si vous n'êtes pas sûr de vous, faites-leur confiance et laissez-les choisir.

### ***Attaque par fixation de session***

Quand vous utilisez des authentifications ou des procédures similaires, il vous faut faire attention à un type d'attaque qui s'appelle *fixation de session*. Pour résumer brièvement, une telle attaque repose sur la possibilité pour l'attaquant de fournir un numéro de session connu à la victime avant qu'elle s'authentifie, et par la suite de bénéficier de son authentification.

Vous trouverez plus de renseignements dans le chapitre sur la sécurité. Si les détails de fonctionnement ne vous intéressent pas, assurez-vous juste de faire appel à `session_regenerate_id()` avant d'authentifier la personne ou de stocker des informations trop confidentielles pour la première fois de la session. Cette fonction va changer l'identifiant de session tout en gardant les informations contenues. Un exemple d'implémentation est donné dans le cas d'application à la fin de ce chapitre. Vous pouvez aussi limiter les possibilités d'attaque par fixation en activant la directive `session.use_only_cookie` dans le fichier de configuration de PHP.

### ***Vérifiez l'identité de l'utilisateur***

Il existe plusieurs manières d'ajouter un supplément de sécurité aux sessions. L'une d'elle est de stocker dans la session le plus d'informations possible sur l'utilisateur : son adresse IP et la version de son navigateur par exemple. À chaque page, vérifiez si les informations sont bien les mêmes qu'au départ ; dans le cas contraire, refusez l'accès et détruisez la session. Ces informations sont falsifiables, il ne s'agit donc pas d'une sécurité absolue, mais quelqu'un qui a réussi à voler un identifiant de session n'aura pas forcément réussi à voler ces informations : vous évitez tout de même toute une catégorie d'attaques.

### ***N'ayez pas confiance***

L'utilisation des sessions garantit normalement que vous seul ayez accès aux données que vous y stockez. Cela dit, on n'est jamais trop prudent : si vous pouvez l'éviter, ne stockez pas d'informations trop confidentielles dans la session. Par exemple, au lieu de stocker le nom et le mot de passe de l'utilisateur pour vérifier qu'il est authentifié, stockez simplement son nom et le fait que le mot de passe était valide. Ainsi, si jamais quelqu'un arrive à lire la session, il n'y aura pas divulgation du mot de passe.

## Cas d'application

### Authentification par formulaire

#### Contexte

Un client vous a demandé de faire passer un de ses sites Internet à la vitesse supérieure en mettant en place un extranet pour ses clients. Cette partie implique une forte sécurité, car il souhaite mettre à disposition des informations confidentielles. Vous souhaitez donc pouvoir en restreindre l'accès aux utilisateurs identifiés et appartenant à une liste établie. Le site de votre client est hébergé chez un spécialiste sous la forme d'un hébergement mutualisé.

#### Réalisation

Pour créer un espace sécurisé, on fait généralement appel à une des deux méthodes suivantes :

- utilisation des contrôles d'accès intégrés au serveur web,
- utilisation d'une authentification gérée par l'application PHP via des sessions.

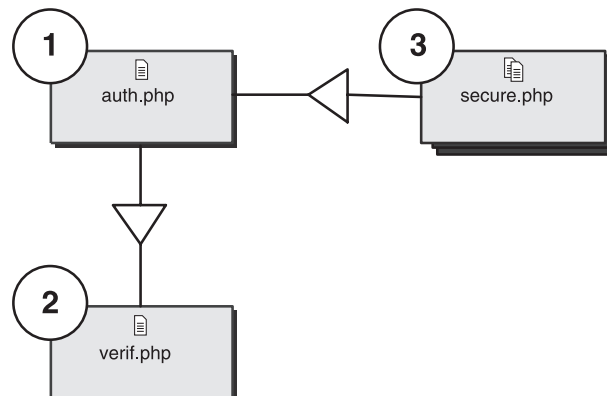
Dans notre cas, nous sommes dans le cadre d'un hébergement mutualisé et nous n'avons aucun moyen de modifier la configuration Apache, que ce soit de manière globale ou de manière locale (via des fichiers `.htaccess`). La seule solution est donc l'utilisation des sessions PHP. Ces sessions vous permettent de garder trace de l'identification donnée par les gens et validée par vos soins. Nous allons donc créer un fichier permettant de s'identifier, un fichier permettant de valider les données soumises et un script de vérification de session. Ce dernier script sera inclus en haut de chaque page qui aura besoin d'être sécurisée.

#### Architecture du système

Nous allons individualiser les trois scripts suivants et les utiliser comme le montre la figure 11-3 :

- `auth.php` : le formulaire pour s'identifier ;
- `verif.php` : le script vérifiant le couple pseudo/mot de passe et identifiant l'utilisateur ;
- `secure.php` : le script qui vérifie si l'utilisateur a accès à la ressource demandée.

**Figure 11-3**  
*Procédure  
d'identification*



### Le formulaire d'authentification : auth.php

Le formulaire d'authentification n'est a priori pas compliqué puisque, dans sa version la plus simple, deux champs suffisent : pseudo et mot de passe.

```
<html>
<head><title>Identification</title></head>
<body>
  <form method="post" action="verif.php">
    <p>
      <label for="nom">Pseudo : </label>
      <input type="text" name="pseudo">
    </p>
    <p>
      <label for="motdepasse">Mot de passe : </label>
      <input type="password" name="motdepasse">
    </p>
    <p>
      <input type="submit" value="s'identifier">
    </p>
  </form>
</body>
</html>
```

### Le formulaire de validation : verif.php

L'utilisateur qui cherche à s'identifier nous fournit via la page précédente (auth.php) un pseudo et un mot de passe. Ces informations sont envoyées via la méthode POST et sont donc accessibles dans la superglobale \$\_POST[[]].

Nous allons donc vérifier que le couple pseudo/mot de passe proposé correspond bien à un couple existant. Si c'est le cas, une variable de session contenant le pseudo de l'utilisateur lui sera assignée. Celle-ci permettra de l'identifier.

Dans l'exemple, la fonction verification() est une fonction imaginaire qui renvoie TRUE quand le mot de passe est le bon et FALSE dans le cas contraire ; à vous de la définir selon vos besoins.

```
<?php
// Initialisation de la session
session_start() ;

// Si on a reçu les données d'un formulaire :
if ( isset( $_POST['pseudo'] ) && isset ( $_POST['motdepasse'] ) ) {

  // On les récupère
  $nom = $_POST['pseudo'] ;
  $motdepasse = $_POST['motdepasse'] ;
```

```
// On teste si le mot de passe est valide :
if ( verification( $nom, $motdepasse ) ) {
    // Le mot de passe est valide, l'utilisateur est identifié
    // On change d'identifiant de session
    session_regenerate_id() ;

    // On sauvegarde donc son nom dans la session
    $_SESSION['nom'] = $nom ;
    $message = 'vous êtes correctement identifié' ;
} else {
    // sinon on avertit l'utilisateur :
    $message = 'Mauvais mot de passe' ;
    $message .= '<a href="auth.php">retour</a>' ;
}
} else {
    // Un des champs n'est pas rempli
    $message = 'le login ou le mot de passe est vide' ;
    $message .= '<a href="auth.php">retour</a>' ;
}
?>
<html>
<head><title>Identification</title></head>
<body><p>
<?php echo $message ?>
</p></body>
</html>
```

### La fonction verification()

Nous avons utilisé dans notre exemple précédent la fonction `verification()` pour valider que le login est le mot de passe correspondent bien dans la base de données. Techniquement, c'est à vous d'implémenter cette fonction selon le schéma de base de données que vous utilisez, mais nous allons vous présenter ci-dessous un exemple minimal pour vous permettre une mise en place rapide.

La fonction `verification()` prend en argument un nom d'utilisateur et un mot de passe. La fonction renvoie `TRUE` si le nom d'utilisateur et son mot de passe correspondent à une entrée dans la base.

Nous allons considérer que nous avons une base de données nommée « application » et une table nommée « user » contenant les champs permettant l'authentification :

- identifiant : `id_user`
- nom d'utilisateur : `login`
- mot de passe non crypté (dans un cas réel utilisez toujours un mot de passe crypté, à l'aide de la fonction `crypt()` par exemple) : `pass`

```
<?php
function verification($nom,$pass){
```



```
// Connexion SQL
$dbhote = 'localhost';
$dbuser = 'root';
$dbpass = 'monpass';
$dbbase = 'application';

$dsn = "mysql:dbname=$dbbase;host=$dbhote";
$dbh = new PDO($dsn, $dbuser, $dbpass);

// Création de la requête SQL
$nom_sql = $dbh->quote($nom) ;
$pass_sql = $dbh->quote($pass) ;
$sql = "SELECT count(*) as nbres FROM user "
      . " WHERE login=$nom_sql AND pass=$pass_sql" ;

// Exécution de la requête SQL
$result = $dbh->query($sql);
$row = $result->fetch();
$result = null ;
if($row['nbres'] == 1){
    return TRUE;
}else{
    return FALSE;
}
}
?>
```

Vérifiez que vous avez bien créé une base de données avec une base « application », que vous avez bien une table « user » contenant trois champs et incluez ce bout de code dans votre fichier `verif.php` pour que votre application fonctionne.

### La page sécurisée : `secure.php`

Pour vérifier si l'utilisateur est déjà authentifié, nous allons lire cette variable dans le tableau de session `$_SESSION[]`. Si l'utilisateur est authentifié, alors la variable `$_SESSION['nom']` existe. Dans le cas contraire, c'est que jamais l'utilisateur n'a fourni de mot de passe valide.

```
<?php
session_start() ;
// On vérifie si l'utilisateur est identifié
if ( !isset( $_SESSION['nom'] ) ) {

    // La variable de session n'existe pas,
    // donc l'utilisateur n'est pas authentifié
    // On redirige sur la page permettant de s'authentifier
    header('Location: auth.php') ;
    // On arrête l'exécution
    exit() ;
}
```

Il vous suffit maintenant d'ajouter `include('secure.php')` dans les scripts réservés pour que seules les personnes autorisées par la fonction `verification()` puissent les utiliser. Les autres seront redirigées vers le formulaire d'identification.

**Sécuriser encore plus votre application**

Si vous souhaitez améliorer la sécurité de votre application, il convient d'utiliser des fonctions de chiffrement sur les mots de passe. Pour en savoir plus, consultez le chapitre 7, et notamment la partie concernant la fonction `crypt()`.



# 12

## Gestion des objets

---

Un des gros reproches faits à PHP 4 portait sur son modèle objet très réduit. L'équipe de développement a corrigé le tir avec PHP 5. PHP propose désormais un modèle objet à simple héritage complet. Les améliorations portent principalement sur les outils d'aide au développeur (interfaces, classes abstraites, contrôle d'appel, etc.) et à l'intégration (surcharge, utilisation avec les syntaxes PHP préexistantes, autochargement des définitions). PHP permet donc, si vous le souhaitez, de programmer suivant la philosophie objet et n'a désormais plus grand-chose à envier à Java concernant la gestion de la programmation orientée objet.

### Introduction aux objets

Cette introduction a pour but de présenter très brièvement les concepts de la programmation orientée objet. Il s'agit d'une initiation permettant d'avoir les bases pour aborder les détails techniques et syntaxiques de ce chapitre. Si vous voulez comprendre la philosophie de la programmation orientée objet, nous vous conseillons de vous reporter à un ouvrage consacré au sujet.

### *Pourquoi programmer en objet ?*

La programmation par objets ou par procédures classiques est une affaire de goût, et on trouve des gens compétents quelle que soit la méthode utilisée. La programmation orientée objet comporte toutefois les avantages suivants :

- un code réutilisable – les types d'objets peuvent servir de bases pour d'autres types d'objets, en ne réimplémentant que ce qui change entre les deux.

- un code naturel – l’homme voit naturellement les choses sous forme d’objets avec des actions à y appliquer et des caractéristiques propres. Réutiliser ces éléments permet d’avoir une bonne conceptualisation.
- un code modulaire – chaque type contient son propre contexte et n’interfère avec les autres types que via des interfaces bien définies. Il est facile d’isoler un module ou de développer des modules séparément.
- un code compréhensible – chaque objet regroupe ses propriétés et ses méthodes. On comprend tout de suite à quoi s’applique une fonction et à quoi correspond une variable.

Ces avantages ne sont pas exclusifs à la programmation objet. Il est possible d’avoir un code compréhensible, réutilisable, naturel et modulaire sans objets, comme il est possible d’avoir un code incompréhensible, non réutilisable, non naturel et monolithique avec des objets. Il s’agit juste d’avoir une solution simple qui favorise ces avantages et dirige le développeur vers une bonne conception.

### ***Qu’est-ce qu’un objet ?***

Le concept d’objet en programmation est fait pour être naturel et ressembler à ce que manipule un non-informaticien. Un objet est une chose quelconque. La voiture de mon voisin est un objet, l’utilisateur de mon application est un objet, mon compte en banque peut être vu comme un objet, une écriture de mon compte en banque est aussi un objet. Toute donnée manipulée peut en fait être représentée comme un objet.

#### **Attributs et méthodes**

Chaque objet a des attributs qui lui sont propres. Mon compte en banque a un attribut qui définit le numéro de compte, un autre qui définit le solde actuel, un troisième qui est une liste des différentes opérations, et ainsi de suite. Les attributs peuvent être vus comme les caractéristiques propres de l’objet.

Les objets peuvent avoir des méthodes. Il s’agit des actions qu’on peut appliquer à un objet. Toujours en prenant mon objet de compte en banque, il existe une méthode pour le solder, une pour ajouter une écriture, une pour le déplacer dans une autre banque, etc.

### ***Qu’est-ce qu’une classe ?***

Une classe est un modèle de données. On peut la voir comme une famille d’objets. Tous les objets d’une même classe sont similaires. Ils partagent les mêmes attributs et méthodes.

On peut ainsi imaginer une classe représentant les voitures. Toutes les voitures (tous les objets de la classe voiture) ont des plaques d’immatriculation, un moteur avec une certaine puissance et un nombre de portières identifiables (ils ont des attributs communs). Tous les objets de cette classe ont aussi des méthodes pour démarrer, freiner, accélérer, tourner, etc.

## Qu'est-ce qu'une instance ?

Nous venons de voir qu'une classe était un type d'objet. Une instance est une représentation particulière d'une classe.

### Important

Un objet est une instance de classe, bien que de nombreux abus de langage assimilent un objet à une classe.

Pour donner un exemple, Mégane est une classe et une instance de cette classe pourrait être la voiture que vous venez d'acheter, qui est bleue et sans options. Une autre instance de la classe Mégane pourrait être la voiture rouge garée en bas de chez vous. En revanche, il faut faire attention au fait qu'il peut y avoir plusieurs instances ayant les mêmes propriétés sans qu'elles soient identiques. Par exemple, si quelqu'un d'autre achète la même voiture que vous, une Mégane bleue sans options, il s'agira d'une autre instance de la même classe.

## Utilisation simple des objets

### Déclarer une classe

Le mot-clé `class` permet de définir une nouvelle classe. Il doit être suivi du nom de la classe et d'un bloc entre accolades. Entre ces accolades pourront être insérés les attributs de la classe et les définitions des méthodes, ou encore des commentaires. L'exemple suivant définit une classe `voiture` :

```
class voiture {  
    // Contenu de la classe  
}
```

### Attributs

Le contenu de la classe est structuré en deux parties. La première partie permet la déclaration de tous les attributs de la classe. Ces attributs sont déclarés en utilisant la syntaxe des variables et un des mots-clés suivants : `public`, `private` ou `protected`. L'exemple suivant définit un attribut `marque` :

```
class voiture {  
    public $marque ;  
}
```

Dans un premier temps, nous utiliserons le mot-clé `public` pour déclarer les attributs de la classe. La description et la signification des différents mots-clés sont données plus loin, à la section « Sûreté de programmation ».

**Note**

Pour garder une compatibilité avec la version 4 de PHP, il est possible de déclarer l'attribut avec le mot-clé `var`, qui est équivalent au nouveau mot-clé `public`. L'utilisation de la syntaxe PHP 4 provoquera une erreur de niveau `E_STRICT` pour avertissement.

Il est possible de définir une valeur par défaut avec la même syntaxe qu'une affectation.

```
<?php
class voiture {
    public $marque = 'ferrari';
}
?>
```

**Méthodes**

La deuxième partie du contenu d'une classe permet la déclaration des méthodes. Ces méthodes se déclarent exactement comme des fonctions. Deux classes différentes peuvent utiliser des méthodes avec le même nom sans provoquer de conflit. L'exemple suivant définit une méthode `freiner` :

```
<?php
class voiture {
    public $marque ;
    function freiner( $force_de_freinage ) {
        // Instructions pour faire freiner
    }
}
?>
```

**Constantes**

La version 5 de PHP permet de définir des constantes locales à une classe. De telles constantes vous permettent de ne pas polluer l'espace global avec des constantes que vous n'utilisez que dans le cadre d'un certain type d'objet. Pour déclarer une constante dans une classe, il vous suffit de procéder comme pour une variable avec une valeur par défaut, mais en utilisant le mot-clé `const` et sans mettre le symbole `$`. Une convention classique est de spécifier les constantes en majuscules dans le code pour mieux les identifier et les isoler.

L'exemple ci-dessous permet de définir trois types de tondeuses à gazon : poussée, tractée ou autoportée.

```
<?php
class TondeuseGazon {
    const TRACTEE = 1 ;
    const AUTOPORTEE = 2 ;
    const POUSSEE = 4 ;
    public $type ;
}
?>
```

## Utilisation des objets

### Instanciation

Pour pouvoir utiliser un objet, il faut d'abord le créer à partir d'un modèle (la classe). On utilise la syntaxe d'affectation avec le mot-clé `new`. L'exemple suivant crée un objet de la classe `voiture` :

```
<?php
// Déclaration de la classe
class voiture {
    public $marque ;
    function freiner( $force_de_freinage ) {
        // Instructions pour faire freiner
    }
}

// Instanciation d'un objet
$mavoiture = new voiture() ;
?>
```

Les parenthèses lors de l'instanciation sont ici optionnelles. Elles ne seront nécessaires que quand nous utiliserons des paramètres à passer au constructeur lors de l'instanciation. Nous les utilisons pour avoir un code plus homogène.

#### Attention

Dans la version 5 de PHP, si vous n'utilisez pas le chargement automatique des classes, vous devez les déclarer avant de les utiliser. Il n'est plus possible d'utiliser des classes déclarées en fin de script.

### Utilisation d'un attribut

Une fois qu'un objet a été instancié, on peut commencer à utiliser ses attributs et ses méthodes. Les attributs s'utilisent comme des variables classiques. On accède à un attribut grâce à l'opérateur flèche (`->`). L'exemple suivant accède à l'attribut `$solde` d'un objet de type `CompteEnBanque` ; il est illustré à la figure 12-1.

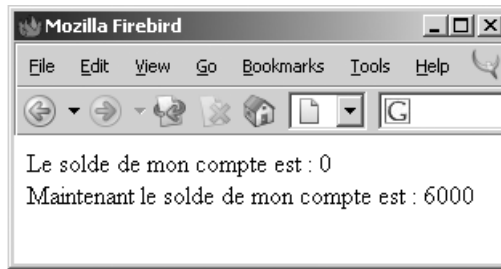
```
<?php
class CompteEnBanque {
    public $solde = 0;
}

$mon_compte = new CompteEnBanque() ;
echo 'Le solde de mon compte est : ' ;
echo $mon_compte->solde, '<br>' ; // Affiche 0

$mon_compte->solde = 6000 ;
echo 'Maintenant le solde de mon compte est : ' ;
echo $mon_compte->solde, '<br>' ; // Affiche 6000
?>
```



**Figure 12-1**  
*Utilisation  
d'un objet*



### Attention

L'attribut s'appelle sans mettre de \$ après l'opérateur ->.

PHP est assez laxiste avec la déclaration des variables. Il est possible d'utiliser des attributs non déclarés dans la classe sans aucune limitation. C'est toutefois une méthode déconseillée car par la suite la compréhension de votre code vous semblera complexe.

### Utilisation des méthodes

L'appel d'une méthode se fait de façon similaire à celui d'une fonction. Comme pour les attributs, il suffit de séparer l'objet et la méthode par l'opérateur ->. L'exemple suivant appelle la méthode klaxonne() d'un objet de type voiture :

```
<?php
class voiture {
    function klaxonne() {
        echo "Vous klaxonnez fort !" ;
    }
}
$ma_voiture = new voiture() ;
$ma_voiture->klaxonne () ; // Affiche Vous klaxonnez fort !
?>
```

### Référence à l'objet en cours

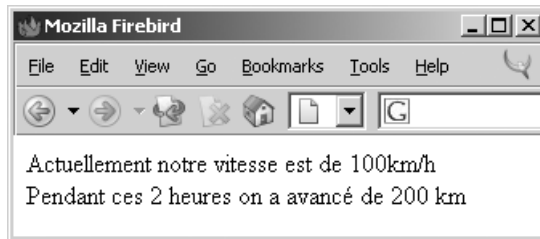
Il est souvent utile de pouvoir faire référence à l'objet en cours dans une méthode. C'est par exemple le cas pour accéder à un attribut ou lancer une autre méthode. La métavariante \$this est une référence permanente vers l'objet courant. On parle de métavariante car ce n'est pas à vous de la définir, ce sera fait automatiquement par PHP à l'exécution de chaque méthode.

Vous trouverez le résultat de l'exemple suivant à la figure 12-2.

```
<?php
class voiture {
    public $vitesse = 0;
    function avance( $temps ) {
```

```
$distance = $temps * $this->vitesse ;
echo "Pendant ces $temps heures on a avancé de $distance km" ;
}
}
$ma_voiture = new voiture() ;
$ma_voiture->vitesse = 100 ; // On avance à 100 km/h
echo 'Actuellement notre vitesse est de ' ;
echo $ma_voiture->vitesse, 'km/h<br>';
$ma_voiture->avance( 2 ) ; // On avance 2h
?>
```

**Figure 12-2**  
*Utilisation  
d'un objet*



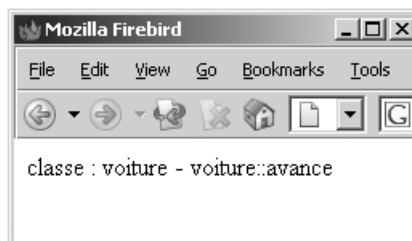
### Constantes d'aide au débogage

À tout moment, dans une méthode, vous pouvez faire appel aux métaconstantes `__CLASS__` et `__METHOD__`. Elles vous retourneront les noms de la classe et de la méthode en cours d'exécution.

Une capture d'écran de ce que donne le script suivant est présentée à la figure 12-3.

```
<?php
class voiture {
    public $vitesse ;
    function avance( $temps ) {
        echo 'classe : ', __CLASS__, ' - ', __METHOD__, '<br>' ;
    }
}
$ma_voiture = new voiture() ;
$ma_voiture->vitesse = 100 ; // On avance à 100 km/h
$ma_voiture->avance( 2 ) ;
?>
```

**Figure 12-3**  
*Utilisation  
d'un objet*



## Constantes

Les constantes s'utilisent de manière statique. L'accès statique et les fonctionnements mis en œuvre seront décrits plus loin. Les syntaxes suivantes vous permettront d'accéder aux constantes sans vous préoccuper des détails.

### Accès à une constante depuis l'extérieur

Depuis n'importe où dans le code, vous pouvez faire référence à une constante de la classe xxx grâce à l'opérateur xxx::.

```
<?php
class TondeuseGazon {
    const TRACTEE = 1 ;
    const AUTOPORTEE = 2 ;
    const POUSSEE = 4 ;
    public $type ;
}
$maTondeuse = new TondeuseGazon() ;
$maTondeuse->type = TondeuseGazon::POUSSEE ;
echo $maTondeuse->type ; // Affiche le chiffre 4
?>
```

### Accès à une constante locale

À l'intérieur d'une méthode, il est aussi possible d'accéder aux constantes de l'objet en cours grâce à l'opérateur self::.

```
<?php
class TondeuseGazon {
    const TRACTEE = 1 ;
    const AUTOPORTEE = 2 ;
    const POUSSEE = 4 ;
    public $type ;

    function setTondeusePoussee() {
        $this->type = self::POUSSEE ;
    }
}
$maTondeuse = new TondeuseGazon() ;
$maTondeuse->setTondeusePoussee() ;
echo $maTondeuse->type ; // Affiche le chiffre 4
?>
```

## Déréférencement des méthodes

En PHP 4, si une fonction ou une méthode retournait un objet, il fallait d'abord enregistrer la valeur dans une variable, et c'est seulement après qu'on pouvait utiliser l'opérateur ->.

```
class voiture {
    function cetteVoiture() {
        return $this ;
    }
}
```

```
function avance( $distance ) {
    echo "on avance de $distance mètres" ;
}
}
$maVoiture = new voiture() ;
$cetteVoiture = $maVoiture->cetteVoiture() ;
$cettevoiture->avance( 100 ) ;
// Affiche on avance de 100 mètres
```

À partir de PHP 5, on peut utiliser directement la valeur de retour d'une méthode ou d'une fonction :

```
<?php
class voiture {
    function cetteVoiture() {
        return $this ;
    }
    function avance( $distance ) {
        echo "on avance de $distance mètres" ;
    }
}
$maVoiture = new voiture() ;
$maVoiture->cetteVoiture()->avance( 100 ) ;
// Affiche on avance de 100 mètres
?>
```

### Affichage d'un objet

Quand vous essayez d'afficher directement un objet (et non un de ses attributs ou une valeur retournée par une méthode), PHP renvoie le texte « Object ».

```
class voiture {
}
$maVoiture = new voiture() ;
echo $maVoiture ; // Affiche Object
```

La version 5 de PHP apporte une méthode particulière nommée `__toString()`. Si cette méthode est définie, le résultat de son appel est affiché à la place du texte « Object ».

L'utilisation de la méthode `__toString()` est mise en exemple par le code suivant et la figure 12-4.

```
<?php
class voiture {
    function __toString() {
        return 'Vous êtes dans la classe \'voiture\'' ;
    }
}
$maVoiture = new voiture() ;
echo $maVoiture ;
?>
```

Figure 12-4

Utilisation  
de la méthode  
`__toString()`



Il est tentant d'utiliser une telle fonctionnalité pour l'intégrer dans la logique de votre application, et de la confondre avec un transtypage automatique vers une chaîne de caractères. C'est à déconseiller car cette fonctionnalité agit uniquement dans les affichages (instructions `echo` ou `print`).

```
<?php
class voiture {
    function __toString() {
        return 'une voiture' ;
    }
}
$maVoiture = new voiture() ;
echo htmlentities( $maVoiture ) ; // N'affiche PAS une voiture
// car la méthode __toString() n'est pas automatiquement appelée
?>
```

Si vous comptez utiliser la transformation en texte ailleurs (dans une fonction de traitement de texte par exemple, ou via un transtypage), il vous faudra appeler la fonction `__toString()` manuellement car PHP ne le fera pas automatiquement :

```
<?php
class voiture {
    function __toString() {
        return 'une voiture' ;
    }
}
$maVoiture = new voiture() ;

// La ligne suivantes affichent une voiture
echo htmlentities( $maVoiture->__toString() ) ;
?>
```

## Vérifier le type d'un objet

Dans vos applications, vous aurez probablement besoin de savoir de quel type est un objet, ou s'il appartient à un sous-type d'une classe connue. Vous pouvez vérifier qu'un objet spécifié appartient à une classe donnée ou à une de ses sous-classes grâce au mot-clé `instanceof`.

```
<?php
```

```
class voiture {
}
$maVoiture = new voiture() ;
if ($maVoiture instanceof voiture) {
    echo "Il s'agit d'une voiture" ;
} else {
    echo "Il ne s'agit pas d'une voiture" ;
}
?>
```

L'opérateur renvoie vrai si `$maVoiture` est de la classe `voiture`, d'une classe dérivée de `voiture` ou d'une classe implémentant l'interface `voiture` (les notions d'interface et de classe dérivée seront décrites plus bas).

Il est toutefois possible de connaître la classe exacte d'un objet et pas seulement son appartenance à une famille via la fonction `get_class()`. En fournissant un objet en paramètre, elle renvoie une chaîne de caractères contenant le nom de la classe à laquelle appartient l'objet.

```
<?php
class voiture {
}
$maVoiture = new voiture() ;
echo get_class($maVoiture) ;
// Affiche voiture
?>
```

Il est aussi possible de remonter la hiérarchie des classes et d'accéder au nom de la classe parente grâce la fonction `get_parent_class()` (la notion de classe parente sera expliquée plus loin dans ce chapitre). Vous pouvez lui fournir en paramètre soit un objet, soit un nom de classe :

```
<?php
class vehicule {
}
class voiture extends vehicule {
}
$maVoiture = new voiture() ;
echo get_class($maVoiture) ;
// Affiche voiture
echo get_parent_class($maVoiture) ;
// Affiche vehicule
echo get_parent_class('voiture') ;
// Affiche vehicule
?>
```

## Copie et référence

### Attention

Ce qui suit est une différence fondamentale avec le modèle objet de PHP 4 et sera la cause d'une grosse partie des incompatibilités de scripts PHP 4 avec PHP 5. C'est le point à surveiller si vous reprenez d'anciens scripts PHP 4.

Pour comprendre la suite, il faut définir la différence entre deux objets qui sont « identiques » et deux objets qui sont « les mêmes ». Deux objets identiques sont deux objets qui ont exactement les mêmes attributs. C'est le cas de deux écharpes vertes au magasin : elles sont indifférenciables, mais ne sont pas les mêmes et si j'en colore une en rouge, l'autre restera verte et on verra la différence. Inversement, j'ai une référence à mon compte en banque sur mon logiciel de comptabilité et mon banquier en a une chez lui. Ces deux objets sont les mêmes : si je fais un virement, il apparaîtra sur les deux.

### Le comportement PHP 4

Avec PHP 4, les objets étaient passés par copie si rien n'était spécifié. Le script suivant affichera la valeur 1 sous PHP 4 :

```
// PHP 4
class objet {
    public $nombre ;
    function affiche() {
        echo "objet ".$this->nombre ;
    }
}

$obj1 = new objet() ;
$obj1->nombre = 1 ;

$obj2 = $obj1 ; // On crée une copie sous PHP4
// Les objets sont alors identiques mais ne sont pas les mêmes

$obj2->nombre = 2 ;
echo $obj1->nombre ; // Affiche 1 sous PHP 4
// La modification de l'objet 2 n'a pas affecté l'objet 1
```

On pouvait toutefois forcer l'affectation par référence grâce à l'opérateur &, comme pour les autres types de variable. Voici les dernières lignes modifiées :

```
// PHP 4
$obj1 = new objet() ;
$obj1->nombre = 1 ;

$obj2 = &$obj1 ; // On crée une référence
// Les objets 1 et 2 sont les mêmes
```

```
$obj2->nombre = 2 ;  
echo $obj1->nombre ; // Affiche 2  
// La modification affecte les deux références du même objet
```

Ce comportement posait problème pour une utilisation intensive des objets. En effet, si on envoie un objet à une fonction ou à une méthode et qu'il y soit modifié, la modification n'apparaît pas une fois sorti de la fonction. En fournissant l'objet en paramètre, on créait une copie, la fonction manipulait un objet identique, copié sur celui qu'on avait envoyé, mais qui n'était pas celui qu'on avait envoyé.

Pour garder un comportement simple, il fallait souvent penser à fournir des références. Un simple oubli provoquait des erreurs difficiles à repérer.

### PHP 5, le passage par référence

La version 5 de PHP modifie cet état de fait et utilise par défaut l'affectation par référence quand il s'agit d'objets. Pour simplifier, on peut considérer que l'opérateur & est toujours présent, implicitement, à chaque fois qu'un objet est affecté à une variable ou passé à une fonction.

Pour mettre en évidence l'importance du passage par référence, voici un petit script qui gère des objets compte.

```
<?php // PHP 5  
class compte {  
    public $montant ;  
    function virer($valeur, $destination) {  
        $this->montant -= $valeur ;  
        $destination->montant += $valeur ;  
    }  
}  
$eric = new compte() ;  
$eric->montant = 100 ;  
$cyril = new compte() ;  
$cyril->montant = 100 ;  
$eric->virer(50, $cyril) ;  
echo $cyril->montant ; // Affiche 150  
?>
```

Présenté sous cette forme, le résultat tombe sous le sens. Pourtant, sans le passage implicite par référence, c'est-à-dire comme en PHP 4, le script aurait affiché 100 et non 150. Dans ce cas en effet, l'objet `$destination` modifié par le virement n'aurait pas été le compte de Cyril mais une copie du compte de Cyril.

Pour obtenir le même résultat en PHP 4, il aurait fallu ajouter la référence en mettant un & devant la variable concernée lors de la déclaration de la méthode `virer()` :

```
<?php // PHP 4  
class compte {  
    public $montant ;  
    function virer($valeur, &$destination) {
```



```
$this->montant -= $valeur ;
$destination->montant += $valeur ;
}
}
$eric = new compte() ;
$eric->montant = 100 ;
$cyril = new compte() ;
$cyril->montant = 100 ;
$eric->virer(50, $cyril) ;
echo $cyril->montant ; // Affiche 150
?>
```

## ***Garder la compatibilité avec PHP 4***

Si vous avez une application PHP 4 reposant massivement sur le fonctionnement par copie des objets, vous pouvez demander au moteur de garder l'ancien comportement en activant dans `php.ini` la directive de configuration `zend.zel_compatibility_mode`. Vous ne pourrez toutefois pas mixer les comportements PHP 4 et PHP 5 dans une même exécution.

## ***La copie explicite d'objet, ou clonage***

### **Clonage par défaut**

Il peut toutefois être utile de copier volontairement un objet, pour se retrouver avec deux objets distincts ayant des attributs initialement identiques. Pour copier un objet, vous pouvez utiliser le mot-clé `clone` :

```
<?php
class objet {
    public $nombre ;
    function affiche() {
        echo "objet ".$this->nombre ;
    }
}

$obj1 = new objet() ;
$obj1->nombre = 1 ;

$obj2 = clone $obj1 ; // On crée une copie, un clone
// Les objets sont alors identiques mais ne sont pas les mêmes

$obj2->nombre = 2 ;
echo $obj1->nombre ; // Affiche 1
// La modification de l'objet 2 n'a pas affecté l'objet 1
?>
```

## Clonage manuel

Le mot-clé `clone` se contente de copier bit à bit toutes les valeurs de l'objet source vers l'objet destination. Vous pouvez toutefois redéfinir son comportement en définissant une méthode `__clone()` dans votre classe. Cela peut par exemple être utile pour gérer des ressources comme des descripteurs de fichiers, des connexions vers des bases de données, des identifiants uniques, etc.

Lors d'un clonage, PHP commence par dupliquer l'objet en mémoire en copiant une à une toutes ses propriétés (publiques ou privées). Il appelle alors, si elle existe, la méthode `__clone()` de l'objet. Grâce à cette fonction, vous pouvez modifier à loisir l'objet cloné (le `$this`).

```
<?php
class test {
    public $compte = 1 ;
    function __clone() {
        $this->compte = 10 ;
    }
}

$original = new test() ;
$copie = clone $original ;
echo $copie->compte ; // Affiche 10
?>
```

## Contrôles d'accès pendant le clonage

### Note

Cette explication utilise les notions de contrôles d'accès décrites plus loin dans ce chapitre. Vous pouvez passer cette explication si vous ne les utilisez pas.

Il vous faut garder à l'esprit que votre méthode de clonage est soumise aux mêmes règles de contrôle d'accès que toutes les autres, elle ne pourra pas accéder directement aux propriétés privées des classes mères :

```
<?php
class mere {
    private $compte = 1 ;
    public function afficheMere() {
        echo "mere::compte = $this->compte <br>\n" ;
    }
}

class fille extends mere {
    public function __clone() {
        $this->compte = 10 ;
    }
}
```

```
public function afficheFille() {
    echo "fille::compte = $this->compte <br>\n" ;
}
}

$original = new fille() ;
$clone = clone $original ;

// La propriété de la classe mère n'a pas été modifiée
// car la classe fille n'a pas le droit d'y accéder
$clone->afficheMere() ; // Affiche mere:compte = 1

// C'est une nouvelle propriété publique de la classe fille
// mais de même nom qui a été modifiée à la place
$clone->afficheFille() ; // Affiche fille:compte = 10
```

Pour opérer des modifications sur des propriétés privées lors du clonage, il faut que chaque classe qui veut modifier ses propriétés privées implémente une méthode de clonage, et que vous appelez la méthode parent quand vous implémentez `__clone()`.

```
<?php
class mere {
    private $compte = 1 ;
    public function afficheMere() {
        echo "mere::compte = $this->compte <br>\n" ;
    }
    public function __clone() {
        $this->compte = 5 ;
    }
}
class fille extends mere {
    public function __clone() {
        // On modifie la propriété privée de mère en appelant
        // sa propre méthode de clonage
        parent::__clone() ;
        // Puis on crée une propriété publique de la classe fille
        $this->compte = 10 ;
    }
    public function afficheFille() {
        echo "fille::compte = $this->compte <br>\n" ;
    }
}

$original = new fille() ;
$clone = clone $original ;

// La méthode fille::__clone() a appelé mere::__clone()
// et cette dernière a pu modifier la propriété privée
$clone->afficheMere() ; // Affiche mere:compte = 5
```

```
// On a toujours une propriété publique de la classe fille
// avec un nom identique
$clone->afficheFille() ; // Affiche fille:compte = 10
```

## Égalité et identité

L'utilisation des copies et des références peut amener à des erreurs simples lors des tests d'égalité. Il faut bien faire attention à ce que l'on veut : soit tester l'égalité (tous les attributs sont égaux) ; soit tester l'identité (les deux variables référencent le même objet). L'égalité est testée par == ; l'identité est testée par === (voir le script suivant et son résultat figure 12-5) :

```
<?php
class test {
    var $var = 1 ;
}
$a = new test() ;
$b = $a ;
$c =& $b ;

if ($a == $b) echo '$a est égal à $b <br>' ;
if ($b == $c) echo '$b est égal à $c <br>' ;
if ($a === $b) echo '$a est une référence de $b <br>' ;
if ($b === $c) echo '$b est une référence de $c <br>' ;
?>
```

Figure 12-5

*Différencier  
l'égalité et l'identité*



## Constructeurs et destructeurs

### Constructeur

PHP prévoit un moyen d'initialiser un objet à sa création. Il s'agit d'exécuter une méthode spécifique lors de la création de l'objet. On parle de constructeur.

Si une méthode porte le nom `__construct()` (n'oubliez pas les deux traits de soulignements avant le nom), elle servira de constructeur. Il est possible de fournir des paramètres au constructeur en les ajoutant entre parenthèses dans l'instanciation. L'exemple suivant nous

montre comment on peut initialiser un objet en donnant des paramètres à son constructeur. Son résultat est donné à la figure 12-6 :

```
<?php
class voiture {
    public $vitesse = 0;
    public $marque;
    public $annee;
    public $modele;

    // Définition du constructeur
    function __construct($marque,$modele,$annee) {
        $this->marque = $marque;
        $this->annee = $annee;
        $this->modele = $modele;
    }

    // Définition de la fonction avance
    function avance( $temps ) {
        $distance = $temps * $this->vitesse ;
        echo "Pendant ces $temps heures on a avancé de $distance km" ;
    }

    // Définition de la fonction afficher
    function affiche( ) {
        echo 'Il s\'agit d\'une voiture de marque ' ;
        echo $this->marque, '<br>';
        echo 'Le modèle est ', $this->modele, '<br>';
        echo 'L\'année d\'achat est ', $this->annee, '<br>';
    }
}

$ma_voiture = new voiture('Renault','clio',2004) ;
$ma_voiture->affiche();
?>
```

**Figure 12-6**  
*Utilisation  
d'un constructeur*



**Attention**

Si votre classe est une classe dérivée et si le constructeur y est redéfini, seul le constructeur de la classe fille sera appelé. PHP n'appelle pas implicitement le constructeur parent, c'est à vous de le faire. Vous pouvez vous reporter à la section sur l'héritage pour plus de détails sur la redéfinition de méthodes et l'appel de la méthode de la classe parente.

**Compatibilité PHP 4**

PHP 4 utilisait un autre système, qui vérifiait la présence d'une méthode de même nom que la classe et l'utilisait comme constructeur. Cette syntaxe rendait complexe la compréhension des programmes, car lors d'un héritage, il fallait vérifier qu'aucune méthode n'avait le même nom que la classe, ou la classe parent, ou la classe parent de la classe parent, etc.

PHP 5 garde toutefois ce système pour compatibilité si aucune méthode `__construct()` n'existe. S'il trouve un constructeur PHP 5 et un constructeur PHP 4, PHP utilise le constructeur PHP 5 et produit un message d'information de niveau `E_STRICT`.

**Destructeur**

De la même manière qu'il existe un constructeur, on peut définir un destructeur. Il s'agit d'une méthode nommée `__destruct()`. Elle est automatiquement appelée quand l'objet est détruit, soit par `delete()` ou `unset()`, soit à la fin du script.

**Attention**

Le destructeur n'est appelé que lorsque la dernière référence à un objet est détruite, c'est-à-dire quand l'objet n'existe plus. Si vous effacez une variable avec `unset()` et que son contenu soit un objet, il peut rester des références ailleurs. Le destructeur ne serait alors pas appelé tout de suite. Vous pouvez forcer la destruction d'un objet à partir d'une référence avec la fonction `delete()`.

Un destructeur est pratique pour fermer les ressources ouvertes : fichiers, connexions vers des serveurs de bases de données, etc.

De même que pour le constructeur, si votre classe est une classe dérivée et si le destructeur y est redéfini, le destructeur de la classe parente ne sera pas appelé implicitement par PHP. C'est à vous de le faire explicitement.

```
<?php
class mere {
    function __destruct() {
        echo "appel au destructeur de la classe mère <br>" ;
    }
}
```

```
class fille extends mere{
    function __destruct() {
        echo "appel au destructeur de la classe fille <br>" ;
        parent::__destruct() ;
    }
}

$objet = new fille() ;
unset($fille) ;

// Affiche
// appel au destructeur de la classe fille
// appel au destructeur de la classe mère
?>
```

## La notion d'héritage

### *Définition de la notion d'héritage*

La catégorisation par classes d'objets est assez limitée. Assez souvent, plusieurs classes partagent elles-mêmes certains de leurs attributs et méthodes. On peut parler de catégories d'objets et de sous-catégories.

Il est possible d'expliquer cette relation avec la notion d'héritage. Ainsi, l'objet qui définit ma voiture appartient à la classe des voitures, qui elle-même dérive de la classe des véhicules à moteur. On dit alors que la classe des voitures hérite de la classe des véhicules à moteur : elle peut en utiliser les méthodes et les attributs.

Pour le langage PHP, une classe ne peut hériter que d'une seule classe. L'héritage multiple est interdit et il est impossible de dire par exemple que la classe des écharpes en laine hérite de la classe des écharpes et de la classe des vêtements en laine, simultanément.

### *Définition d'une classe héritée*

Pour définir un héritage, on utilise le mot-clé `extends` après le nom de la classe dans la déclaration. Dans l'exemple suivant, la classe des voitures hérite de la classe des véhicules à moteur.

Le schéma UML de la figure 12-7 correspond au code suivant :

```
<?php

class vehicule_a_moteur{
    public function freine(){
        // Code pour vehicule_a_moteur.freine ...
    }

    public function avance(){
        // Code pour vehicule_a_moteur.avance ...
    }
}
```

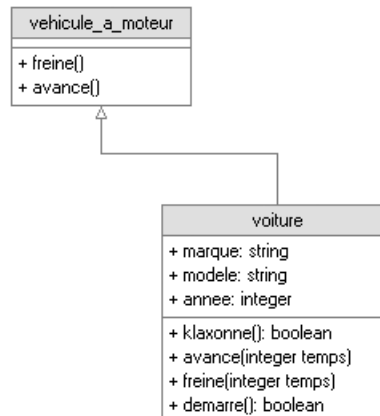
```
    }  
  }  
  
  class voiture extends vehicule_a_moteur{  
  
    public $marque;  
    public $modele;  
    public $annee;  
  
    public function klaxonne(){  
      // Code pour voiture.klaxonne ...  
    }  
  
    public function avance($temps=NULL){  
      // Code pour voiture.avance ...  
    }  
  
    public function freine($temps=NULL){  
      // Code pour voiture.freine ...  
    }  
  
    public function demarre(){  
      // Code pour voiture.demarre ...  
    }  
  }  
}  
?>
```

**Note**

Des outils permettant de conceptualiser une application PHP avec UML seront décrits à la fin du livre dans le chapitre sur les outils de développement.

**Figure 12-7**

*Notation UML  
de l'héritage*





Automatiquement, la classe `voiture` pourra utiliser tous les membres (méthodes et attributs) de la classe mère. Si la classe fille redéfinit certaines méthodes ou certains attributs déjà présents dans la classe mère, ces derniers seront remplacés.

### Redéfinition d'attribut ou de méthode

Si un attribut est redéfini, c'est sa dernière définition qui est utilisée pour déterminer la valeur par défaut de l'objet de la classe fille :

```
<?php
class vehicule_a_moteur {
    public $nombre_roues = 2 ;
}

class voiture extends vehicule_a_moteur {
    public $nombre_roues = 4 ;
}

$berline = new voiture() ;
echo $berline->nombre_roues ;
// Affiche 4
?>
```

Si une méthode est redéfinie, c'est sa dernière définition qui est utilisée :

```
<?php
class vehicule_a_moteur {
    function nombre_de_roues() {
        echo 2 ;
    }
}

class voiture extends vehicule_a_moteur {
    function nombre_de_roues() {
        echo 4 ;
    }
}

$berline = new voiture() ;
$berline->nombre_de_roues() ;
// Affiche 4
?>
```

### Héritage strict

Les développeurs du moteur PHP adhèrent au concept d'héritage strict. Il s'agit dans notre exemple sur les voitures de considérer que si la classe `voiture` hérite de la classe `vehicule`, toute instance de `voiture` doit pouvoir être manipulée comme un `vehicule` générique, en ignorant les particularités de la classe fille. Toute méthode redéfinie doit être

compatible avec la méthode précédente. Pour revenir sur notre exemple de `voiture`, bien qu'elle accepte une vitesse dans sa méthode `avance()`, elle doit aussi accepter cette même instruction sans paramètre, comme n'importe quel véhicule.

On dit que les méthodes de la classe fille doivent avoir des prototypes compatibles avec ceux de la classe mère. Il est possible d'ajouter des paramètres supplémentaires, à condition qu'ils soient facultatifs. Il est aussi possible de rendre facultatifs des paramètres en leur donnant une valeur par défaut.

Pour résumer, le nombre de paramètres obligatoires de la méthode fille doit être inférieur ou égal au nombre de paramètres possibles de la méthode mère ; le nombre de paramètres possibles de la méthode fille doit quant à lui être supérieur ou égal au nombre de paramètres possibles de la méthode mère. Par exemple, une méthode qui a trois paramètres obligatoires sur cinq peut être remplacée par une méthode qui a un paramètre obligatoire (nombre inférieur) et cinq facultatifs (donc six au total, ce qui est supérieur aux cinq initiaux).

Seuls les constructeurs ne sont pas soumis à cette règle car la notion de constructeur générique n'a pas vraiment de sens.

Cet héritage strict est une contrainte importante qui n'existait pas en PHP 4. Un message d'erreur de niveau `E_STRICT` sera envoyé pour chaque infraction. Si vous ne souhaitez pas vous confiner dans un héritage strict, il vous suffira de les ignorer (ce qui est probablement le cas par défaut). Vous pourrez trouver plus de renseignements sur ce que sont les messages `E_STRICT` dans le chapitre 19 sur la gestion des erreurs.

## Accès aux méthodes parentes

Si une méthode est redéfinie dans la classe fille, il peut être utile de volontairement appeler la méthode de la classe parente. Il suffit alors d'utiliser une notation statique (voir plus loin pour plus de détails sur ce qu'est un appel statique et comment l'utiliser) avec le mot-clé `parent`. La syntaxe `parent::xxx()` permet d'accéder à la méthode `xxx` de la classe mère.

C'est par exemple le cas si on ne veut que faire un ajout. Il suffit alors dans notre méthode de faire notre ajout puis appeler la méthode parente qui fera le gros du travail.

```
<?php
class vehicule_a_moteur {
    function avancer() {
        echo "J'avance \n" ;
    }
}

class voiture extends vehicule_a_moteur {
    function passer_la_vitesse( $vitesse ) {
        echo "Je passe la vitesse $vitesse \n" ;
    }
}
```

```
function avancer() {
    $this->passer_la_vitesse(1) ;
    parent::avancer() ;
}
}

$maVoiture = new voiture() ;
$maVoiture->avancer() ;
// Affiche "Je passe la vitesse 1" puis "J'avance"
?>
```

## Sûreté de programmation

Un des défauts mis en avant concernant le modèle objet de la version 4 est le manque de sûreté de programmation ; il a été comblé avec PHP 5. Vous retrouverez la plupart de ces principes dans d'autres langages objets comme C++ ou Java, ils ne sont guère différents dans PHP 5.

Il est important toutefois de noter que ces nouvelles fonctionnalités sont tout à fait optionnelles. Vous pouvez, si vous le souhaitez, utiliser les objets PHP 5 de la même façon que les objets PHP 4 (si ce n'est le fait qu'ils sont désormais passés par référence). Vous ne perdrez aucune possibilité technique ; ces syntaxes ne sont présentes que pour aider les développeurs à traquer les erreurs dans le développement et l'utilisation des composants, rien d'autre.

### Contrôle d'accès

Le contrôle d'accès est une fonctionnalité qui permet de filtrer les accès aux attributs et aux méthodes.

### Accès public

Une méthode (ou attribut) publique est accessible depuis toute votre application. C'est le cas par défaut, et le comportement dans PHP 4. Vous pouvez toutefois le préciser explicitement en utilisant le mot-clé `public` devant les attributs et méthodes dans la déclaration de la classe.

Les deux déclarations suivantes sont équivalentes :

```
class vehicule_a_moteur {
    var $marque ; // Affichage PHP4
    function avancer() {
        echo "J'avance \n" ;
    }
}
```

et

```
<?php
class vehicule_a_moteur {
```

```
public $vitesse ; // Affichage PHP5
public function avancer() {
    echo "J'avance \n" ;
}
}
?>
```

## Accès privé

Une méthode (ou attribut) en accès privé n'est utilisable qu'à l'intérieur même de la classe. Il est interdit d'y faire appel à partir de l'extérieur. Le mot-clé à utiliser est `private`.

```
<?php
class vehicule_a_moteur {
    private $vitesse = 30 ;
    public function avancer() {
        echo "J'avance à la vitesse de " ;
        echo $this->vitesse , " km/h \n" ;
        // Appel autorisé car on est à l'intérieur de la classe
    }
}
$maVoiture = new vehicule_a_moteur() ;
$maVoiture->avancer() ;
// Affiche "J'avance à la vitesse de 30km/h"
?>
```

Si le développeur accède tout de même par erreur à un attribut (ou à une méthode) privé depuis l'extérieur de la classe, PHP retourne une erreur fatale.

```
echo $maVoiture->vitesse ;
// On est à l'extérieur de la classe, l'appel est interdit
// PHP génère une erreur fatale
```

## Accès protégé

Un accès protégé est une solution intermédiaire entre un accès privé et un accès public. Il est toujours interdit de faire appel à une méthode ou à un attribut protégé depuis l'extérieur de l'objet, mais la méthode (ou l'attribut) est utilisable par les classes dérivées. Le mot-clé pour un accès protégé est `protected`.

```
class vehicule_a_moteur {
    protected $vitesse = 30 ;
    private $roues = 4 ;
}

class voiture extends vehicule_a_moteur {
    public function avancer() {
        echo "J'avance à la vitesse de " ;
        echo $this->vitesse , " km/h \n" ;
        // Appel autorisé car on est à l'intérieur d'une classe dérivée
    }
}
```

```
public function nombreRoues() {
    echo "il y a ", $this->roues, "\n" ;
    // Appel interdit, le nombre de roues est privé
}
}

$maVoiture = new voiture() ;
$maVoiture->avancer() ;
// Affiche "J'avance à la vitesse de 30km/h"

$maVoiture->nombreRoues() ;
// Génère une erreur fatale car la méthode essaie
// d'accéder à un attribut privé
```

### Utilisation

L'utilisation ou non des contrôles d'accès est un choix que vous devez faire au début de votre projet. Globalement, ces syntaxes permettent de repérer facilement les erreurs d'accès à certains composants.

Ces fonctionnalités sont particulièrement utiles lors d'un développement à plusieurs ou lors de la programmation d'une brique logicielle. Marquer explicitement des méthodes ou des attributs comme privés ou protégés permet de dissuader l'utilisateur de ces objets d'accéder à des détails d'implémentation qui peuvent changer par la suite.

### Redéfinition des contrôles d'accès

En dérivant une classe, vous pouvez changer le contenu d'une méthode ou la valeur par défaut d'une propriété. Dans ce cas, vous pourriez aussi être amené à redéfinir la directive de contrôle d'accès. PHP vous laisse libre de changer ces contrôles à l'unique condition que la nouvelle directive soit identique ou plus large que l'ancienne. Si une classe présente à l'extérieur une méthode publique, vous êtes assuré de pouvoir appeler cette méthode sur tous les objets de cette classe ou dérivés de cette classe.

Autrement dit, une méthode protégée peut être remplacée par une méthode de même niveau ou une méthode publique. Une méthode publique ne peut être remplacée que par une autre méthode publique. Si une méthode était déclarée privée dans la classe parente, vous pourriez la définir avec n'importe quel contrôle d'accès dans la classe fille. Dans ce dernier cas, il y a cependant des comportements non naturels à prendre en compte.

### Cas particulier des méthodes privées

Les redéfinitions de méthodes privées bénéficient d'une logique spécifique. En effet, il ne faut pas que la classe dérivée puisse accéder à l'implémentation de la classe parente.

Si vous redéfinissez une méthode privée, PHP considérera qu'il a deux méthodes de même nom simultanément dans la classe. Si c'est une méthode de la classe mère qui y fait appel, elle accèdera à la méthode privée initiale. Si inversement c'est une méthode de la classe fille qui y fait appel, elle accèdera à la nouvelle implémentation.

Ce comportement peut être source de beaucoup d'erreurs et nous vous conseillons vivement d'éviter de redéfinir des méthodes privées afin de ne pas rendre trop complexe la relecture de votre code.

Dans l'exemple suivant, bien que les méthodes `afficheMere()` et `afficheFille()` contiennent exactement le même code, elles n'auront pas le même résultat, car la méthode `affiche()` ne sera pas la même :

```
<?php
class mere {
    private function affiche() {
        echo "classe parent <br>" ;
    }
    public function afficheMere() {
        echo $this->affiche();
    }
}

class fille extends mere {
    private function affiche() {
        echo "classe dérivée <br>" ;
    }
    public function afficheFille() {
        $this->affiche() ;
    }
}

$mere = new mere() ;
$mere->afficheMere() ; // Affiche classe parent

$fille = new fille() ;
$fille->afficheMere() ; // Affiche classe parent
$fille->afficheFille() ; // Affiche classe fille
?>
```

## Typage

Il est possible de renforcer les contrôles sur les objets en donnant des informations de types sur les paramètres des méthodes. Il est en effet possible de dire à PHP de n'accepter qu'un type d'objet bien précis comme paramètre. Il suffit alors de préfixer le paramètre par le nom de la classe souhaitée.

```
<?php
class voiture {
    public $marque = 'Fiat' ;
    function faireUnAccidentAvec( voiture $autreVoiture ) {
        echo 'Il y a eu un accident avec une voiture ' ;
        echo $autreVoiture->marque ;
    }
}
```

```
$maVoiture = new voiture() ;
$autreVoiture = new voiture() ;
$maVoiture->faireUnAccidentAvec( $autreVoiture ) ;
// Affiche Il y a eu un accident avec une voiture Fiat
?>
```

Si on essaie de passer en paramètre un objet qui n'est pas de la classe spécifiée, n'en dérive pas ou n'implémente pas l'interface spécifiée (voir plus bas pour la description des interfaces et de leur fonctionnement), alors PHP renvoie une erreur fatale.

```
$maVoiture->faireUnAccidentAvec( 'Fiat' ) ;
// Provoque une erreur car 'Fiat'
// n'est pas un objet de type voiture, ni un sous-type de voiture
```

#### Attention

Seuls les types d'objets peuvent être forcés de cette manière. Il est impossible de forcer un type PHP comme une chaîne de caractères, un entier ou un tableau. Cela est fait pour éviter de compliquer l'utilisation des types de base, qui se fait normalement de manière non typée en toute transparence. Si vous tenez à forcer une telle correspondance, il vous faudra encapsuler votre donnée dans un objet.

## Classes abstraites et interfaces

Les classes abstraites et interfaces sont une étape de plus dans la sûreté de programmation. Il s'agit de s'assurer que certains types d'objets implémentent certaines méthodes afin de les utiliser sans craindre que l'objet ne fasse pas tout ce qui était prévu. Les classes finales permettent le concept inverse : s'assurer qu'une méthode ou un attribut ne sera pas redéfini par la suite dans une classe dérivée.

Les interfaces peuvent être vues comme des contrôles de qualité (vérifier que les objets correspondent bien aux spécifications) et les classes abstraites comme des implémentations incomplètes, à finir.

### Classes abstraites

Une classe abstraite est un début d'implémentation d'une classe. On définit certaines méthodes et attributs en obligeant les classes dérivées à les implémenter. On peut ainsi présenter un début d'implémentation et forcer les gens qui la réutilisent à l'étendre en la complétant.

Pour déclarer une classe ou une méthode comme abstraite, il faut en préfixer la définition par le mot-clé `abstract`. La classe suivante impose que les sous-types de véhicules implémentent une méthode pour avancer et une pour s'arrêter.

```
abstract class vehicule {
    abstract function avancer();
    abstract function arreter();
}
```

Une classe contenant une méthode abstraite (ou plusieurs) doit être déclarée comme abstraite. Si ce n'est pas le cas, PHP renvoie une erreur. Cette particularité implique qu'une classe dérivée n'implémentant pas toutes les méthodes abstraites doit aussi être déclarée comme abstraite pour ne pas provoquer une erreur.

```
$test = new vehicule() ;  
// Provoque une erreur fatale  
// car on essaye d'instancier une classe abstraite directement  
  
class voiture extends vehicule {  
    function avancer() {  
        echo 'on avance' ;  
    }  
}  
// Provoque une erreur fatale car la classe voiture  
// n'implémente rien pour s'arrêter  
// et n'est pas notée comme une classe abstraite
```

Une telle procédure permet de définir des méthodes communes à tous les types de véhicules en interdisant l'utilisation de véhicules génériques. On doit déclarer une classe qui hérite de `vehicule` et qui implémente les méthodes souhaitées pour pouvoir instancier un objet qui est de type `vehicule`.

Les classes abstraites sont généralement utilisées pour spécifier ces types d'objets très génériques, inutilisables en eux-mêmes, mais qui forcent les sous-types à avoir quelques particularités.

## Interfaces

La notion d'interface est proche de celle de classe abstraite, mais un peu plus générique. Il s'agit de définir une API (*Application Programming Interface*) qui sera utilisée par un composant. Tous les objets qui passeront par ce composant devront, pour que l'application fonctionne, proposer cette interface.

On déclare une interface de manière similaire à une classe abstraite mais avec le mot-clé `interface`. Les méthodes sont forcément publiques.

```
interface peutAvancer {  
    public function avancer();  
    public function arreter();  
}
```

On peut déclarer une classe comme implémentant une certaine interface grâce au mot-clé `implements`. Il est possible d'implémenter plusieurs interfaces en les séparant par des virgules.

```
interface faitDeLaLumiere {  
    function allumer() ;  
    function eteindre() ;  
}  
  
class voiture implements peutAvancer, faitDeLaLumiere {
```



```
function avancer() {
    echo 'on avance' ;
}
function arreter() {
    echo 'on arrete' ;
}
function allumer() {
    echo 'les phares sont allumés' ;
}
function eteindre() {
    echo 'les phares sont éteints' ;
}
}
```

De même que pour les classes abstraites, si une classe implémente seulement une partie des interfaces, elle doit être spécifiée comme abstraite pour ne pas provoquer une erreur fatale. Ni cette classe abstraite ni l'interface ne peuvent être instanciées : il faut déclarer une classe complète, implémentant toutes les méthodes des interfaces.

Les interfaces sont utilisées pour déclarer qu'un objet se conforme à un certain comportement, ou qu'il peut gérer un certain type d'événements et de fonctionnalités.

#### Note

Les classes abstraites peuvent aussi implémenter des interfaces. Les interfaces peuvent hériter d'une ou plusieurs autres interfaces (dans ce cas, il faut séparer les différents noms des interfaces héritées par des virgules).

Les interfaces sont utiles pour forcer la présence de méthodes et de fonctionnalités appelables par l'utilisateur. Seules des méthodes publiques peuvent donc être déclarées dans une interface.

Une différence importante entre une classe abstraite et une interface est qu'il est possible pour une classe d'implémenter plusieurs interfaces alors qu'on ne peut hériter que d'une classe abstraite.

### Classes et méthodes finales

Le concept de méthodes finales est l'opposé de celui de méthodes abstraites. Il s'agit de déclarer à PHP qu'aucune classe dérivée n'a le droit de modifier l'implémentation de la méthode. Une classe dérivée essayant de déclarer une méthode au même nom provoquera une erreur fatale. Le mot-clé à ajouter devant la méthode est `final`.

```
class voiture extends vehicule {
    final function avancer() {
        echo 'on avance' ;
    }
}
```

Il est de la même façon possible de définir une classe comme finale. Elle ne pourra alors plus être dérivée, toute l'implémentation est considérée comme bloquée.

```
final class voiture extends vehicule {
    function avancer() {
        echo 'on avance' ;
    }
}
```

## Accès statiques

### Accès à une classe arbitraire

On appelle un accès statique à une méthode ou à un attribut un appel direct à la classe, sans instancier d'objet. On traite alors les méthodes et attributs comme des fonctions et des variables classiques. Il n'y a plus de contexte (le `$this` n'est pas défini).

On appelle statiquement une méthode ou un attribut en préfixant son nom par celui de la classe :

```
class voiture {
    public $roues = 4 ;
    function classique() {
        return $this->roues ;
    }
    function statique() {
        return 4 ;
    }
}
// Accès statique
echo voiture::statique() ; // Affiche 4
echo voiture::classique() ; // N'affiche rien, génère une erreur
// Accès classique
$mavoiture = new voiture() ;
echo $mavoiture->classique() ; // Affiche 4
```

### Définition en vue d'un accès statique

Il est possible de déclarer explicitement une méthode ou un attribut comme statique. Ils ne seront alors appelables que statiquement. Il vous suffit de préfixer la déclaration par le mot-clé `static`.

```
<?php
class voiture {
    static $roues = 4 ;
    static function statique() {
        return 4 ;
    }
}
```

```
echo voiture::statique() ; // Affiche 4
echo voiture::$roues ; // Affiche 4
?>
```

## Accès à la classe en cours

Lors d'un appel statique, on perd tout contexte. Il n'est plus possible d'utiliser une syntaxe comme `$this->methode()` (ou alors le `$this` fait référence à l'objet qui appelle, pas à la classe appelée).

Il peut être toutefois intéressant de référencer une autre méthode (ou un autre attribut) statique de la même classe. Pour ce faire, vous pouvez utiliser le mot-clé `self`, il désigne la classe en cours (à différencier de `$this` qui référence l'objet en cours).

```
<?php
class voiture {
    static $roues = 4 ;
    static function nombreRoues() {
        return self::$roues ;
    }
}
echo voiture::nombreRoues() ; // Affiche 4
?>
```

## Accès à la classe parente

Dans la partie sur l'héritage, nous avons parlé d'un opérateur `parent::` sans le détailler. Il s'agit en fait simplement d'un appel statique avec un mot-clé `parent` qui référence la classe dont hérite la classe en cours.

Utiliser cette syntaxe permet d'accéder aux méthodes parentes. Si vous êtes dans le contexte d'un objet, l'appel statique à la classe parente ne fait pas disparaître le `$this`. On peut donc se servir de toutes les méthodes parentes, statiques ou pas.

```
class vehicule {
    public function affiche() {
        echo 'Il y a ', $this->roues, ' roues' ;
    }
}
class voiture extends vehicule {
    public $roues = 4 ;
    public $portieres = 2 ;
    public function affiche() {
        parent::affiche() ;
        echo ' et ', $this->portieres, ' portières' ;
    }
}
$maVoiture = new voiture() ;
$maVoiture->affiche() ;
// Affiche Il y a 4 roues et 2 portières
```

## Chargement automatique

Quand vous utilisez une classe, vous devez penser à toujours inclure les fichiers de définition nécessaires. Pour cela, les fonctions `include_once()` et `require_once()` sont utiles, mais elles imposent de tenir à jour les définitions incluses quand on programme et qu'on ajoute des objets.

Pour faciliter l'utilisation des objets, PHP 5 propose un système de chargement automatique. Si vous tentez d'instancier un objet d'une classe inexistante, PHP va chercher une fonction nommée `__autoload()` (ne pas oublier les deux soulignements en préfixe). Si cette fonction est présente, elle est appelée avec le nom de la classe manquante en paramètre ; charge à elle de déclarer la classe ou de charger le fichier PHP contenant sa définition.

L'exemple suivant vous montre un exemple d'implémentation simple :

```
function __autoload( $nom_de_classe ) {  
    require_once( 'lib/'.$nom_de_classe.'.class.php' ) ;  
}  
$objet = new MaClasse() ;
```

## Utilisation via les sessions

Les objets sont sauvegardés dans les sessions comme n'importe quelle autre variable PHP (vous pouvez vous reporter au chapitre dédié aux sessions pour plus de détails). Par défaut, seuls les contenus des attributs et le nom de la classe sont sauvegardés. Quand la session est réouverte, PHP crée un objet de cette classe et restaure la valeur des attributs.

Cette procédure suffit pour tous les objets simples mais risque de poser problème si vos objets accèdent à certaines ressources comme des objets DOM (Document Object Model), des connexions réseau ou des fichiers. Les ressources ne peuvent être sauvegardées en session car elles impliquent des types de données plus complexes que de simples chaînes de caractères. Le stockage par défaut des sessions peut aussi poser problème si vos objets ont des liens entre eux : vous risquez vite d'avoir des objets qui ont des références circulaires (A référence B et B référence A).

### Utilisation de `__sleep()` et `__wakeup()`

Pour vous permettre de gérer plus facilement la mise en session, PHP fournit deux événements : quand l'objet est stocké en session (on parle de mise en sommeil), PHP appelle la méthode `__sleep()` de l'objet si elle est présente. Au réveil (quand la session est relue), c'est la méthode `__wakeup()` qui est utilisée.

Il est ainsi possible de fermer un fichier ouvert dans `__sleep()`, sauvegarder son chemin dans un attribut, puis le rouvrir dans `__wakeup()` afin qu'il soit de nouveau utilisable quand on récupère la session.

La fonction `__sleep()` doit retourner un tableau contenant le nom des différents attributs qui doivent être sauvegardés. Si la fonction ne retourne rien, aucun attribut ne sera sauvegardé dans la session. Il vous est ainsi possible de déterminer des attributs qui ne seront pas sauvegardés en session.

## Surcharge

### Attention

Le terme « surcharge » est ici employé dans une définition qui ne sera pas forcément la même que dans d'autres langages. Il ne s'agit pas ici de définir plusieurs prototypes différents d'une même méthode. Pouvoir spécifier plusieurs prototypes n'est en effet pas utile avec PHP puisque les paramètres acceptent par défaut des données sans vérification de type.

PHP définit trois méthodes dites de surcharge. Il s'agit d'intercepter les appels à des attributs ou à des méthodes non définis. Ces trois méthodes vous permettent de répondre à PHP comme si la méthode ou l'attribut existait. Elles sont toutes optionnelles : elles seront utilisées si elles existent, le comportement par défaut sera adopté dans le cas contraire.

Ces surcharges ont deux principales fonctions. Elles permettent tout d'abord de faire évoluer les objets sans avoir à toucher au code qui les utilise. Si par exemple vous aviez l'habitude d'utiliser directement des attributs, vous pouvez passer à un système de méthodes `get` et `set` de manière transparente.

L'autre utilisation des surcharges PHP est de créer des objets d'interface qui ne savent pas quelles peuvent être les méthodes ou attributs appelés. On peut par exemple utiliser ce comportement pour interfacer un service web. La classe locale qui fait l'interface ne sait pas quelles méthodes sont disponibles sur le service, elle ne définit donc aucune méthode. Quand une méthode est appelée, le nom est récupéré puis renvoyé à l'objet distant.

### Note

L'utilisation des méthodes suivantes doit être faite avec raison. Elles peuvent facilement mener à du code dit « spaghetti » et rendre difficile la relecture ou la compréhension.

Elles comportent de plus une limitation sur leur utilisation. Il n'est pas possible d'utiliser ces fonctions pour affecter ou retourner une données par référence. Cette limitation d'ailleurs un effet de bord qui empêche d'utiliser des attributs virtuels (`__get`) qui renvoient des tableaux dans certains contextes.

## Affectations des attributs

Si votre classe implémente une méthode `__set()`, cette méthode sera appelée à chaque fois qu'on essaye d'affecter une valeur à un attribut inexistant. Cette méthode doit accepter deux paramètres, le nom de l'attribut et sa valeur.

```
<?php
class maClasse {
    private $valeurs = array() ;
    public function __set($nom, $valeur) {
        echo "on essaie de mettre '$valeur' dans $nom" ;
        $this->valeurs[$nom] = $valeur ;
    }
}
$monObjet = new maClasse() ;
$monObjet->propriete = 'texte' ;
// Affiche on essaie de mettre 'texte' dans propriete
?>
```

## Lecture d'attribut (Mutator)

Parallèlement à `__set()`, il existe un `__get()`. Si vous l'implémentez, cette méthode sera appelée à chaque fois qu'on essaiera de lire un attribut inexistant dans un objet de la classe. Elle prend un paramètre qui définit le nom de l'attribut auquel on essaye d'accéder. La valeur de retour de `__get()` sera utilisée par PHP comme valeur de l'attribut, comme s'il existait.

```
<?php
class maClasse {
    private $valeurs = array() ;
    public function __set($nom, $valeur) {
        echo "on essaie de mettre '$valeur' dans $nom \n" ;
        $this->valeurs[$nom] = $valeur ;
    }
    public function __get($nom) {
        echo "on essaie de lire $nom \n" ;
        return $this->valeurs[$nom] ;
    }
}
$monObjet = new maClasse() ;
$monObjet->propriete = 'texte' ;
echo $monObjet->propriete ;
// Affiche
// On essaie de mettre 'texte' dans propriete
// On essaie de lire propriete
// texte
?>
```

La surcharge d'attributs peut se révéler très utile pour maintenir et étendre les fonctionnalités de classes qui ont été mal prévues au départ. Si vous aviez décidé d'utiliser un attribut directement en lecture et en écriture et que par la suite vous cherchez à faire un calcul lors de ces écritures et lectures, il vous suffit d'ajouter un `__get()` et un `__set()` qui implémentent cette logique. Ces deux méthodes peuvent aussi vous servir à initialiser un attribut non existant quand il est appelé.

## Appel d'une méthode (Accessor)

De manière similaire à `__get()` et `__set()`, la méthode `__call()` intercepte les appels aux méthodes inexistantes. La méthode reçoit deux paramètres : le nom de la méthode et un tableau de paramètres. PHP répercutera la valeur de retour comme si elle venait de la méthode appelée et non de la fonction de surcharge.

```
<?php
class maClasse {
    public function __call($nom, $valeur) {
        echo "on essaye d'appeler $nom avec '$valeur[0]' \n" ;
        return strlen($valeur[0]) ;
    }
}
$monObjet = new maClasse() ;
echo 'longueur : ', $monObjet->longueur('texte') ;
// Affiche
// On essaie d'appeler longueur
// avec 'texte'
// longueur : 5
?>
```

## Itérateurs

Le début du chapitre s'est concentré sur la description du fonctionnement des objets et leur utilisation. Pour une utilisation plus naturelle, il reste à intégrer la programmation orientée objet aux structures de contrôles de PHP. Le motif d'itérateur permet d'utiliser les objets avec les boucles `foreach()`. On peut voir un objet implémentant les méthodes d'itération comme une liste évoluée.

`foreach()` permet normalement de manipuler les éléments d'un tableau un à un de manière simple. Utiliser cette fonction avec les objets veut simplement dire à PHP que l'objet manipulé est un groupe d'objets, et qu'on fournit des méthodes standards pour naviguer entre ces sous-objets.

### Utilisation simple

Utiliser un objet avec `foreach()` est en fait très simple. Il vous suffit pour cela d'implémenter l'interface `Iterator`. Dès lors, PHP pourra traverser votre objet avec `foreach()`

comme s'il s'agissait d'un tableau. Votre objet pourra alors retourner une liste de couples clé/valeur lisible par PHP.

```
$liste = new ListeUtilisateur ;
foreach($liste as $uid => $login) {
    echo "Utilisateur n° $uid : $login <br />" ;
}
```

L'interface `Iterator` vous impose d'implémenter les quelques méthodes nécessaires à PHP pour récupérer les couples clé/valeur pour retourner les éléments à PHP : `current()`, `key()`, `next()`, `valid()` et `rewind()`.

On considère qu'il existe une sorte de curseur qui pointe sur un élément de la liste. Ces méthodes permettent soit de récupérer l'élément pointé par le curseur, soit de déplacer le curseur sur un autre élément.

La première méthode appelée est la méthode `rewind()`. Elle demande à l'objet d'itération de faire pointer le curseur sur le premier élément de la liste. Par la suite, PHP cherche à savoir si un élément est disponible avec la méthode `valid()` ; elle doit renvoyer `TRUE` s'il reste un élément disponible, `FALSE` sinon.

Si `valid()` a retourné une valeur évaluée à vrai, PHP fait appel à `current()` et `key()`. Ces deux méthodes doivent retourner la clé et la valeur de l'élément pointé par le curseur. La méthode `next()` est ensuite utilisée pour déplacer le curseur sur l'élément suivant. PHP reprend alors la fonction `valid()` et tourne en boucle tant qu'elle renvoie une valeur évaluée à `TRUE`.

L'exemple suivant implémente un itérateur pour gérer une liste d'utilisateurs. Son résultat est affiché à la figure 12-8.

```
<?php
class ListeUtilisateur implements Iterator {
    private $login = array();
    private $uid = array() ;
    private $index = 0;

    public function current() {
        return $this->login[$this->index];
    }
    public function next() {
        $this->index += 1 ;
    }
    public function valid() {
        return ($this->index +1 <= count($this->login)) ;
    }
    public function key() {
        return $this->uid[$this->index];
    }
    public function rewind() {
        $this->index = 0 ;
    }
}
```



```

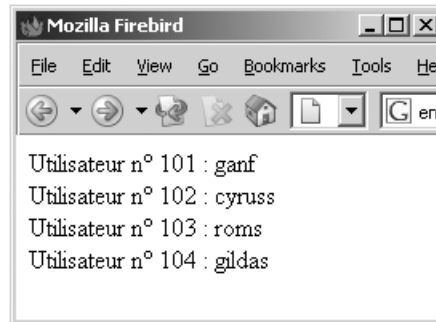
public function ajouteUtilisateur($uid, $login) {
    $this->login[] = $login ;
    $this->uid[] = $uid ;
}
}

$liste = new listeUtilisateur() ;
$liste->ajouteUtilisateur(101, 'ganf') ;
$liste->ajouteUtilisateur(102, 'cyruss') ;
$liste->ajouteUtilisateur(103, 'roms') ;
$liste->ajouteUtilisateur(104, 'gildas') ;

foreach($liste as $uid => $login) {
    echo "Utilisateur n° $uid : $login <br />" ;
}

```

Figure 12-8

*Exemple d'itérateur***Remarque**

Grâce à cette syntaxe, il est possible que `foreach()` renvoie plusieurs éléments avec la même clé. La clé n'est assurée d'être unique que dans un tableau. Vous ne pourrez donc pas forcément lire un objet avec `foreach()` et remplir un tableau avec le résultat : vous auriez des pertes.

Dans notre exemple, on voit mal l'intérêt d'un objet par rapport à un tableau. Maintenant, on peut imaginer que la classe `ListeUtilisateur` soit une classe de base et qu'on implémente deux classes dérivées, une pour Microsoft Windows et une pour Unix. Voilà un début d'exemple de ce qu'on pourrait avoir pour Unix :

```

// Ceci n'est qu'un exemple et ne doit pas être utilisé
// tel quel pour gérer votre fichier de mots de passe système

class ListeUtilisateurUnix extends ListeUtilisateur {
    public function __construct() {
        $this->chargeFichierSysteme() ;
    }
    protected function chargeFichierSysteme() {

```

```
$this->uid = array() ;
$this->login = array() ;
$this->index = 0 ;
$utilisateurs = file('/etc/passwd') ;
foreach($utilisateurs as $utilisateur) {
    $param = explode(':', $utilisateur) ;
    if (trim($param[0]) !== '') {
        $this->uid[] = $param[2] ;
        $this->login[] = $param[0] ;
    }
}

public function ajouteUtilisateur($uid, $login, $passwd=NULL) {
    $salt = md5(uniqid(mt_rand(), 1));
    $crypt = crypt($passwd, $salt) ;
    $fp = fopen('/etc/passwd', 'a') ;
    fputs($fp, "\n$login:$crypt:$uid:100:/home/$uid:/bin/bash");
    fclose($fp) ;
    $this->login[] = $login ;
    $this->uid[] = $uid ;
}
}
```

### Utilisation complète

Avoir toute l'interface d'itération dans l'objet n'est pas forcément pratique. PHP donne la possibilité à votre classe de retourner un objet qui servira d'interface pour `foreach()`.

Si votre classe implémente l'interface `IteratorAggregate`, elle devra implémenter une méthode `getIterator()`. Cette méthode doit retourner un objet implémentant l'interface `Iterator` et qui sera utilisé au final par `foreach()`.

```
<?php
class groupeCouleurs implements IteratorAggregate {
    public $couleurs = array() ;
    function ajouteCouleur($couleur) {
        $this->couleurs[] = $couleur ;
    }
    function getIterator() {
        $iterator = new GroupeCouleursIterator($this->couleurs) ;
        return $iterator ;
    }
}

class GroupeCouleursIterator implements Iterator {
    private $array = array() ;
    private $key ;
    private $current ;
    function __construct( $array ) {
        $this->array = $array ;
    }
}
```

```
function rewind() {
    reset($this->array);
    $this->next();
}
function valid() {
    return $this->key !== NULL;
}
function key() {
    return $this->key;
}
function current() {
    return $this->current;
}
function next() {
    list($key, $current) = each($this->array);
    $this->key = $key;
    $this->current = $current;
}
}
$couleurs = new GroupeCouleurs() ;
$couleurs->ajouteCouleur('bleu') ;
$couleurs->ajouteCouleur('vert') ;
$couleurs->ajouteCouleur('rouge') ;
$couleurs->ajouteCouleur('jaune') ;
echo "Couleurs : " ;
foreach( $couleurs as $couleur ) {
    echo " $couleur" ;
}
// Affiche
// Couleurs : bleu vert rouge jaune
```

## Notations d'index

De la même façon que les itérateurs avec `foreach()`, il est possible d'intégrer les objets avec les syntaxes de tableau, plus exactement la notation `$variable[index]`.

Pour obtenir ce fonctionnement, vous devez implémenter l'interface `ArrayAccess` dans la classe concernée. Cette interface vous impose la définition de quatre méthodes : `offsetExists()` définit si une valeur existe à l'index spécifié en paramètre ; elle renvoie `TRUE` si c'est le cas, `FALSE` sinon. La méthode `offsetGet()` permet de lire une valeur en spécifiant son index ; `offsetSet()` fait l'opération inverse et affecte la valeur en second paramètre à l'index fourni en premier. Finalement `offsetUnset()` permet d'effacer un élément à partir de son index.

Voici un exemple d'implémentation qui mime totalement le fonctionnement d'un tableau :

```
<?php
class tableau implements ArrayAccess {
    private $tableau = array() ;
```

```
function offsetExists( $index ) {
    return isset( $this->tableau[$index] ) ;
}
function offsetGet( $index ) {
    return $this->tableau[$index] ;
}
function offsetSet( $index, $valeur ) {
    return $this->tableau[ $index ] = $valeur ;
}
function offsetUnset( $index ) {
    unset( $this->tableau[ $index ] ) ;
}
}

$tab = new tableau() ;
if ( !isset($tab[42]) ) {
    $tab[42] = 'texte' ;
}
echo $tab[42] ; // Affiche texte
unset( $tab[42] ) ;
```

## Auto-incrémentation

L'auto-incrémentation (et la décrémentation) est un cas particulier. PHP lit la valeur et l'incrémente dans la foulée, sans repasser par la méthode d'affectation. Pour pouvoir utiliser ces syntaxes, il faut que la méthode `offsetGet()` renvoie la valeur par référence et non par copie. Dans le cas contraire, PHP retourne une erreur.

```
<?php
class tableau implements ArrayAccess {
    private $tableau = array() ;
    function offsetExists( $index ) {
        return isset( $this->tableau[$index] ) ;
    }
    function &offsetGet( $index ) {
        // Notez le & devant le nom de la fonction
        return $this->tableau[$index] ;
    }
    function offsetSet( $index, $valeur ) {
        return $this->tableau[ $index ] = $valeur ;
    }
    function offsetUnset( $index ) {
        unset( $this->tableau[ $index ] ) ;
    }
}

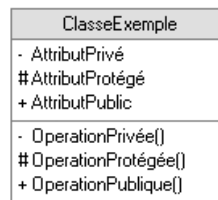
$tab = new tableau() ;
if ( !isset($tab[42]) ) {
    $tab[42] = 1 ;
}
echo ++$tab[42] ; // Affiche 2
unset( $tab[42] ) ;
```

## Coupler PHP et UML

L'utilisation de la programmation orientée objet tend souvent à la construction d'applications complexes dans lesquelles le besoin d'outils facilitant la compréhension de l'application est nécessaire.

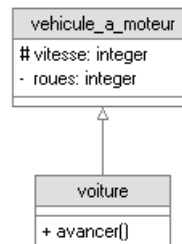
L'outil généralement reconnu pour remplir ce rôle est UML (*Unified Modeling Language*). Il s'agit d'un langage graphique de représentation des processus informatiques, et en particulier des applications orientées objet. Avec le diagramme de classes UML, chaque classe est représentée à l'aide d'un rectangle contenant ses propriétés et ses méthodes. La visibilité de ses membres est lisible d'un seul coup d'œil à l'aide des signes présents en amont des noms (- pour privé, # pour protégé et + pour public).

**Figure 12-9**  
*Classe UML*



Dans un diagramme de classes, les relations entre les classes sont également représentées ; par exemple, l'héritage est représenté par une flèche comme dans l'exemple suivant :

**Figure 12-10**  
*Héritage en UML*



À toute application orientée objet correspond une représentation graphique de la sorte. Cette représentation a plusieurs avantages :

- permettre d'avoir une vue de plus haut niveau sur l'application ;
- voir les relations entre les différents composants (la densité des relations aussi appelées couplage) ;
- pouvoir communiquer plus facilement sur une application de plus grande envergure à l'aide d'un support visuel connu de tous les membres de l'équipe.

UML est un vaste sujet qui prendrait à lui seul un ouvrage entier pour en débattre et nous vous conseillons, si vous souhaitez approfondir ce sujet, de vous reporter à des ouvrages spécifiques. Vous pourrez toutefois retrouver au chapitre sur les outils de développement deux IDE (*Integrated Development Environment*) qui vous permettront de modéliser vos graphes UML et de préparer des squelettes de code PHP en conséquence.

**Note**

P. ROQUES. – **Les Cahiers du programmeur UML** – *Modéliser un site e-commerce*. Eyrolles 2002.

P. ROQUES, F. VALLÉE. – **UML 2.0 en action** – *De l'analyse des besoins à la conception*. Eyrolles 2004.

## Introspection

L'introspection (*reflection* en anglais) permet d'inspecter du code pendant son exécution. Il est alors possible de connaître les différentes méthodes disponibles pour un objet, de connaître les classes parentes, les attributs, etc. Outre la phase de découverte, il est possible de manipuler tous ces objets trouvés : appeler une méthode par exemple.

L'introspection peut être vue comme une méthode pour décrire et manipuler le code en cours d'exécution. Toutes ses fonctionnalités sont orientées objet et vous aurez besoin d'une bonne compréhension des différentes fonctionnalités objet pour vous en servir.

On s'en sert principalement dans deux cas :

- L'exploration de classes internes du système comme celles qui composent les extensions `mysqli` ou `simplexml` : vous pourrez alors résoudre un problème de documentation.
- L'utilisation des classes et des méthodes de manière hautement dynamique : par exemple décider lors de l'exécution si vous souhaitez appeler la méthode A ou la méthode B. C'est particulièrement intéressant pour gérer des greffons (les plug-ins que nous connaissons tous) et découvrir leurs fonctionnalités de manière dynamique.

## Principes pour démarrer

### Les différents objets à manipuler

PHP définit une classe par objet qu'on peut manipuler via l'introspection :

- `ReflectionFunction` fait référence à une fonction ;
- `ReflectionClass` à une classe ;
- `ReflectionException` à une exception ;
- `ReflectionObject` à une instance de classe (un objet) ;
- `ReflectionProperty` à un attribut dans un objet ;
- `ReflectionMethod` à une méthode ;
- `ReflectionParameter` à un paramètre de fonction ou de méthode ;
- et `ReflectionExtension` à une extension de PHP.

## Instanciation et premier export

Tout d'abord, on instancie un objet de la bonne classe (voir la liste précédente pour les différentes possibilités) en lui donnant en paramètre le nom de la classe, de la méthode ou de la fonction à décrire :

```
// On cherche à décrire la classe SimpleXMLElement
$classeReflexion = new ReflectionClass( 'SimpleXMLElement' );
// On cherche à décrire la fonction strlen()
$fonctionStrlen = new ReflectionFunction( 'strlen' );
// On décrit l'extension DOM de PHP
$extensionDom = new ReflectionExtension( 'dom' );
```

Avant d'aller plus loin, il est possible de demander à PHP de nous décrire le contenu de ces objets. On utilise alors la méthode statique `export()` de la classe `reflection` :

```
<?php
// On cherche à décrire la classe SimpleXMLElement
$classeReflexion = new ReflectionClass( 'SimpleXMLElement' );
// On demande à PHP d'afficher le contenu de cet objet
echo '<pre>' ;
Reflection::export( $classeReflexion );
echo '</pre>' ;
?>
```

Dans notre exemple, nous obtenons l'affichage texte correspondant à la figure 12-11 :

**Figure 12-11**  
*Exemple d'export*



```
Class [ class SimpleXMLElement implements Traversable ] {
  - Constants [0] {
  }
  - Static properties [0] {
  }
  - Static methods [0] {
  }
  - Properties [0] {
  }
  - Methods [5] {
    Method [ final public method __construct ] {
    }
    Method [ public method asXML ] {
    }
    Method [ public method xpath ] {
    }
    Method [ public method attributes ] {
    }
    Method [ public method children ] {
    }
  }
}
```

On peut donc voir qu'il s'agit d'une classe qui implémente l'interface `Traversable` (pour gérer les itérateurs), qui a cinq méthodes internes (définies par PHP et non par l'utilisateur) : un constructeur qui ne peut pas être dérivé, et quatre méthodes publiques (`asXML()`, `xpath()`, `attributes()` et `children()`).

**Note**

Vous pouvez donc profiter de cet export pour découvrir les possibilités qu'offrent les classes d'inspection. Il suffit de les explorer directement pour connaître les méthodes définies. Elles sont généralement assez explicites. Il vous suffit de remplacer `SimpleXMLElement` par `ReflectionClass` dans notre exemple précédent.

## Les fonctions

### Nom et emplacement dans les sources

Quand on inspecte une fonction, on accède à quelques méthodes spécifiques. La première `getName()` retourne le nom de la fonction instanciée. Il est possible de connaître le fichier où est définie la fonction via la méthode `getFileName()`. Les méthodes `getStartLine()` et `getEndLine()` renvoient quant à elles les numéros de ligne de début et fin de la fonction.

```
<?php
function mafonctiontest() {
    echo "hello" ;
}
$fct = new ReflectionFunction( 'mafonctiontest' ) ;
echo 'La fonction se nomme : ' . $fct->getName() . "<br>\n";
echo 'Elle est définie dans le fichier ' . $fct->getFileName() ;
echo ' entre les lignes ' . $fct->getStartLine() ;
echo ' et ' . $fct->getEndLine() ;
?>
```



Figure 12-12

*Nom et emplacement dans les sources*



En plus de ces trois méthodes, nous avons deux méthodes `isUserDefined()` et `isInternal()` qui retournent un booléen pour savoir respectivement si la fonction est une fonction utilisateur ou une fonction native de PHP.

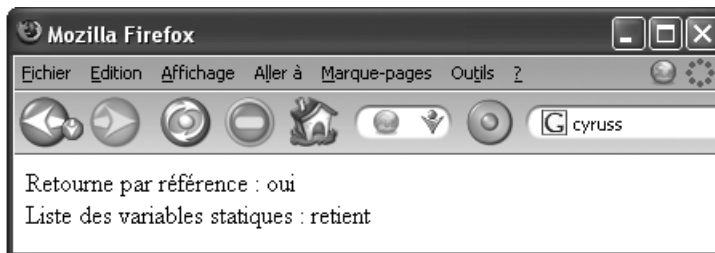
### Implémentation de la fonction

Il est possible d'aller plus loin dans l'exploration de la fonction et en trouver les détails d'implémentation. Cette partie vous permettra par exemple de découvrir comment est implémentée une fonction dans une extension PHP mal documentée, ou comment explorer le contenu d'un fichier de plug-in dans une application.

Ainsi, la méthode `returnsReference()` retourne un booléen vrai si la fonction retourne une valeur par référence (la déclaration est précédée du symbole `&`) ou par copie (par défaut). La méthode `getStaticVariables()` permet de récupérer les variables statiques d'une fonction et leur valeur. Elle retourne un tableau associatif de ces variables avec le nom comme clé.

```
<?php
function &incrimente() {
    static $retient = 0 ;
    return $retient++ ;
}
$fct = new ReflectionFunction( 'incrimente' ) ;
echo 'Retourne par référence : ' ;
echo (($fct->returnsReference())?'oui':'non' ). "<br>\n" ;
echo 'Liste des variables statiques : ' ;
echo implode( ', ', array_keys( $fct->getStaticVariables() ) );
?>
```

**Figure 12-13**  
*Information sur une fonction*



Les méthodes `getNumberOfParameters()` et `getNumberOfRequiredParameters()` retournent respectivement les nombres de paramètres et les nombres de paramètres obligatoires dans la fonction. La différence entre les deux est donc le nombre de paramètres qui ont une valeur par défaut, ceux qui sont facultatifs.

Il est aussi possible de connaître plus de détails sur ces paramètres (leur nom et leur valeur par défaut) avec la méthode `getParameters()`. Elle retourne un tableau d'objets de type `ReflectionParameter`.

## Les paramètres

Les paramètres sont des objets retournés par la méthode `getParameters()` (cette méthode a été vue pour les fonctions mais elle est aussi valable pour les méthodes de classe). Ils implémentent la classe `ReflectionParameter`.

La méthode `getName()` permet de connaître le nom de variable utilisé lors de la déclaration de la fonction. La méthode `isPassedByReference()` retourne un booléen vrai si le paramètre est récupéré par référence, et faux sinon. La méthode `isOptional()` permet, elle, de savoir si le paramètre est optionnel, et finalement la méthode `getDefaultValue()` permet de récupérer la valeur par défaut d'un paramètre optionnel.

```
<?php
function aleas($min=0 , $max=15) {
    return mt_rand();
}
$fct = new ReflectionFunction( 'aleas' ) ;
$params = $fct->getParameters() ;
$min = $params[0] ;
echo $min->getName() . ' => ' . $min->getDefaultValue() ;
```

## Gestion des documentations en ligne

Il existe des briques logicielles qui permettent de générer une documentation automatique à l'aide des commentaires du code source. Il suffit d'utiliser une syntaxe spécifique, proche de Javadoc (utilisée pour le langage Java). Vous pouvez trouver un descriptif des syntaxes utilisées par PHP sur le site <http://www.phpdoc.org/>.

Cette documentation en ligne peut être extraite via la méthode `getDocComment()`. Vous pourrez par ce biais générer facilement la documentation de votre application.

```
<?php
/**
 * Retourne un nombre aléatoire
 *
 * @return int
 */
function aleas() {
    return mt_rand();
}
$fct = new ReflectionFunction( 'aleas' ) ;
echo $fct->getDocComment() ;
?>
```

**Figure 12-14**  
*Documentation  
en ligne*



## Exécution de la fonction

Enfin, il est possible d'exécuter une fonction décrite par les objets d'introspection. Il suffit d'exécuter la méthode `invoke()` avec les mêmes paramètres que si vous aviez utilisé la fonction directement.

Vous gagnez ainsi en dynamisme. C'est ainsi que peuvent par exemple être gérés les plug-ins. Une couche de votre application se charge d'explorer les fonctions des différents plug-ins. Elle vous retourne différentes instances de la classe `ReflectionFunction`. Une seconde couche de votre application récupère ces instances et décide laquelle exécuter.

```
<?php
function aleas($min=0 , $max=15) {
    return mt_rand();
}
$fct = new ReflectionFunction( 'aleas' );
echo $fct->invoke(0, 1) ;
echo $fct->invoke() ;
```

À partir de PHP 5.1 il existe aussi une méthode `invokeArgs()` qui prend les différents arguments pour la fonction dans un tableau unique.

## Les objets, classes et interfaces

On instancie la classe `ReflectionClass` en passant au constructeur le nom de la classe à explorer en unique argument. L'objet renvoyé nous permettra d'obtenir des informations sur la classe en question de la même manière que celle que nous avons employée sur les fonctions. Les interfaces sont gérées comme des classes spéciales et peuvent donc être utilisées de la même manière.

On peut de la même manière obtenir la description d'une classe à partir d'un objet. Il faudra alors instancier `ReflectionObject` au lieu de `ReflectionClass`, le reste est équivalent.

### Similarités avec la gestion des fonctions

Les méthodes `getName()`, `getStartLine()`, `getEndLine()`, `getFileName()`, `isUserDefined()` et `isInternal()` fonctionnent de la même manière que pour la description des fonctions.

```
<?php
class maclasetest { /* ... */ }
$fct = new ReflectionClass( 'maclasetest' );
echo 'La classe se nomme : ' . $fct->getName() . "<br>\n";
echo 'Elle est définie dans le fichier '. $fct->getFileName();
echo ' entre les lignes ' . $fct->getStartLine();
echo ' et '. $fct->getEndLine() ;
```

### Définition de la classe

La méthode `getParentClass()` permet d'accéder au nom de la classe parent s'il y en a une. En soumettant aussi la classe parent à l'introspection PHP, vous pourrez déterminer toute

la hiérarchie des classes et leur rôle. Vous pouvez aussi tester si la classe explorée est ou non une classe dérivée d'une classe X donnée via la méthode `isSubClassOf()`.

Les méthodes `isAbstract()` et `isFinal()` retourneront chacune vrai ou faux selon que la classe explorée est ou non une classe abstraite ou une classe finale. La méthode `isInterface()` vous permettra de vérifier s'il s'agit d'une interface. Si c'est le cas, une valeur vraie vous sera retournée.

```
<?php
class maclasetest { /* .... */ }
class test { /* ... */ }
$fct = new ReflectionClass( 'maclasetest' );
if ($fct->isSubClassOf('test')) {
    echo "La classe maclasetest est dérivée de la classe test" ;
} elseif( ! $fct->getParentClass() ) {
    echo "La classe maclasetest n'a aucune classe parent" ;
}
```

Enfin, la méthode `getInterfaces()` permet de récupérer la liste des interfaces implémentées par leur nom. On peut aussi vérifier que la classe courante implémente bien une interface en passant le nom de l'interface à la méthode `implementsInterface()`. Cette dernière renverra une valeur vrai si l'interface est réellement implémentée et déclarée, faux dans le cas contraire.

### Instanciation de classe

On peut récupérer une nouvelle instance de classe via un appel à `newInstance()`.

Certains types de classe telles que les classes abstraites ou les interfaces ne sont pas instanciables. Par introspection, il est possible de connaître ce facteur via la méthode `isInstantiable()` qui renvoie une valeur vrai si on peut instancier la classe.

On peut aussi vérifier qu'un objet appartient bien à la classe explorée grâce à la méthode `isInstance()`. Elle renverra la valeur vrai si l'objet est bien de la classe étudiée, faux dans le cas contraire.

### Gestion des méthodes et des attributs

La liste des différentes méthodes accessibles pour la classe explorée peut être obtenue avec `getMethods()`.

```
<?php
class test {
    /* classe exemple */
}
$fct = new ReflectionClass( 'test' );
echo 'La classe se nomme : ' . $fct->getName() . "<br>\n";
echo 'Elle est définie dans le fichier ' . $fct->getFileName();
echo ' entre les lignes ' . $fct->getStartLine()
    . ' et ' . $fct->getEndLine();
```

On peut récupérer une méthode spécifique par son nom via la méthode `getMethod()`. Selon le cas, un objet ou une liste d'objets d'inspection de méthode seront retournés. Ces objets pourront ensuite être utilisés pour étudier les différentes méthodes.

Il est possible de récupérer un objet décrivant le constructeur avec la méthode `getConstructor()`. PHP récupère alors automatiquement le bon constructeur, que celui-ci utilise la convention de nommage PHP 4 ou la convention de nommage PHP5.

De la même manière, les méthodes `getProperty()` et `getProperties()` permettent d'accéder à un attribut de la classe ou à tous ses attributs. Les méthodes `getConstant()` et `getConstants()` permettent elles d'accéder aux constantes.

### Droit d'accès et déclaration

Les méthodes `isPrivate()`, `isProtected()` et `isPublic()` retournent chacune un booléen qui permet de savoir respectivement si la méthode est une méthode privée, protégée ou publique.

De même, `isStatic()` permet de vérifier si la méthode est déclarée comme statique, `isFinal()` et `isAbstract()` servent elles à voir si la méthode est finale ou abstraite.

Les méthodes `isConstructor()` et `isDestructor()` retournent vrai si la méthode est respectivement un constructeur de classe ou un destructeur, faux dans le cas contraire.

Enfin, la méthode `getDeclaringClass()` vous permet de récupérer le nom de la classe où a été déclarée la méthode. Les méthodes pour connaître le nom du fichier et les numéros de lignes contenant la déclaration sont les même que pour une fonction.

### Exécution de la méthode

Comme pour une fonction, la méthode `invoke()` permet d'exécuter la méthode. L'objet sur lequel sera exécutée la méthode doit être fourni en argument. Vous pouvez fournir la valeur `NULL` pour un appel statique.

## Les attributs

Les objets de description des attributs de classe peuvent être obtenus à partir de la classe elle-même et des méthodes `getProperty()` ou `getProperties()`.

La méthode `getName()` permet de connaître leur nom, les méthodes `isPrivate()`, `isProtected()`, `isPublic()`, `getDeclaringClass()` et `isStatic()` fonctionnent de manière similaire à la description des méthodes de classe.

La méthode `isDefault()` renvoie vrai si l'attribut est un attribut par défaut de la classe. Les méthodes `getValue()` et `setValue()` permettent de lire ou d'écrire une valeur dans l'attribut. Elles prennent comme premier argument une instance de classe sur laquelle faire la lecture/écriture (ou `NULL` pour un attribut statique). Enfin, la méthode `setValue()` prend une valeur à affecter en second argument.

# 13

## Gestion de fichiers

---

De nombreuses applications travaillent avec des fichiers. Que ce soit pour les lire, les remplir, les supprimer ou même changer leurs attributs, il existe un grand nombre de fonctions. Certaines vont de la plus simple et générique à des fonctions pointues et bas niveau vous permettant de gérer toute la procédure. Dans ce chapitre, nous allons voir dans un premier temps comment manipuler le contenu de fichiers (lecture et écriture), puis nous nous intéresserons aux informations les concernant et à leur manipulation sur le disque. À travers ces différentes étapes, nous détaillerons quelques fonctions spécifiques intéressantes, comme l'extraction automatique de fichiers de configuration ou de fichiers issus d'un tableur.

L'accès aux fichiers locaux est très rapide. Si vous avez peu de traitements et de tris à faire sur le contenu, il est généralement plus performant d'utiliser des fichiers qu'une base de données. C'est pourquoi nous vous conseillons de lire attentivement ce chapitre. Vous verrez par la suite qu'il est intimement lié au chapitre suivant, traitant de *socket* TCP/IP et de gestion de flux.

### Lecture et écriture

L'utilisation de fichiers n'est pas très compliquée. Nous commencerons par vous présenter les fonctions permettant de traiter facilement les principales actions sur des fichiers. Ensuite, nous travaillerons sur la méthode pas à pas pour vous permettre de bien comprendre la procédure dans son ensemble et donc de gérer des actions plus complexes.

#### Attention

Quand vous ouvrez un fichier PHP distant, n'oubliez pas qu'il sera exécuté et que donc vous n'afficherez pas son contenu mais le résultat de son exécution.

## Fonctions d'accès rapide

L'un des objectifs de PHP 5 est de simplifier le travail des développeurs en leur offrant des fonctions permettant de gérer simplement les principales manipulations de données. C'est le cas pour le traitement de fichiers, puisque deux fonctions ont vu le jour permettant de lire ou écrire entièrement un fichier : `file_get_contents()` et `file_put_contents()`. Nous verrons également des fonctions permettant de travailler sur les fichiers CSV (format de données pour les tableurs) et sur les fichiers de configuration (type `php.ini`).

### Lecture rapide

Il existe plusieurs fonctions permettant de lire tout un fichier en une passe. La plus simple est `file_get_contents()`. Elle prend en argument l'adresse du fichier et retourne une chaîne de caractères avec l'intégralité du contenu.

```
<?php
$contentu_fichier = file_get_contents('monfichier.txt');
echo $contentu_fichier;
?>
```

Si vous souhaitez afficher directement le contenu au lieu de le traiter comme une chaîne de caractères, vous pouvez utiliser la fonction `readfile()`. Elle est identique à la fonction `file_get_contents()`, mais envoie le contenu du fichier vers la sortie standard (généralement l'affichage) et retourne le nombre d'octets lus.

```
<?php
if (readfile('monfichier.txt')) {
    // Le fichier a été correctement affiché
} else {
    // Rien n'a été affiché
}
?>
```

Dans ces deux fonctions, vous pouvez spécifier un booléen comme second argument optionnel. S'il est évalué à `TRUE` et si l'adresse du fichier est une adresse relative, alors le fichier sera cherché à partir de tous les chemins fournis dans la directive de configuration `include_path`.

Il existe de plus une fonction similaire à `readfile()`, mais qui opère sur un fichier déjà ouvert avec `fopen()`. La fonction `fpasssthru()` lit l'intégralité du fichier à partir de la position actuelle et en envoie le contenu vers l'affichage. L'unique argument de cette fonction est le descripteur de fichier retourné par `fopen()` à l'ouverture (les détails concernant `fopen()` et les ouvertures de fichiers seront donnés plus loin dans ce chapitre).

```
<?php
$fp = fopen('monfichier.txt', "r") ;
if (fpasssthru($fp)) {
    // Fichier correctement affiché
} else {
    // Rien n'a été affiché
}
```

```
fclose($fp) ;  
?>
```

La fonction `file()`, quant à elle, prend en argument une adresse de fichier et retourne le contenu dans un tableau ligne à ligne. Les caractères de fin de ligne sont gardés et non éliminés dans l'opération. Tous les éléments du tableau, sauf éventuellement le dernier, contiennent donc un caractère de fin de ligne comme dernier caractère.

```
<?php  
$tab = file('monfichier.txt');  
for($i=0 ; $tab[$i] ; $i++){  
    echo $tab[$i].<br>';  
}  
?>
```

#### Remarque

Cette fonction peut se révéler assez gourmande en ressources si vous n'avez pas besoin d'utiliser le contenu ligne par ligne. Dans le cas où la séparation par ligne n'est pas votre but, il vaut probablement mieux utiliser `file_get_contents()`.

### Lecture des fichiers de configuration type

PHP vous offre un moyen simple d'utiliser des fichiers de configuration classique de syntaxe similaire au `php.ini` (les fichiers de configuration de Windows sont pour la plupart à ce format). Dans l'exemple suivant, nous vous présentons le début du fichier `mirirc.ini` :

```
[text]  
accept=*.bmp,*.gif,*.jpg,*.log,*.mid,*.png,*.txt,*.wav,*.zip  
ignore=*.exe,*.com,*.bat,*.dll,*.ini,*.vbs,*.js,*.htm,*.html  
network=Undernet  
commandchar=/  
linesep=-  
timestamp=[HH:nn]  
theme=mIRC Classic  
  
;Les commentaires commencent par un point virgule  
[dirs]  
logdir=logs\
```

La fonction `parse_ini_file()` prend en paramètre l'adresse d'un tel fichier et retourne son contenu sous forme d'un tableau associatif. Si vous fournissez `TRUE` comme deuxième paramètre (optionnel) vous obtiendrez un tableau à deux dimensions, la première étant le nom de la section (valeurs entre crochets dans le fichier de configuration) et la deuxième les associations clé/valeur contenues dans cette section. Le rendu de l'exemple suivant est illustré dans la figure 13-1.

```
<?php  
$tab = parse_ini_file("test.ini");
```



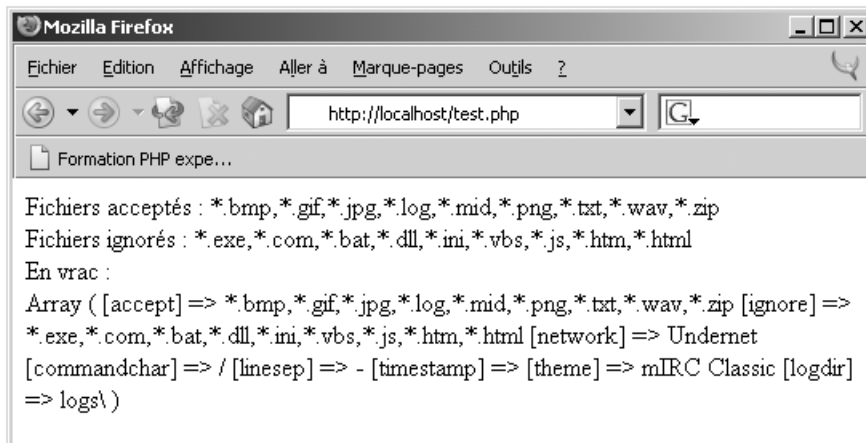
```

echo 'Fichiers acceptés : '.$tab['accept'].'<br>';
echo 'Fichiers ignorés : '.$tab['ignore'].'<br>';
echo 'En vrac :<br>';
print_r($tab);
?>

```

### Attention

La fonction `parse_ini_file()` est très stricte avec la syntaxe du fichier. Si elle retourne `FALSE`, c'est probablement parce que vous avez une erreur de syntaxe dans votre fichier de configuration, par exemple un caractère spécial dans une valeur non entourée de guillemets.



**Figure 13-1**  
*parse\_ini\_file*

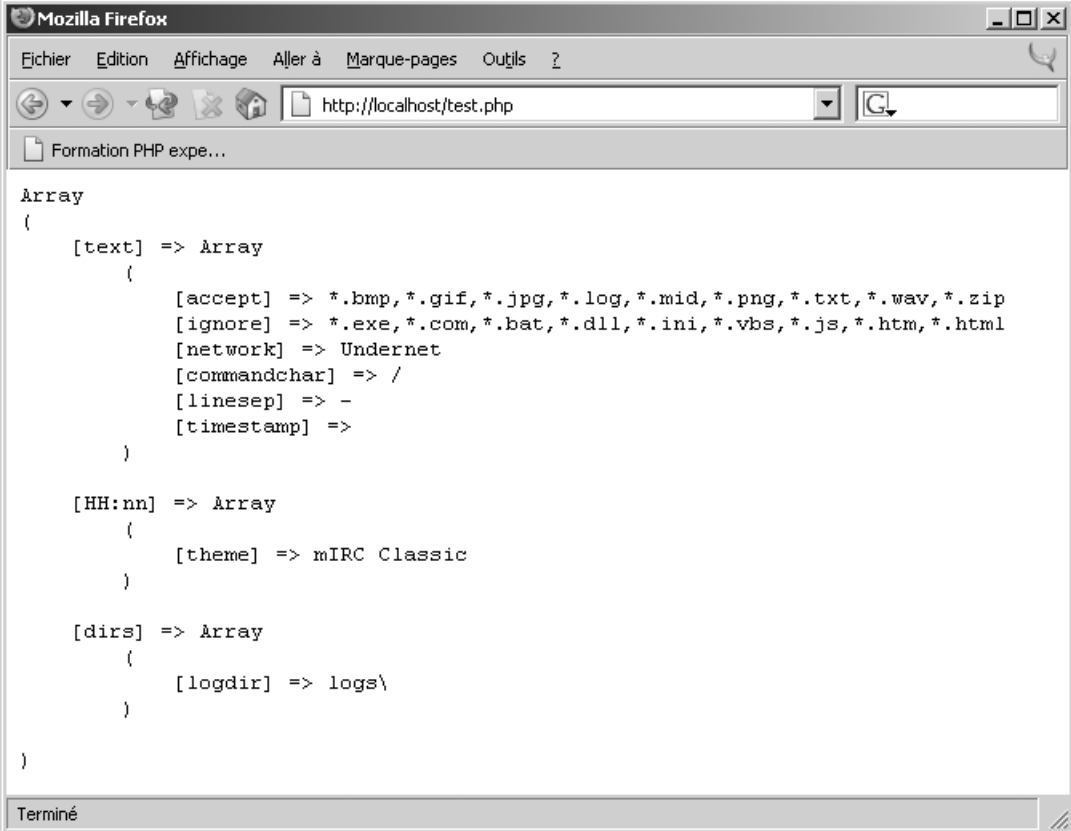
Lors de l'interprétation du fichier `ini`, PHP réutilise les constantes connues. Ainsi, si vous utilisez la chaîne `PHP_OS` (sans guillemets) comme valeur dans votre fichier `ini`, PHP la remplacera par la constante `PHP_OS` définie précédemment dans votre script.

La fonction `parse_ini_file()` peut prendre un second paramètre (optionnel) qui, s'il est évalué à `TRUE`, analyse également les sections (texte contenu entre crochet ouvrant et crochet fermant). Un exemple est donné à la figure 13-2.

```

<?php
$tab = parse_ini_file("test.ini",TRUE);
echo '<pre>' ;
print_r($tab);
echo '</pre>' ;
?>

```



```
Array
(
    [text] => Array
        (
            [accept] => *.bmp,*.gif,*.jpg,*.log,*.mid,*.png,*.txt,*.wav,*.zip
            [ignore] => *.exe,*.com,*.bat,*.dll,*.ini,*.vbs,*.js,*.htm,*.html
            [network] => Undernet
            [commandchar] => /
            [linesep] => -
            [timestamp] =>
        )
    [HH:nn] => Array
        (
            [theme] => mIRC Classic
        )
    [dirs] => Array
        (
            [logdir] => logs\
        )
)
```

Terminé

**Figure 13-2**  
*parse\_ini\_file, par sections*

### Exploiter un fichier de tableur (CSV)

Les fichiers CSV (*Comma Separated Values*) sont un format standard d'échange pour les tableurs. Il s'agit simplement d'un tableau à deux dimensions représenté dans un fichier. Habituellement, le séparateur de la première dimension (les lignes) est le caractère de fin de ligne et celui de deuxième dimension (les colonnes) est la virgule. Les chaînes de texte sont généralement délimitées par des guillemets afin de ne pas interférer avec les virgules. Les deux lignes qui suivent sont un exemple de ce qu'on peut trouver dans un fichier CSV :

```
"N","Titre","Nb pages","V 1","Relectures","Cas d'applications"
1,"Qu'est ce que PHP ?",28,1,1,"NA"
```

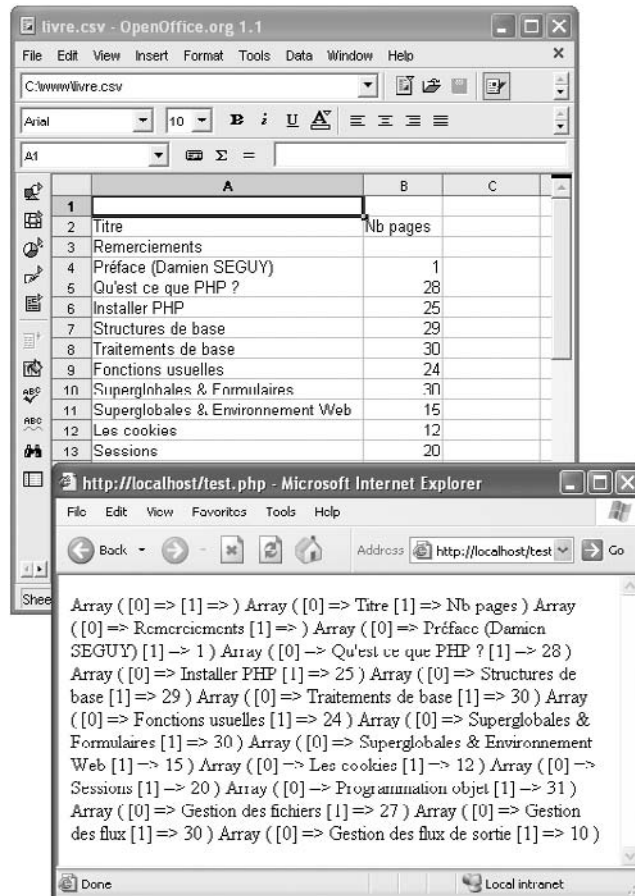
La fonction `fgetcsv()` permet de lire une ligne d'un fichier CSV précédemment ouvert avec `fopen()`. Elle renvoie un tableau indexé avec les différentes valeurs et fonctionne de manière similaire à `fgets()` (vous trouverez les descriptions de `fgets()` et `fopen()` plus

loin dans ce chapitre, avec les autres fonctions bas niveau). Le premier argument est le descripteur de fichier et le second est la taille maximale de la ligne. Il faut bien faire attention à ce que la taille précisée soit supérieure au nombre de caractères (fin de ligne compris) de la ligne la plus longue du fichier ; ce paramètre n'est pas optionnel.

Le résultat du code exemple suivant est illustré à la figure 13-3 :

```
<?php
$fp = fopen('livre.csv','a+');
while ($tab = fgetcsv($fp,1000)){
    print_r($tab);
}
?>
```

**Figure 13-3**  
*Interpréter un fichier  
de tableur*



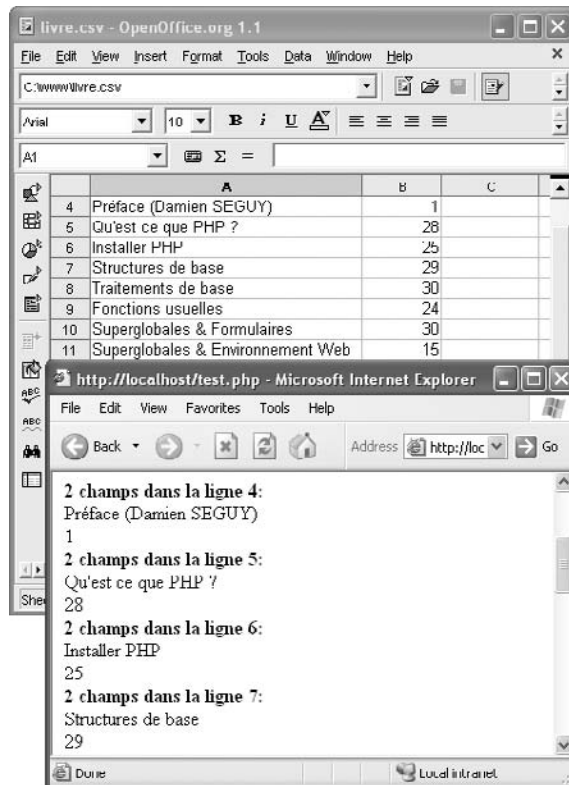
Deux paramètres optionnels sont disponibles : le premier vous permet de préciser un séparateur de deuxième dimension alternatif (la virgule est utilisée par défaut) et le

second un délimiteur pour les chaînes de texte (sinon, le guillemet est utilisé). Un exemple d'utilisation est donné dans le code suivant et à la figure 13-4 :

```
<?php
$row = 1;
$fp = fopen ("livre.csv","r");
while ($data = fgetcsv ($fp, 1000, ",")) {
    $num = count ($data);
    print "<b>$num champs dans la ligne $row: </b><br>\n";
    $row++;
    for ($c=0; $c < $num; $c++) {
        print $data[$c] . "<br>\n";
    }
}
fclose ($fp);
?>
```

**Figure 13-4**

*Interpréter et  
afficher un fichier  
de tableur*



#### Attention

Une ligne vide ne provoquera pas d'erreur, mais simplement une ligne contenant une seule valeur : NULL.

## Écriture rapide

Par rapport à la lecture, il y a moins de besoins différents pour écrire dans un fichier. Il en résulte qu'il n'existe qu'une seule fonction d'accès rapide pour écrire dans un fichier. La fonction `file_put_contents()` prend en paramètre une adresse de fichier et une chaîne de caractères. La chaîne est alors écrite dans le fichier. Si le fichier existait déjà, son contenu est écrasé.

```
<?php
$var = 'Ensemble de texte';
file_put_contents('monfichier.txt', "contenu du fichier $var");
?>
```

Il est toutefois possible de demander à ce que la chaîne en argument soit ajoutée au fichier au lieu de remplacer le contenu actuel. Il suffit alors de spécifier la constante `FILE_APPEND` comme troisième paramètre optionnel.

```
<?php
$fichier = 'monfichier.txt';
$contenu = 'Contenu du fichier';
file_put_contents($fichier, $contenu, FILE_APPEND);
?>
```

## Ouverture d'un fichier

Maintenant que nous avons vu les fonctions permettant d'accéder de façon simple et rapide aux fichiers, voyons comment effectuer un traitement de fichier de façon plus précise.

Avant d'utiliser un fichier, il faut l'ouvrir. Derrière ce terme se cache une définition simple : on demande au système d'exploitation de rechercher un fichier à partir de son adresse et de nous retourner un pointeur vers ce fichier. Ce pointeur est appelé descripteur de fichier (*file descriptor*).

La fonction `fopen()` permet de déclencher l'ouverture. Elle prend deux paramètres : le nom complet (avec l'adresse) du fichier, et un mode d'ouverture.

### Adresses de fichier et séparateurs de répertoire

Sous Windows, les adresses de fichiers contiennent des barres obliques inverses (caractère `\`). Comme nous l'avons vu dans le chapitre 3, il est nécessaire de protéger ces caractères en les doublant afin qu'ils ne soient pas interprétés par PHP. Sous Linux, le séparateur est une barre oblique (caractère `/`).

Aussi, pour unifier l'écriture de localisation des fichiers, PHP accepte l'utilisation de la barre oblique comme séparateur quel que soit le système d'exploitation. Nous vous conseillons d'ailleurs de préférer cette possibilité à l'écriture classique. Elle est plus simple à lire et à écrire (il n'y a pas à doubler les barres obliques inverses), mais elle est surtout portable entre les différents systèmes d'exploitation, évitant de faire deux versions.

Si l'accès au fichier est possible en lecture, alors la fonction renvoie un descripteur de fichier. Si ce n'est pas le cas, elle retourne la valeur `FALSE` et un message d'alerte. Il est

donc nécessaire de toujours s'assurer que le fichier existe et qu'il est accessible avant de l'utiliser.

```
<?php
$fp = fopen('monfichier.txt', 'rb')
    or die("fichier 'monfichier.txt' inaccessible") ;
// Ouvre le fichier en lecture et stocke le descripteur dans $fp
// Ou arrête l'exécution en cas d'erreur
```

Un troisième paramètre optionnel permet de définir où chercher le fichier. Ainsi, si vous y spécifiez la valeur `TRUE`, alors un fichier avec une adresse relative sera cherché à partir des différents répertoires de la directive `include_path` à la place du répertoire courant (voir le chapitre 2 à ce sujet).

#### Attention

Pensez que si vous n'incluez pas le répertoire courant (symbolisé par un point) dans la directive `include_path`, le fichier ne sera alors pas cherché dans ce répertoire, même s'il y est présent. Faites aussi attention à l'ordre d'apparition des répertoires. Le répertoire courant n'aura la priorité que s'il apparaît en premier.

### Emplacement du fichier

L'adresse est par défaut relative au répertoire en cours. Si vous ne spécifiez pas une adresse absolue (commençant par `/` sur un Unix ou l'identifiant de disque sur un Windows), le fichier sera cherché dans le répertoire actuel.

#### Utiliser des adresses absolues ou des adresses relatives

Utiliser une adresse absolue ou une adresse relative est une notion à prendre en compte lors de la conception de l'application. Une adresse absolue rendra complexe une migration sur une plate-forme différente ou dans un autre répertoire. Une adresse relative impose de toujours savoir quel est le répertoire courant (ce n'est pas toujours celui qui contient le script si vous faites des inclusions multiples).

Une ligne directrice pour savoir quoi utiliser consiste à se baser sur le fait que le fichier à ouvrir appartient à l'application ou au système. S'il appartient au système, alors il vaut probablement mieux utiliser une adresse absolue afin que son chemin ne dépende pas de l'emplacement de votre application. Inversement, si le fichier appartient à votre application, une adresse relative permettra de ne rien modifier si vous changez de répertoire tout votre applicatif et ses données.

Une manière d'éviter les problèmes est de définir en début de script une constante qui sera le préfixe à utiliser devant toutes vos adresses de fichiers. Laissez-la vide si vous voulez du local ou définissez-la si vous voulez une adresse absolue. Il vous sera alors facile de changer en une fois la référence de tous les fichiers.

Une autre astuce, si le problème est de connaître le répertoire en cours, est d'utiliser le répertoire dans lequel est votre script (ce répertoire est, lui, toujours le même, quel que soit le fichier qui a été appelé au départ par le serveur web). On exploite alors la constante `__FILE__`, qui définit l'adresse du script en cours (elle change quand on inclut un script, il ne s'agit donc pas d'une vraie constante) et la fonction `dirname()` qui donne un répertoire à partir d'une adresse. L'adresse du répertoire contenant le script actuel est donc `dirname(__FILE__)`.

## Fichiers distants

PHP offre une grande possibilité d'abstraction pour l'ouverture des fichiers. Si vous fournissez l'adresse d'une page web en HTTP ou celle d'un fichier en FTP, vous pourrez y accéder en lecture comme si vous étiez sur le système de fichier local (moyennant des temps d'accès plus longs). Tout ce que vous avez à faire est de préfixer l'adresse par `ftp://` ou `http://` comme si vous la tapiez dans la barre d'adresse de votre navigateur. Une description plus complète des possibilités d'abstraction sera présentée au chapitre suivant.

## Mode d'ouverture

Le mode d'ouverture définit si on veut accéder au fichier en lecture ou en écriture. Dans le cas de la lecture, il faut préciser "r" (read en anglais). Pour accéder au fichier en écriture, on note "w" (write en anglais). Le fichier est alors écrasé (remplacé) pour que l'utilisateur puisse en réécrire le contenu. Si le fichier n'existe pas, il est alors créé.

Pour accéder au fichier en écriture, mais en conservant le contenu, on utilise le mode d'accès "a" (append). Le contenu du fichier est sauvegardé et les écritures se font à la fin du fichier. Si le fichier n'existe pas, il est alors créé.

Si un + est ajouté après le mode d'accès alors le fichier sera accessible en écriture et en lecture (dans le cas d'un mode d'accès "r+", le pointeur sera positionné en début de fichier comme pour "w" mais le contenu ne sera pas tronqué à zéro.

**Tableau 13-1 Modes d'ouverture**

Mode	Signification
r	Ouvre en lecture seule, et place le pointeur au début du fichier.
r+	Ouvre en lecture et écriture, et place le pointeur au début du fichier.
w	Ouvre en écriture seule; place le pointeur au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
w+	Ouvre en lecture et écriture ; place le pointeur au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
a	Ouvre en écriture seule ; place le pointeur à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.
a+	Ouvre en lecture et écriture ; place le pointeur à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

Sous Windows, il est en outre demandé de faire la distinction entre un fichier binaire et un fichier texte, le système d'exploitation traitant de manière spécifique les fins de ligne lors de l'accès aux fichiers texte. Dans le cas d'un fichier binaire, il faut ajouter la lettre b (binary) juste après le r. Dans le cas d'un fichier texte, il faut ajouter t (text). Utiliser ces suffixes n'a aucune influence sur les autres systèmes d'exploitation. Il est donc recommandé de toujours utiliser le suffixe b quand on ouvre un fichier sous Unix, afin de garantir la portabilité éventuelle de l'application.

## Lecture d'un fichier

Maintenant que nous avons ouvert notre fichier, il nous reste à lire son contenu. Nous avons pour cet usage quatre fonctions de base.

### Lire caractère par caractère

La fonction `fgetc()` prend en paramètre le descripteur de fichier, lit un caractère et le renvoie comme valeur de retour. Lors du prochain appel, la fonction lira le caractère suivant, et ainsi de suite jusqu'à la fin du fichier. S'il n'y a plus de caractère à lire, alors la fonction renvoie la valeur `FALSE`.

```
<?php
$fp = fopen("monfichier.txt", "rb");
$caractere = fgetc( $fp );
if ( $caractere !== FALSE ) {
    echo "La première lettre est $caractere" ;
} else {
    echo "Le fichier ne contient aucun caractère" ;
}
?>
```

### Lire par ligne

La fonction `fgets()` retourne tous les caractères jusqu'à la prochaine fin de ligne (comprise). Elle permet donc de récupérer un fichier ligne par ligne. Toutefois, une telle lecture est coûteuse car on manipule alors des chaînes de caractères de grande taille. Si vous spécifiez une taille en octets comme deuxième paramètre, juste après le descripteur de fichier, `fgets()` retournera au maximum ce nombre de caractères, même si aucune fin de ligne n'a été trouvée. Il est fréquent de spécifier une taille arbitraire de 1 024 octets pour lire toute la ligne. Si la chaîne retournée ne se termine pas par une fin de ligne et contient moins de caractères que le maximum autorisé, c'est qu'on est arrivé en fin de fichier.

```
<?php
$fp = fopen("monfichier.txt", "rb") ;
$ligne = fgets( $fp , 1024 ) ;
$taille = strlen( $ligne ) ;
if ( $ligne[$taille - 1] != "\n" && $taille < 1024 ) {
    echo "Le fichier contient une ligne unique : $ligne" ;
} elseif( $ligne[$taille - 1] != "\n" ) {
    echo "La ligne est de plus de 1024 caractères : $ligne" ;
} else {
    echo "La première ligne est $ligne" ;
}
?>
```

#### Attention

Si la lecture s'arrête sur une fin de ligne, le caractère de fin de ligne est renvoyé dans la chaîne. S'il n'est pas souhaité, vous pouvez le supprimer avec la fonction `rtrim()` :

```
$chaine = rtrim(fgets($fp), "\r\n") ;
```



### Lire un fichier entier

La fonction `fread()` permet de lire plusieurs octets en une fois et retourne une chaîne de caractères. Le nombre d'octets à lire est à spécifier en second paramètre, après le descripteur de fichier. Si la chaîne retournée est plus courte que ce qui a été demandé, c'est que la fin du fichier a été atteinte.

```
<?php
$fp = fopen("monfichier.txt", "rb");
$s = fread( $fp , 42 ) ;
echo "Les 42 premiers caractères sont : $s <br>" ;
if ( strlen($s) < 42 )
    echo "Le fichier contient moins de 42 caractères" ;
?>
```

Il est donc possible de lire tout un fichier en une seule fois grâce à cette fonction. Il suffit de spécifier en deuxième paramètre la taille du fichier récupérée avec la fonction `filesize()`. Cette fonction prend en argument l'adresse du fichier et en retourne la taille en octets.

```
<?php
$fp = fopen("monfichier.txt", "rb");
$s = fread( $fp , filesize('monfichier.txt') ) ;
echo $s;
?>
```

Enfin, si vous souhaitez disposer d'une fonction vous permettant de récupérer des motifs complexes, vous pouvez utiliser `fscanf()`. Celle-ci est identique en tout point à la fonction `sscanf()` décrite dans le chapitre sur les traitements de chaînes, mis à part qu'elle prend en paramètre un descripteur de fichier à lire plutôt qu'une chaîne de caractères. Nous vous laissons donc vous reporter à ce chapitre pour en connaître les détails.

### Écriture dans un fichier

Il existe une seule fonction d'écriture simple : `fwrite()`. Elle prend en paramètre le descripteur de fichier et la chaîne à insérer dans le fichier.

```
<?php
// Ouvre le fichier
// Écrase le fichier actuel s'il existe ou le crée sinon
$fp = fopen("monfichier.txt", "wb");
// Insère le texte « PHP 5 » dans le fichier
fwrite( $fp , "PHP 5" ) ;
?>
```

Si une taille en octets est fournie comme troisième argument à `fwrite()`, alors elle spécifie la taille maximale de la chaîne à écrire. Par exemple, en spécifiant 42, seuls les 42 premiers caractères de la chaîne spécifiée seront écrits.

```
<?php
// Ouvre le fichier
```

```
$fp = fopen("monfichier.txt", "ab");
$var = 'Le PHP 5 nouveau est arrivé !';
// Insère le texte « Le PHP 5 » dans le fichier
fwrite( $fp , $var, 9);
// La fin du texte passé en argument n'est donc pas écrite
?>
```

**Note**

Vous verrez peut-être dans d'autres scripts l'utilisation de la fonction `fputs()`. Il s'agit d'un alias pour la fonction `fwrite()` ; il n'y a aucune différence à l'utilisation.

## Positions dans le fichier

Comme on l'a vu précédemment, le système garde en mémoire un pointeur de position dans le fichier. Il permet de lire ou d'écrire le fichier séquentiellement. Ce pointeur est automatiquement géré lors des lectures et écritures de façon à avancer à chaque opération.

Il est possible de définir manuellement une position. C'est utile par exemple pour revenir au début du fichier ou retenir une position pour y revenir par la suite. Trois fonctions permettent de traiter les opérations courantes sur ce pointeur de position.

### Placer le pointeur en début de fichier

La fonction `rewind()` permet de revenir en début de fichier, à l'octet zéro. Elle prend en paramètre un descripteur de fichier retourné par `fopen()`, renvoie `TRUE` en cas de réussite et `FALSE` en cas d'échec.

```
<?php
$fp = fopen("monfichier.txt", "rb");
// On affiche le fichier ligne par ligne
while($ligne = fgets( $fp , 1024 )){
    echo $ligne ;
}
// On se place au début :
rewind($fp);
// On réaffiche la première ligne
$ligne1 = fgets( $fp , 1024 );
echo $ligne1 ;
?>
```

### Placer le pointeur en un point du fichier

La fonction `fseek()` permet de positionner le pointeur de position à un certain endroit dans le fichier. Elle prend en paramètre le descripteur de fichier et un nombre d'octets.

Ainsi, `fseek($fp,42)` positionnera le pointeur après le quarante-deuxième octet du fichier décrit par `$fp`. Il est possible de passer un troisième argument optionnel. S'il est équivalent à la constante `SEEK_CUR`, le nombre d'octets est ajouté à la position actuelle au lieu de

prendre comme référence le début du fichier. S'il est à `SEEK_END`, la référence est la fin du fichier (vous voudrez donc probablement donner une taille en octets négative pour accéder à la position voulue). La valeur `SEEK_SET` est la valeur par défaut, elle prend pour référence le début du fichier.

```
<?php
// On ouvre le fichier
$fp = fopen("monfichier.txt", "rb") ;
// On se positionne au 42e octet
fseek( $fp, 42 );
echo fgetc($fp);
// On se positionne 10 octets plus loin, donc au 52
fseek( $fp, 10, SEEK_CUR );
echo fgetc($fp);
// Puis 20 octets avant la fin du fichier
fseek( $fp, -20, SEEK_END );
echo fgetc($fp);
// On revient au 42e octet à partir du début
fseek( $fp, 42, SEEK_SET );
echo fgetc($fp);
?>
```

La fonction `fseek()` est habituellement utilisée de concert avec `ftell()`, qui permet de connaître la position actuelle du pointeur à partir du descripteur de fichier. C'est par exemple utile pour retenir une position et y revenir par la suite ou construire un index du fichier qui évitera de le parcourir complètement la prochaine fois.

```
<?php
$fp = fopen("monfichier.txt", "rb");
/* On effectue différents traitements sur le fichier */
fseek( $fp, 42 );
// On récupère l'emplacement du pointeur :
$position = ftell($fp);
?>
```

#### Attention

Si vous avez ouvert le fichier en mode ajout (a), les données écrites seront toujours écrites à la fin, quelle que soit la position courante.

### Détection de fin de fichier

Nous avons vu comment détecter la fin de fichier avec les fonctions de lecture. Toutefois, ces méthodes ne sont pas adaptées à tous les cas. Par exemple, pour détecter la fin du fichier, nous sommes obligés de lire des caractères, lesquels ne seront plus lus par la suite (car le pointeur de position aura avancé). Afin d'éviter ce problème, vous pouvez tester si le pointeur de position est à la fin du fichier simplement avec la fonction `feof()`. Elle prend en argument le descripteur de fichier et renvoie `TRUE` si le pointeur est à la fin du fichier, `FALSE` sinon.

```
<?php
$fp = fopen('monfichier.txt', "rb") ;
while (!feof($fp)){
    $ligne = fgets( $fp , 1024 );
    echo $ligne;
}
?>
```

## Fermeture d'un fichier

Une fois que le fichier a été manipulé, il faut le fermer. Cette action permet au système de libérer les ressources associées au fichier, par exemple son emplacement sur le disque ou la position du descripteur à l'intérieur du fichier. Comme pour les bases de données, PHP ferme automatiquement à la fin du script les fichiers encore ouverts.

Le nombre de fichiers ouverts sur un système est toutefois limité. Afin de ne pas surcharger le système, il est important de toujours fermer les fichiers ouverts une fois qu'on a fini de les utiliser. Pour ce faire, il suffit d'utiliser la fonction `fclose()` en lui donnant le descripteur de fichier en argument.

```
<?php
// Ouverture du fichier
$fp = fopen("monfichier.txt", "rb");
// On lit le fichier
echo "première ligne du fichier : " ;
echo fgets( $fp , 1024 ) ;
// On ferme le fichier
fclose($fp) ;
?>
```

## Gestion du tampon

Lors des écritures dans les fichiers, les données écrites sont inscrites en mémoire plutôt que directement dans le fichier. Elles ne sont recopiées sur le disque que par paquets ou à la fermeture, afin de minimiser les appels système et les accès disque.

Il est pourtant possible de demander à PHP de forcer l'écriture de toutes les données se trouvant actuellement dans le tampon mémoire grâce à la fonction `fflush()`. Elle prend en unique argument le descripteur de fichier.

```
<?php
$fp = fopen($fichier, 'w') ;
fputs($fp, $texte) ;
// Si un autre script essaie de lire le fichier maintenant
// il ne lit pas forcément ce que nous venons d'ajouter
// PHP le garde peut-être en mémoire pour mutualiser les écritures
fflush($fp) ;
// Maintenant, on est sûr que le texte est réellement écrit
?>
```

Il est aussi possible de spécifier manuellement la taille maximale du tampon à utiliser. Il suffit pour cela de donner la taille en second argument à la fonction `set_file_buffer()`, juste après le descripteur de fichier. Vous devriez laisser faire PHP et votre système d'exploitation dans la plupart des cas, et donc ne pas modifier cette valeur.

## Accès concurrents

À part certains cas précis où un seul accès est fait sur un fichier (script d'administration), plusieurs scripts risquent d'utiliser un même fichier simultanément. Cela ne pose aucun problème tant que tous les accès sont en lecture. En revanche, si un des scripts accède au fichier en écriture pendant qu'un autre y accède en lecture, il est possible que la lecture se fasse mal, lisant un fichier incomplet ou corrompu. Si deux scripts accèdent au même fichier en écriture simultanément, les résultats peuvent être imprévisibles et il est probable que le contenu ne soit plus cohérent.

Pour éviter des accès concurrents, il est possible de verrouiller un fichier quand on l'utilise. Un verrou peut être soit partagé (pour faire des lectures, autorisant plusieurs accès partagés simultanément), soit exclusif (pour écrire, aucun autre accès n'est alors autorisé en même temps). Pour poser ce type de verrous, on utilise la fonction `flock()`. Elle prend en paramètres le descripteur de fichier et soit `LOCK_SH` (pour un verrou partagé), soit `LOCK_EX` (pour un verrou exclusif).

On utilise généralement un verrou exclusif pour une écriture (un seul script doit écrire dans le fichier à un instant T) et un verrou partagé pour les lectures.

Une fois le verrou demandé, le script sera mis en attente jusqu'à qu'il soit obtenu. Si vous ne voulez pas être bloqué, vous pouvez donner la valeur `TRUE` dans un troisième argument optionnel. Dans ce cas, la fonction renverra `FALSE` si elle n'a pas réussi à obtenir le verrou (et si donc vous ne pouvez pas utiliser le fichier).

Pour relâcher le verrou, il faut refaire un appel à `flock()` avec la valeur `LOCK_UN`. Tant que vous ne faites pas cet appel et que le processus est en exécution, le verrou reste actif et peut bloquer les accès aux autres scripts. Pensez donc bien à relâcher le verrou dès que vous avez fini vos manipulations.

```
<?php
$fp = fopen('monfichier.txt', 'r' );
flock( $fp, LOCK_SH );
/* utilisation du fichier en lecture */
flock( $fp, LOCK_UN );
fclose( $fp );
?>
```

### Important

Pensez qu'une fois le verrou relâché, vous n'avez plus de contrôle d'accès. Il vous faut donc vider le tampon avec la fonction `fflush()` avant de retirer le verrou, et tout de suite fermer le fichier après l'avoir fait. Sans cette manipulation, PHP pourrait garder en mémoire certaines écritures. Si elles sont écrites dans le fichier après que le verrou a été relâché, les résultats peuvent être aléatoires.

## Troncature à l'ouverture

Le verrou s'obtient à partir du descripteur de fichier. Il faut donc déjà avoir ouvert le fichier avec `fopen()` avant de savoir s'il est disponible avec `flock()`. Si vous ouvrez avec le mode d'accès `w`, le fichier est tronqué à l'ouverture, avant d'obtenir le verrou et potentiellement pendant qu'un autre script le lit ou l'écrit. Vous risquez ainsi de lire un fichier à moitié (s'il est tronqué avant la fin de la lecture) ou de perdre des données (s'il est tronqué au milieu d'une écriture).

Si vous comptez tronquer le fichier à l'ouverture, il vous faudra alors utiliser un autre fichier comme contrôle d'accès. Vous aurez alors à obtenir le verrou sur ce fichier de contrôle et ne devrez ouvrir le fichier voulu qu'une fois ce verrou obtenu. Dès que vous aurez fini avec le fichier destination, vous pourrez retirer le verrou du fichier de contrôle. Grâce à ce stratagème, vous n'essaieriez jamais d'ouvrir le fichier destination tant que vous n'avez pas le verrou. Vous ne subirez jamais le problème de troncature (puisque si vous tronquez, c'est que vous êtes seul à accéder au fichier).

```
<?php
// On obtient le verrou sur le fichier de contrôle
$controle = fopen('monfichier_controle.txt', 'r' );
flock( $controle, LOCK_EX );

// On écrit le fichier réel
$ecriture = fopen('monfichier.txt', 'w' );
/* utilisation du fichier en écriture */
fclose( $ecriture );

// On relâche le verrou
flock( $controle, LOCK_UN );
fclose( $controle );
?>
```

Une autre solution est d'ouvrir le fichier en lecture-écriture avec `r+` et de ne tronquer le fichier qu'une fois le verrou obtenu. Pour cela, vous pouvez utiliser la fonction `ftruncate()`. Elle prend en arguments le descripteur de fichier et une taille à laquelle tronquer ; la valeur 0 aura le même effet qu'une ouverture avec le mode `w`.

## Limitations du système de verrou

Le système de verrou repose sur des fonctionnalités du système d'exploitation et du système de fichiers. Il est inaccessible sous Windows avec un système de fichiers de type FAT (donc inutilisable avec, entre autres, les systèmes d'exploitation Windows 95, 98 et Me). Cela n'est cependant pas problématique en général, car ces systèmes d'exploitation sont dédiés à une utilisation limitée à la bureautique et éventuellement au développement mais ne doivent en aucun cas servir de serveur en production.

De plus, le verrou fonctionne au niveau processus. Quand un verrou est posé, c'est tout le processus qui est habilité (ou non) à utiliser le fichier. Cette procédure rend donc

impossible d'utiliser ces verrous sur un système de processus légers (*threads*), comme c'est le cas sur le serveur web IIS de Microsoft ou par défaut sur la version Windows de Apache 2.

### Alternative sur système Unix

Sous Unix, une alternative à l'utilisation des verrous est possible. Vous pouvez obtenir une modification d'un fichier de manière atomique (soit rien n'est écrit, soit tout est écrit complètement, le fichier n'est jamais à un état intermédiaire) en écrivant dans un fichier temporaire et en le copiant vers la destination voulue à l'aide de la fonction `rename()`. Cette fonction prend en arguments l'adresse du fichier source (notre fichier temporaire) et l'adresse de destination (le fichier qu'on veut écrire).

```
<?php
// Adresse d'un fichier temporaire
$nomtemporaire = tempnam('/tmp', '');
// La fonction tempnam() crée un fichier avec un nom unique

// Écriture du fichier
$fp = fopen($nomtemporaire, 'w');
/* fermeture du fichier */
fclose($fp);

// Déplacement à l'adresse souhaitée
rename($nomtemporaire, 'adresse réelle du fichier');
?>
```

L'utilisation de ce procédé ne compromet pas les accès concurrents, car tant qu'il est utilisé, l'ancien fichier n'est pas effacé ; il est simplement déconnecté et l'adresse renvoie vers notre nouveau fichier. Les scripts qui ont ouvert le fichier avant le déplacement continueront à agir sur l'ancien fichier comme si de rien n'était et ne bloqueront pas l'écriture.

#### Note

Le défaut de cette méthode est que la date de création et l'*inode* (sorte d'identifiant unique) du fichier seront changés dans l'opération.

## Manipulation de fichiers

### *Copie et déplacement*

Pour copier ou déplacer un fichier, vous pouvez utiliser les fonctions `copy()` (copier) et `rename()` (déplacer). Pour les deux fonctions, le premier argument est l'adresse source et le second argument est l'adresse destination.

```
copy("fichier source", "fichier destination");
```

**Note**

Malgré son nom, la fonction `rename()` permet réellement de déplacer des fichiers entre des répertoires, même présents sur des systèmes de fichiers différents.

## Création et effacement

### Création de fichier

Pour créer un fichier, il existe deux méthodes. La première est d'ouvrir le fichier en écriture avec `fopen()` et de le fermer après (éventuellement en le remplissant entre-temps). La seconde option est de fournir l'adresse à la fonction `touch()`. Elle créera le fichier s'il n'existe pas ou mettra à jour ses dates d'accès et de modification s'il existe.

```
<?php
$fichier = 'fantome.txt';
if ( ! file_exists($fichier) ) {
    touch( $fichier );
}
?>
```

Il est également possible d'utiliser cette fonction pour changer la date de modification d'un fichier. Pour cela, il faut spécifier la date souhaitée sous forme d'un *timestamp* dans le second argument.

```
<?php
/*Ici je décide de la date de dernière modification que je veux donner à mon fichier */
$time = mktime(0,0,0,10,10,2003);

$fichier = 'fantome.txt';
touch( $fichier,$time);
?>
```

**Note**

Seul le propriétaire d'un fichier ou le super-utilisateur pourra mettre une date arbitraire avec le second argument.

### Effacement de fichier

Dans les systèmes Unix, on différencie l'adresse d'un fichier et son contenu. En spécifiant une adresse, on dit qu'on lie le contenu à un répertoire sous un certain nom. Un même contenu peut avoir plusieurs adresses (plusieurs liens physiques); le contenu n'est effacé que quand le dernier lien est supprimé et que plus aucun processus n'y accède.



La fonction d'effacement de PHP hérite de ce nom, même si elle fonctionne aussi sous Windows. Vous pouvez effacer une adresse (et donc le fichier s'il n'est plus référencé à aucune autre adresse) en la fournissant en argument à la fonction `unlink()`.

```
if ( file_exists($fichier) ) {  
    unlink( $fichier );  
}
```

## Liens

Un lien symbolique est un élément que l'on rencontre sur la plupart des systèmes de fichiers Unix. Il existe deux sortes de liens : les liens physiques (*hard link*) et les liens symboliques (*symbolic link* ou *soft link*).

### Liens physiques

Les liens physiques servent à référencer un même fichier à deux adresses différentes sur un disque. Vous pouvez créer un tel lien en appelant la fonction `link()` avec, comme premier argument, le fichier source à utiliser et, comme second argument, l'adresse à laquelle faire le lien. Par la suite, la source comme la destination seront considérées comme un même fichier et auront des propriétés identiques. En revanche, l'effacement d'une adresse n'effacera pas l'autre ; elle supprimera simplement une des deux références.

#### Note

Il n'est pas possible de faire un lien physique sur un répertoire.

### Liens symboliques

Le second type de lien est le lien symbolique. Il s'agit d'une sorte de raccourci. Contrairement au lien physique, les deux adresses de fichier ne sont pas identiques : l'une correspond réellement au fichier et l'autre restera toujours un raccourci. Pour créer un lien symbolique, utilisez la fonction `symlink()` au lieu de `link()`. La plupart des fonctions de traitement de fichier détectent les liens symboliques et font leur opération sur la cible, pas sur le lien lui-même.

### Fichiers temporaires

Il est souvent utile d'utiliser des fichiers temporaires, et dans ce cas d'obtenir une adresse unique où écrire. Pour obtenir une telle adresse temporaire, PHP met à disposition la fonction `tmpfile()`. Utilisée sans argument, elle retourne un descripteur de fichier comme si vous aviez utilisé `fopen()`. Le fichier est placé dans le répertoire temporaire du système (`/tmp` sur un Unix) et est automatiquement effacé à la fermeture du script.

Si vous souhaitez pouvoir manipuler le fichier sans qu'il ne soit effacé (par exemple pour le copier ou le déplacer ailleurs par la suite), vous pouvez utiliser la fonction `tempnam()` en fournissant en arguments une adresse de répertoire et un préfixe de nom de fichier. Un fichier au nom aléatoire préfixé par le deuxième argument sera créé dans le répertoire

spécifié et son adresse complète vous sera renvoyée (ou FALSE en cas d'erreur). Si le répertoire n'existe pas, le répertoire temporaire du système sera utilisé. Contrairement à `tmpfile()`, le fichier temporaire n'est pas effacé à la fin du script. Il vous appartient de le faire manuellement.

```
<?php
// On crée un fichier temporaire
$nomtemporaire = tempnam ("/tmp", "F00");
//On l'ouvre pour s'en servir
$fd = fopen($nomtemporaire, "w");
//On écrit dedans
fwrite($fd, "Ecrire dans le fichier temporaire");
//On le ferme
fclose($fd);

// Ici on pourrait l'utiliser, le copier, ...
unlink($nomtemporaire);
?>
```

## Gestion des répertoires

Un répertoire est en fait un fichier classique avec un attribut spécial. Il est donc manipulable comme un fichier avec la plupart des fonctions, même s'il mérite une attention particulière du fait que d'autres fichiers dépendent de lui.

### *Parcourir un répertoire*

Une des manipulations les plus fréquemment faites sur les systèmes de fichiers est le parcours d'un répertoire pour en lister les fichiers. Il existe deux méthodes pour faire ce parcours avec PHP 5. La première est une méthode objet et la seconde repose sur une série de fonctions.

#### Méthode objet

La procédure de parcours d'un répertoire est similaire à la procédure de lecture d'un fichier :

- récupérer un descripteur pointant vers le répertoire ;
- lire le contenu ;
- fermer le descripteur.

Pour initialiser la lecture du répertoire, vous devez faire appel à la fonction `dir()` avec une adresse de répertoire comme argument. Un objet répertoire vous est renvoyé.

```
<?php
// Script récupérant tous les fichiers d'un répertoire
// Dans un tableau $fichiers
$dir = dir('.') ;
```

Ensuite, vous pouvez lire séquentiellement tout le contenu du répertoire (fichiers, liens ou autres répertoires) grâce à la méthode `read()`. Chaque appel renvoie un élément. Quand tous les éléments ont été retournés, la fonction renvoie `FALSE`.

```
$fichiers = array() ;  
while( $nom = $dir->read() ) $fichiers[] = $nom ;
```

#### Attention

La fonction de lecture vous renverra les entrées `.` et `..` représentant le répertoire courant et le répertoire parent. C'est à vous de les filtrer et de les ignorer si vous ne les voulez pas.

Vous pouvez remettre à zéro la lecture séquentielle en appelant la méthode `rewind()`.

Enfin, vous devez libérer la ressource en appelant la méthode `close()`. Cette fermeture est faite automatiquement à la fin de l'exécution si vous ne le faites pas. Cependant, comme pour les fichiers et pour les mêmes raisons, nous vous conseillons de fermer les répertoires ouverts dès que vous ne les utilisez plus.

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

Si vous souhaitez afficher tous les fichiers du répertoire, il vous suffit d'exécuter le script suivant :

```
<?php  
// Script affichant tous les fichiers d'un répertoire  
$dir = dir('.') ;  
while( $nom = $dir->read() ) {  
    echo $nom, '<br>';  
}  
$dir->close() ;  
?>
```

#### Méthode procédurale

Vous pouvez utiliser les fonctions `opendir()`, `readdir()`, `rewinddir()` et `closedir()`, à la place de la méthode objet. Leur fonctionnement est identique aux méthodes décrites plus haut, si ce n'est que l'identifiant retourné par la première fonction devra être fourni en paramètre aux trois autres.

```
<?php  
// Script récupérant tous les fichiers d'un répertoire  
// dans un tableau $fichiers  
$dir = opendir(".") ;  
$fichiers = array() ;  
while( $nom = readdir($dir) ) {  
    $fichiers[] = $nom ;  
    echo $nom.'<br>';  
}  
}
```

```
closedir($dir) ;  
?>
```

### Filtre et listage rapide

Une autre méthode permet de récupérer facilement tous les fichiers d'un répertoire correspondant à un masque précis. Ainsi, la commande suivante fait exactement la même chose que le script précédent avec `opendir()` et `readdir()` :

```
<?php  
$fichiers = glob('./*') ;  
print_r($fichiers);  
?>
```

La fonction `glob()` prend en paramètre un masque et renvoie une liste de fichiers répondant à ce masque. Les caractères utilisables dépendent de votre système d'exploitation, mais généralement, l'astérisque peut remplacer un nombre quelconque de caractères sauf un séparateur de répertoires, et le point d'interrogation remplace un et un seul caractère. Ainsi, `*.html` désigne l'ensemble des fichiers HTML du répertoire courant.

```
<?php  
// Ici on va sélectionner tous les fichiers .jpg  
$fichiers = glob('./*.jpg') ;  
print_r($fichiers);  
?>
```

#### Note

Contrairement aux autres fonctions de lecture de répertoire, `glob()` ne retourne pas les fichiers cachés. Sous Unix les entrées `.` et `..` ne seront donc pas lues.

## Position dans l'arborescence

Chaque processus a, pour le système d'exploitation, une information nommée répertoire courant (ou répertoire de travail). C'est ce répertoire qui est utilisé comme référence dans les adresses relatives, que ce soit pour les ouvertures de fichier ou les inclusions (par exemple `include 'test.php';`).

Vous pouvez connaître le répertoire courant avec la fonction `getcwd()`.

```
<?php  
echo 'répertoire courant : ' , getcwd() ;  
?>
```

#### Attention

Le répertoire courant n'est pas forcément celui qui contient le script en cours. Cela est particulièrement vrai si le script en cours a été inclus par un autre d'un répertoire différent. Pour connaître le répertoire du script en cours, vous pouvez utiliser la syntaxe `dirname(__FILE__)`.

Vous pouvez modifier ce répertoire courant en fournissant une nouvelle adresse à la fonction `chdir()`. La nouvelle adresse servira de référence pour les futurs appels au système de fichiers. Cette fonction retourne `TRUE` en cas de réussite et `FALSE` en cas d'erreur.

```
<?php
echo 'répertoire courant : ' , getcwd().'\n';
//On descend d'un niveau
chdir ('..');
echo 'répertoire courant : ' , getcwd();
?>
```

## Créations et effacements

Il existe des fonctions spécifiques pour créer et supprimer les répertoires : `mkdir()` permet la création et `rmdir()` l'effacement. Elles prennent toutes les deux comme premier paramètre l'adresse du répertoire à créer (ou effacer). La fonction `mkdir()` prend optionnellement un paramètre supplémentaire définissant les droits d'accès (sous forme numérique) du répertoire à créer, la valeur `0777` étant la valeur par défaut. Cette valeur passe ensuite par le masque par défaut (voir plus loin dans ce chapitre pour la gestion des droits d'accès) ; pensez donc à mettre un masque nul si vous souhaitez effectivement pouvoir donner tous les droits à tout le monde sur le répertoire.

### Attention

N'oubliez pas de préfixer les chiffres droits d'accès par un 0 afin de signaler que vous utilisez la notation octale, ce qui est généralement le cas.

## Informations sur les fichiers

Maintenant que nous savons lire et écrire dans les fichiers, il devient nécessaire de pouvoir en connaître différentes informations et attributs. On peut citer par exemple les droits d'accès du fichier afin de le partager ou au contraire de le restreindre, les dates de création et modification pour vérifier si un fichier a été modifié, etc.

Toutes les statistiques sur un fichier sont récupérables à l'aide d'appels système, mais comme ces appels se révèlent coûteux en performance, PHP place les résultats dans un cache. Il en résulte que si vous testez la date de modification avant et après avoir modifié un fichier, elle pourrait sembler n'avoir pas changé. C'est en fait que PHP se sert de la valeur qu'il a récupérée précédemment.

Pour vider le cache des informations sur l'état des fichiers, il vous suffit d'appeler la fonction `clearstatcache()`, sans argument.

### Attention

Il n'est généralement pas possible d'obtenir des informations sur des fichiers distants (accès par HTTP par exemple).

## Existence d'un fichier

La plupart des fonctions de traitement de fichier considèrent comme une erreur de recevoir comme paramètre une adresse de fichier qui n'existe pas ; elles provoquent donc une alerte. Si, dans le fonctionnement normal de votre application, vous pouvez être amené à avoir des fichiers absents, il vous faut pouvoir tester l'existence d'un fichier.

Pour cela, la fonction `file_exists()` prend en paramètre une adresse de fichier et renvoie `TRUE` si le fichier existe, `FALSE` sinon. Il faut toutefois prendre en compte que la notion de fichier est prise au sens large : un répertoire ou un lien (même menant à une destination inexistante) seront considérés comme des fichiers.

```
<?php
if (file_exists('monfichier.txt')) {
    echo " fichier existe " ;
} else {
    echo " rien n'a été affiché " ;
}
?>
```

## Dates de fichiers

Les dates sont parmi les informations les plus collectées sur un fichier. Il existe habituellement trois dates concernant un fichier : la date de création, la date de modification (dernier accès en écriture) et la date de dernier accès (lecture, écriture ou exécution).

Vous pouvez obtenir la date de création avec la fonction `filectime()`, la date de dernière modification avec `filemtime()` et la date de dernier accès avec `fileatime()`. Ces trois fonctions prennent en paramètre l'adresse du fichier et retournent une date sous forme de *timestamp* Unix (pour plus d'informations sur le traitement des dates, vous pouvez vous reporter au chapitre 7).

```
<?php
if (filemtime('fichier.txt') > filectime('fichier.txt')) {
    echo 'le fichier a été modifié depuis sa création';
} else {
    echo 'le fichier n'a pas été modifié';
}
?>
```

### Note

Pour des raisons de performances, la traque de la date de dernier accès est souvent désactivée sur les systèmes de fichiers des serveurs web. Le résultat obtenu sera alors indéterminé.

## Taille de fichier

Vous pouvez connaître la taille d'un fichier en utilisant la fonction `filesize()` avec l'adresse du fichier en argument. La taille sera retournée en octets.

```
<?php
$fichier = 'monfichier.txt';
echo $fichier , ' fait ' , filesize($fichier) , ' octets';
?>
```

L'identifiant du fichier sur le disque (appelé *inode*) peut être récupéré par la fonction `fileinode()`.

Ces deux informations, ainsi que la plupart des précédentes, peuvent être récupérées en une fois dans un tableau associatif grâce à la fonction `stat()`, qui prend en argument une adresse de fichier. Entre autres, dans ce tableau, l'indice `ino` correspond à l'*inode*, l'indice `size` correspond à la taille, l'indice `perm` correspond aux permissions du fichier, les indices `atime`, `mtime` et `ctime` correspondent aux dates de dernier accès, modification et création, `gid` et `uid` sont les identifiants numériques du groupe et de l'utilisateur propriétaire, et l'indice `nlink` donne le nombre de références (nombre de liens) vers le fichier.

```
<?php
$fichier = 'monfichier.txt';
$tab = stat($fichier);
echo $tab['ino'].'<br>';
echo $tab['atime'].'<br>';
echo $tab['ctime'].'<br>';
echo $tab['gid'].'<br>';
?>
```

La fonction `fstat()` est similaire à `stat()` mais prend en paramètre un descripteur de fichier au lieu d'une adresse.

```
<?php
$fichier = fopen('monfichier.txt','r+');
$tab = fstat($fichier);
echo $tab['ino'].'<br>';
echo $tab['atime'].'<br>';
echo $tab['ctime'].'<br>';
echo $tab['gid'].'<br>';
?>
```

## Espace disque disponible

Il est possible de vérifier l'espace disponible sur un système de fichiers en spécifiant une adresse de fichier ou de répertoire de cette partition à la fonction `disk_free_space()`. La fonction `disk_total_space()` fonctionne de la même façon, mais donne la taille totale de la partition.

```
< ?php
$repertoire = '/tmp' ;
```

```
$place = disk_free_space($repertoire) ;  
echo "il reste $place octets de libre";  
?>
```

## Nom et adresse d'un fichier

La fonction `realpath()` permet de connaître le chemin réel d'un fichier sous sa forme canonique. Une adresse relative sera transformée en adresse absolue, les liens symboliques seront résolus et les références telles que `.`, `../` ou les `/` surnuméraires seront transformés. Cette fonction est utile principalement dans deux cas : pour transformer une adresse relative en adresse absolue, et pour vérifier l'adresse réelle à partir d'une adresse donnée par un visiteur. On évite ainsi de se faire leurrer par des liens symboliques ou des adresses masquées par des références relatives. Dans l'exemple suivant, on peut voir une façon d'éviter ce genre de piège :

```
<?php  
$chemin = '/tmp/../etc/password' ;  
$chemin_reel = realpath($chemin) ;  
echo $chemin_reel; // Affiche /etc/password  
?>
```

Une adresse de fichier contient trois parties : l'adresse du répertoire source, le nom de fichier et l'extension. À partir d'une adresse de fichier, la fonction `dirname()` permet de récupérer l'adresse du répertoire source et la fonction `basename()` retournera l'ensemble nom de fichier et extension. Les deux fonctions prennent en argument l'adresse à traiter. Attention toutefois à `dirname()`, qui ne sait traiter correctement qu'une adresse absolue (vous aurez donc peut-être à utiliser `realpath()` avant).

La fonction `pathinfo()` permet de récupérer les trois parties d'une adresse en une étape. En fournissant en argument l'adresse à traiter, la fonction vous retournera un tableau associatif. Dans ce tableau, les répertoire, nom et extension sont référencés par les indices `dirname`, `basename` et `extension`. Les deux premiers indices donneront les mêmes résultats que les fonctions respectives `dirname()` et `basename()`.

## Nature des fichiers

Sur un système de fichiers, il est possible de manipuler plusieurs éléments : des répertoires, des liens et enfin des fichiers. D'autres types, comme les *sockets* Unix ou les fichiers périphériques, peuvent aussi être rencontrés, mais il est peu probable que vous ayez à vous en servir.

Il est possible de déterminer la nature d'une adresse avec trois fonctions : `is_dir()`, `is_link()` et `is_file()` renvoient `TRUE` si l'adresse fournie en argument est respectivement un répertoire, un lien ou un fichier, et `FALSE` sinon.

La fonction `filetype()` renvoie, elle, le type du fichier à partir de son adresse. La chaîne renvoyée contient alors une des valeurs suivantes : `fifo`, `char`, `dir`, `block`, `link`, `file` ou `unknown`.



## Liens symboliques

Quand vous manipulez l'adresse d'un lien symbolique, les fonctions lisent pour la plupart le contenu et les propriétés de la cible et non du lien lui-même.

Vous pouvez toutefois récupérer l'adresse de la cible en lisant le raccourci avec la fonction `readlink()`, qui prend l'adresse du raccourci en argument. Il est aussi possible de vérifier que la cible existe et est valide avec la fonction `linkinfo()`, qui renvoie une valeur nulle si la cible n'existe pas (ou est invalide).

Si vous souhaitez récupérer explicitement les informations sur un lien symbolique et non sur sa cible, vous pouvez utiliser la fonction `lstat()` au lieu de `stat()`.

Pour plus d'explications sur les liens, vous pouvez vous référer à la section « Liens » dans ce chapitre.

## Permissions et droits d'accès

Sur la majorité des systèmes d'exploitation, un fichier se voit associer un jeu de droits d'accès et de permissions. Un fichier a donc un propriétaire (utilisateur et groupe), des droits d'exécution, de lecture et de modification (ou écriture).

Les fonctions `fileowner()` et `filegroup()` permettent de récupérer le propriétaire (respectivement utilisateur et groupe) à partir d'une adresse de fichier. La valeur de retour est un identifiant numérique.

```
<?php
$owner = fileowner('test.ini');
echo "l'uid du propriétaire est $owner";
?>
```

### Note

Si vous voulez récupérer le nom à partir de l'identifiant, il vous faudra fournir ces identifiants aux fonctions `posix_getpwuid()` (dans le cas de l'utilisateur) et `posix_getgrgid()` (dans le cas du groupe).

La fonction `fileperms()` retourne les permissions Unix associées au fichier spécifié. Dans une représentation octale, de droite à gauche, le premier chiffre correspond aux permissions pour tout le monde, le second aux permissions pour le groupe propriétaire et le troisième aux permissions pour l'utilisateur propriétaire.

### Attention

Faites attention aux diverses représentations de l'entier représentant les permissions. Les permissions classiques 0777 sont en fait données en base octale; en base décimale, on obtient 511 ( $511 = 7 \cdot (8^2) + 7 \cdot (8^1) + 7$ ). Le chiffre retourné n'est dans aucune base; si vous l'affichez, il sera par défaut affiché en décimal. Pensez donc bien à utiliser le modulo pour faire vos opérations et pas de simples divisions par dix.

Afin de faciliter les vérifications, il est possible de savoir directement si un fichier nous est accessible (avec les droits du script en cours d'exécution, qui sont probablement ceux du serveur web). La fonction `is_readable()` renvoie `TRUE` si le fichier est accessible en lecture et `FALSE` sinon. La fonction `is_writable()` fait de même pour les droits d'écriture et la fonction `is_executable()` pour les droits d'exécution.

Le rendu du code suivant est illustré à la figure 13-5 :

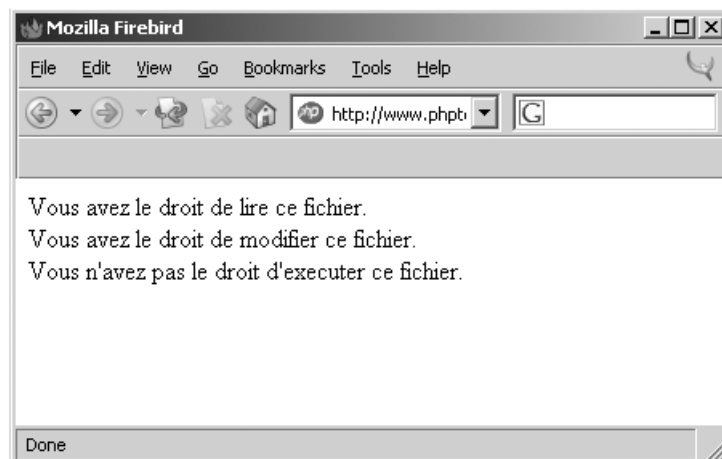
```
<?php
if (is_readable('test.ini')){
    echo 'Vous avez le droit de lire ce fichier.<br>';
}else{
    echo 'Vous n\'avez pas le droit de lire ce fichier.<br>';
}

if(is_writable('test.ini')){
    echo 'Vous avez le droit de modifier ce fichier.<br>';
}else{
    echo 'Vous n\'avez pas le droit de modifier ce fichier.<br>';
}

if(is_executable('test.ini')){
    echo 'Vous avez le droit d\'exécuter ce fichier.<br>';
}else{
    echo 'Vous n\'avez pas le droit d\'exécuter ce fichier.<br>';
}

?>
```

**Figure 13-5**  
*Gestion des droits*



## Changement de propriétaire

Il est possible de modifier le groupe ou l'utilisateur propriétaire d'un fichier grâce aux fonctions `chown()` et `chgrp()`. Ces fonctions prennent en paramètres l'adresse du fichier à modifier et le nom (ou identifiant numérique) de l'utilisateur ou du groupe cible.

Il n'est normalement possible de modifier l'utilisateur propriétaire que si vous êtes super-utilisateur (`root` sous Unix). De même, vous ne pourrez modifier le groupe que si vous êtes le propriétaire du fichier et que le groupe cible soit un des groupes auxquels vous appartenez.

```
// Change le propriétaire du répertoire /home/eda
chown('/home/eda', 'eda');
```

## Modifier les permissions

Si vous êtes super-utilisateur ou propriétaire d'un fichier, vous pouvez en modifier les permissions. Les nouvelles permissions sont à fournir sous forme numérique en second argument à la fonction `chmod()`, le premier argument étant l'adresse du fichier à modifier.

```
<?php
chmod ("/data/www/locale.pdf", 0755);
// Notation octale : valeur du mode correcte
?>
```

### Attention

La fonction ne présume pas de la base numérique utilisée. Les droits d'accès sont habituellement représentés sous forme octale (suite de chiffres de 0 à 7). Vous voudrez donc probablement préfixer votre nombre par 0 pour signaler que vous utilisez un nombre sous forme octale.

## Masque par défaut

Si un fichier est créé lors de son ouverture, il hérite automatiquement des droits d'accès du script (généralement ceux du serveur web) : utilisateur et groupe propriétaires, droits d'accès par défaut de cet utilisateur.

Les droits d'accès Unix par défaut sont gérés avec un principe de masque. Avec un masque nul, les fichiers sont librement accessibles (permissions 0777). Avec un masque XYZ, les permissions seront de 0(7-X)(7-Y)(7-Z). Assez souvent, on met le masque à 0022 (pas de droits d'écriture pour un autre que le propriétaire) ou 0077 (aucun accès pour un autre que le propriétaire).

Vous pouvez redéfinir le masque par défaut en le fournissant en argument à la fonction `umask()`. N'oubliez cependant pas le zéro de préfixe pour montrer qu'il s'agit d'une valeur octale.

Si vous ne fournissez aucun argument, le masque actuel vous sera retourné sans être modifié.

## Sécurité et fichiers

### *Permissions et droits d'accès*

Quand vous créez un fichier, il obtient automatiquement toutes les permissions moins celles bloquées par le masque. Pensez donc à restreindre le plus possible ce masque par défaut. Cela est principalement important si vous stockez des données confidentielles comme des mots de passe.

Pensez aussi que, sur une plate-forme web, vous héritez le plus souvent de l'utilisateur et du groupe du serveur web. Tous les scripts du serveur pourront potentiellement accéder aux mêmes fichiers que vous. Si c'est le cas, envisagez avec sérieux l'utilisation d'une base de données qui aura un contrôle d'accès indépendant de celui du système.

### *Arguments utilisateur*

Une erreur très courante est de choisir le fichier à lire ou écrire selon une donnée reçue d'un formulaire ou d'un élément extérieur à votre application. Si ce principe ne pose pas problème en lui-même, il faut faire très attention à ce que peut fournir l'utilisateur.

Il pourrait par exemple vous demander de lire le fichier de mots de passe système ou celui contenant vos paramètres de connexion à la base de données. Vérifiez donc toujours que l'utilisateur n'essaie pas d'accéder à une adresse qui ne correspond pas à ce que vous attendiez.

Dans l'idéal, établissez une liste des adresses autorisées et vérifiez que celle que vous comptez utiliser s'y trouve. Sinon, réfléchissez à la forme qu'auront vos adresses et aux contraintes que vous leur mettez. Vérifiez alors toujours l'adresse avec la forme idéale et vos contraintes après l'avoir fait passer par `realpath()` pour éviter que l'utilisateur n'essaie de vous tromper.

### *Safe\_mode et open\_basedir*

La directive de configuration `safe_mode` influe beaucoup sur les accès aux fichiers. À chaque accès, PHP vérifie que le propriétaire du fichier auquel vous voulez accéder est bien le même que l'utilisateur en cours. C'est une fonctionnalité contraignante, mais qui vous épargne beaucoup de dégâts (principalement des problèmes de divulgation) si jamais votre application se révèle avoir une faille de sécurité.

La directive `open_basedir` vérifie quant à elle que tous les fichiers auxquels vous essayez d'accéder sont bien dans des sous-répertoires du répertoire spécifié dans la configuration. C'est un contrôle supplémentaire par rapport aux droits d'accès du système. Il empêche en particulier plusieurs utilisateurs sur le même serveur (donc partageant tous l'utilisateur du serveur web) de pouvoir lire et écrire sur les fichiers des autres.

Nous vous conseillons fortement de réfléchir à l'activation de ces deux directives si cela vous est possible.

## Cas d'application

### *Outil de gestion documentaire simple*

#### Contexte

Pour vos documents internes, les utilisateurs avaient l'habitude de partager un même espace FTP où ils mettaient à jour directement les fichiers. Naturellement, constatant le besoin d'archives, ils ont commencé à ajouter un numéro de version et les initiales du dernier auteur dans le nom du document à chaque modification (laissant l'ancienne copie sur le serveur FTP). Une application interne relisant les documents a formalisé cette procédure.

Les noms de fichiers sont de la forme :

```
nomDocument-numeroVersion-initialesDernierAuteur.extension
```

L'accès libre au FTP pose toutefois des problèmes :

- La mise en place de contrôles d'accès fins est délicate (elle impose que chaque utilisateur définisse bien à chaque fois qui doit être capable de relire son fichier).
- Il n'y a pas de contrôle si deux personnes modifient le même document en même temps.
- Rien à part les sauvegardes n'assure que les archives restent sur le FTP.
- Et surtout, il arrive que les utilisateurs oublient de changer le numéro de version ou les initiales de l'auteur.

Afin de faciliter les modifications par les utilisateurs, il a été décidé de développer une application qui gère toute seule l'incrémentation des numéros et la modification des initiales. Elle pourrait même prévenir si jamais deux personnes ont modifié la même version.

Naturellement, on aurait envie d'utiliser une base de données, mais cela nécessiterait de refaire l'application qui se sert déjà de la structure des fichiers sur le FTP. Il a donc été décidé de ne se baser que sur les fichiers.

#### Réalisation

On considère pour cet exemple que votre applicatif web fournit déjà une interface pour envoyer un fichier par formulaire (voir le chapitre 8 pour plus de détails sur ce sujet) et une pour récupérer les initiales de la personne qui utilise la plate-forme. Pour cet exemple simple, le fichier ne doit pas contenir de tiret dans son nom et doit être normé comme indiqué :

```
nomDocument-numeroVersion-initialesDernierAuteur.extension
```

Il suffit donc à un utilisateur de récupérer la dernière version du fichier, de le modifier, de l'enregistrer (sans changer son nom) et de l'envoyer par le formulaire. Le système se chargera de le mettre à jour.

Nous ne développerons que la partie du script qui gère le « versionnement » et la gestion des fichiers, en laissant de côté toute l'interface HTML. Néanmoins, voici un exemple minimaliste de ce pourrait être ce formulaire :

```
<?php
session_start();
//Ici, on définit par défaut les initiales de l'auteur
if(!isset($_SESSION['initialesAuteur']))
    $_SESSION['initialesAuteur'] = "cpdg";
?>
<html>
<head><title>envoyer un fichier</title></head>
<body>
    <form action="get_info.php"
        method="POST" enctype="multipart/form-data">
        <p>
            <input type="file" name="fichier">
            <input type="submit" value="Envoyer">
        </p>
    </form>
</body>
</html>
```

Attention, les fichiers que vous envoyez doivent être nommés suivant la structure établie précédemment, sinon le script ne fonctionnera pas. Dans un script en production, il vous incombera de vérifier ce point.

La première étape est de récupérer les informations sur le fichier téléchargé :

- le nom du fichier, qui nous indique de quel document il s'agit,
- quelle est la version qui a été modifiée,
- quel est l'ancien auteur.

```
<?php
// Fichier envoyé au site Web
$nomFichier = $_FILES['fichier']['name'] ;

// On ne garde que le nom du fichier, pas le chemin
$nomFichier = basename($nomFichier) ;

// Récupération de l'extension
$dernierPoint = strrpos($nomFichier, '.') ;
$extension = substr($nomFichier, $dernierPoint) ;
$nomFichier = substr($nomFichier, 0, $dernierPoint) ;

// Récupération du précédent auteur
$dernierTiret = strrpos($nomFichier, '-' ) ;
$ancienAuteur = substr($nomFichier, $dernierTiret +1) ;
$nomFichier = substr($nomFichier, 0, $dernierTiret) ;
```

```
// Récupération de la version modifiée
$dernierTiret = strrpos($nomFichier, '-' );
$ancienneVersion = substr($nomFichier, $dernierTiret +1) ;
$nomFichier = substr($nomFichier, 0, $dernierTiret) ;

// Le reste est le nom du document
$document = $nomFichier ;
```

Il nous faut ensuite aller regarder sur le répertoire quelle est la dernière version de ce document.

```
$versionActuelle = 0 ;
foreach( glob($document.'-*'.'.$extension) as $fichier ) {
    $debut = strrpos($fichier, '-' ) + 1;
    $longueur = strrpos($fichier, '-') - $debut;
    $version = substr($fichier, $debut, $longueur) ;
    if ($version > $versionActuelle) {
        $versionActuelle = $version ;
        $fichierActuel = $fichier ;
    }
}
```

Il ne nous reste plus qu'à interpréter ces résultats et renvoyer le résultat à l'utilisateur :

```
session_start() ;
if ($versionActuelle > $ancienneVersion) {
    echo "Le document a été modifié depuis votre dernier passage. " ;
    echo "Veuillez récupérer la dernière version ($fichier), " ;
    echo "la mettre à jour et la renvoyer." ;
} else {
    $fichier = $document
        .'-' .($ancienneVersion + 1)
        .'-' .$_SESSION['initialesAuteur']
        . $extension ;
    move_uploaded_file($_FILES['fichier']['tmp_name'], $fichier) ;
    echo "fichier correctement mis à jour vers $fichier" ;
}
?>
```

**Note**

Aucune vérification d'erreur ou de sécurité n'a été faite dans cet exemple. Il est bien entendu nécessaire que vous preniez en compte ces vérifications si vous implémentez un système de ce type.

# 14

## Gestion des flux

---

Le terme de flux est un terme générique désignant une suite de données ; il décrit un écoulement. Les flux reposent sur les mêmes concepts que ceux vus au chapitre précédent sur les fichiers.

Nous étudierons deux types de flux spécifiques. Le premier concerne les données transitant lors de l'exécution d'un programme. Le second traite des *sockets* TCP/IP (vous permettant d'ouvrir des connexions vers la plupart des services réseaux : serveurs web, DNS, services web, etc).

Nous généraliserons ensuite les sujets précédents pour voir qu'il existe une multitude d'abstractions possibles utilisant le même modèle. On peut ainsi utiliser de manière transparente des fichiers *.zip* ou créer ses propres abstractions.

### Exécution de programmes

Utiliser des programmes externes permet d'intégrer PHP à un environnement logiciel existant et de les faire interagir. L'exécution d'un programme via PHP est très simple mais nécessite tout de même une grande attention si vous utilisez des données utilisateur comme paramètres.

#### *Lancement sans interactions*

Le lancement d'un programme externe se fait généralement sans interactions, c'est-à-dire qu'une fois le programme lancé, PHP ne fera qu'attendre le résultat. Il ne communique donc pas avec le programme pendant l'exécution.



## Commande shell rapide

Une commande *shell* est une commande du système comme `dir`, `ls`, `cp`, etc. La liste des commandes *shell* dépend de votre système.

La fonction `shell_exec()` prend en paramètre une commande et en retourne le résultat. Il est aussi possible d'utiliser directement une commande en l'entourant d'apostrophes inverses (aussi dites apostrophes obliques, *backtick* en anglais, caractère ```).

### Attention

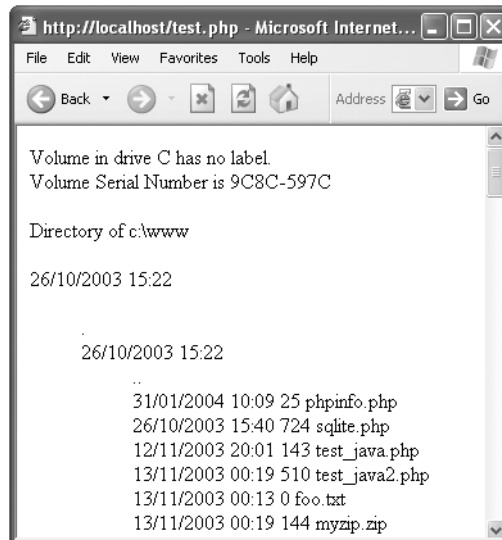
Cette dernière syntaxe est toutefois déconseillée parce qu'elle est difficilement lisible et qu'on risque de la confondre avec une chaîne de caractères entre apostrophes classiques.

Le script suivant lit le contenu d'un répertoire sous un système Windows. On peut voir le résultat à la figure 14-1.

```
<?php
// Sous Windows
$rep = shell_exec('dir') ;
// ou $rep = `dir`;
echo nl2br(utf8_decode($rep)) ;
?>
```

Figure 14-1

Exécution de la commande `dir`



L'exemple suivant fait la même chose sous un système Linux :

```
<?php
// Sous Linux
```

```
$rep = shell_exec('ls -la') ;  
// ou $rep = `ls -la` ;  
echo nl2br($rep) ;  
?>
```

## Exécution d'un programme

La fonction `exec()` permet d'exécuter le programme indiqué en premier argument. Contrairement à la fonction `shell_exec()`, elle ne retourne par défaut que la dernière ligne du résultat.

```
exec(commande [,sortie_standard [,code_retour]])
```

Si vous souhaitez connaître l'ensemble du texte que le programme a renvoyé sur la sortie standard, il vous faut fournir une variable en second paramètre. La fonction remplira alors un tableau contenant les différentes lignes renvoyées par l'exécution du programme.

### Note

Si vous souhaitez avoir directement ce résultat comme valeur de retour de la fonction, vous pouvez utiliser la fonction `passthru()`.

Enfin, les programmes envoient généralement un code retour indiquant la réussite ou non de leur exécution. Le troisième paramètre (optionnel) de la fonction permet de le récupérer.

```
<?php  
exec('./maj_donnees.exe',$result,$code_retour);  
?>
```

Trois fonctions similaires existent :

- la fonction `shell_exec()`, qui a l'avantage de renvoyer simplement tout le résultat en un bloc de texte sans découpage par lignes, mais qui ne permet pas de récupérer la valeur de retour ;
- la fonction `passthru()`, qui a comme valeur de retour le résultat de l'exécution du programme ;
- la fonction `system()`, qui affiche le résultat au lieu de le retourner dans une variable.

```
<?php  
system('rm -rf /tmp', $retour) ;  
if ( !$retour ) echo 'L\'exécution s'est mal passée' ;  
?>
```

### Note

La fonction `system()` vide le tampon d'affichage vers le navigateur après chaque ligne envoyée. Elle vous empêchera d'envoyer des *cookies* ou des en-têtes HTTP par la suite.

Voici un exemple plus poussé de l'utilisation de la fonction `exec()`. Nous simulons ici la fonction PHP `checkdnsrr()`, qui ne marche pas sur les plates-formes Windows.

```
<?php
function checkdnsrr_winNT( $host, $type = '' )
{
    if( !empty( $host ) )
    {
        // On définit le type par défaut
        if( $type == '' ) $type = 'MX';
        @exec( "nslookup -type=$type $host", $output );
        while( list( $k, $line ) = each( $output ) )
        {
            if( eregi( "^$host", $line ) )
            {
                return true;
            }
        }
        return false;
    }
}
echo checkdnsrr_winNT('anaska.com');
?>
```

### Programmes en tâche de fond

Il est possible de lancer des programmes en tâche de fond avec `system()` et `exec()`. Vous pouvez ainsi lancer une tâche longue sans attendre le résultat (ou faire attendre le visiteur). Pour cela, il faut rediriger la sortie standard et la sortie d'erreur. Sous Unix, vous pouvez par exemple rediriger ces deux sorties vers le périphérique nul avec la syntaxe `1>/dev/null 2>&1`.

```
<?php
system('rm -rf /tmp >/dev/null 2>&1', $retour) ;
?>
```

Attention cependant, car normalement l'arrêt d'un processus entraîne l'arrêt de tous les processus fils et donc peut affecter vos programmes se déroulant en tâche de fond. Deux comportements sont possibles selon la façon dont vous avez couplé PHP et votre serveur web :

- Si vous utilisez PHP en CGI ou en ligne de commande, les programmes tournant en tâche de fond seront arrêtés quand le script PHP se terminera.
- Sous un module Apache, si le serveur web est configuré pour redémarrer ses fils au bout d'un certain nombre de requêtes le programme se terminera aussi après un temps aléatoire.

Pour éviter ce fonctionnement, il faut demander au programme de se déconnecter du processus père (celui qui exécute le script PHP). Sous Unix, on préfixera la commande à exécuter par la commande `nohup`.

```
<?php
$cmd = '/etc/initd/apache startssl';
exec('nohup " '.$cmd.'" >/dev/null 2>&1');
?>
```

### Attention

Faites attention aux programmes tournant en tâche de fond. Il est facile d'en lancer plusieurs et d'en faire tourner trop, ou de les oublier et de les laisser consommer inutilement des ressources système. Dans le cadre d'une application web, il est peu probable que ce soit une bonne idée de faire tourner un programme en tâche de fond.

## Lancement interactif

Les fonctions précédentes sont très pratiques et relativement simples à utiliser. Elles ne permettent cependant pas de dialoguer avec un programme externe. Le programme pourrait par exemple demander un paramètre, un mot de passe ou une confirmation avant de se terminer. Nous allons maintenant voir les fonctions qui permettent cette interaction.

### Flux d'entrée, de sortie et d'erreur

Avant d'aller plus loin dans la description de ce jeu de fonctions, il est important de comprendre comment un programme interagit avec son environnement.

Classiquement, un programme comprend trois flux et une valeur de retour :

- La valeur de retour est un code numérique simple permettant de savoir dans quel état le programme s'est arrêté. Il est habituellement égal à 0 quand tout s'est bien passé et non nul quand il y a eu une erreur.
- Le flux de sortie (*output*) permet au programme d'envoyer une réponse à celui qui a exécuté le programme. Par exemple, l'affichage vers l'écran peut être un flux de sortie.
- Le flux d'entrée (*input*) permet d'envoyer des commandes ou des réponses au programme.
- Le flux d'erreur permet au programme de signaler des problèmes.

Habituellement, les flux de sortie et d'erreur sont redirigés vers l'affichage (tout ce que le programme renvoie est affiché) et le flux d'entrée est connecté au clavier (tout ce qui est tapé est reçu par le programme). Ces flux peuvent toutefois être récupérés ou redirigés.

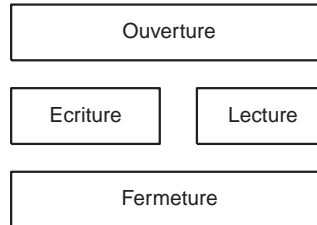
Ces trois flux sont les flux standards que vous retrouverez pour tout programme. Le flux d'entrée est le flux d'identifiant 0, le flux de sortie aura l'identifiant 1 et le flux d'erreur l'identifiant 2. Un programme peut bien sûr ouvrir autant d'autres flux qu'il le souhaite, que ce soit en lecture ou en écriture ; ces flux supplémentaires prendront alors les numéros disponibles suivants.

## Ouverture des flux

La procédure de gestion est la même que pour un fichier (voir figure 14-2) : ouverture, lecture et écriture, puis fermeture.

**Figure 14-2**

*Procédure de gestion de flux*



Si vous ne voulez ouvrir qu'un seul flux (soit le flux d'entrée, soit le flux de sortie), il vous suffit d'utiliser la fonction `popen()`.

`popen (commande, mode)`

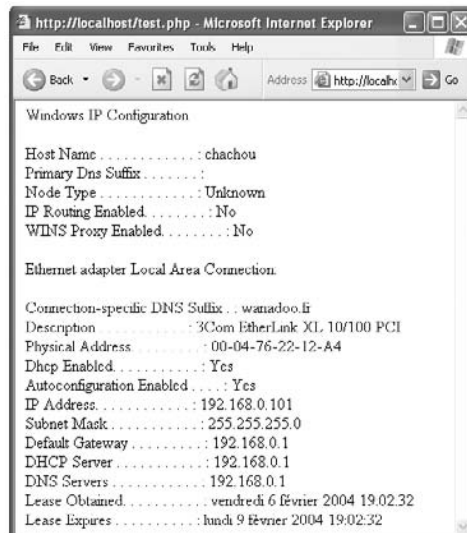
Le premier paramètre est la commande à exécuter. Si vous sélectionnez le mode d'écriture (`w`), vous aurez accès au flux d'entrée, et si vous choisissez le mode lecture (`r`), vous aurez accès au flux de sortie. Vous trouverez le résultat du code suivant à la figure 14-3.

```
<?php
/* Dans notre exemple, on affiche le résultat de l'appel à
ipconfig */

$fp = popen('ipconfig /all', 'r');
echo nl2br(fread($fp,5000));
?>
```

**Figure 14-3**

*Utilisation de la commande `popen`*



**Remarque**

La fonction `popen()`, comme les autres fonctions de gestion des flux, est à rapprocher des fonctions de gestion des fichiers. Ainsi, `popen()` fonctionne de la même manière que `fopen()`.

Utiliser `popen()` plutôt que `exec()` ou `shell_exec()` ne semble pas forcément très utile. Le gain ne sera présent que si vous avez besoin d'une abstraction (gérer la lecture d'un programme de la même façon qu'un fichier) ou si vous comptez utiliser les filtres de flux qui sont décrits en fin de chapitre.

**Connecter plusieurs flux simultanément**

Si vous avez besoin de plusieurs flux sur une même exécution, il vous faudra alors vous reporter vers la fonction `proc_open()`:

```
proc_open( commande, descriptorspec, pipes)
```

Cette fonction prend trois paramètres : une chaîne de caractères pour la commande à exécuter, la liste des flux à connecter, et une variable dans laquelle PHP retournera la liste des descripteurs obtenus. La fonction renvoie un identifiant qui décrit la commande en cours d'exécution. Voici un exemple d'utilisation :

```
$commande = '/usr/local/bin/php' ;  
$flux = array( /* voir plus bas */ ) ;  
$proc = proc_open($commande, $flux, $descripteurs) ;
```

Pour lire l'affichage (flux de sortie, identifiant 1) avec PHP, il faut donner à `$flux` la définition suivante :

```
$flux = array( 1 => array('pipe', 'r') ) ;
```

On n'utilise ici qu'un seul élément (tableau à une entrée). L'élément utilisé est le flux de sortie (la clé est l'identifiant du flux à connecter, 1 dans notre cas). Je souhaite pouvoir l'utiliser directement avec PHP (valeur 'pipe'), en lecture (mode d'accès 'r'). Une fois exécutée `proc_open()`, la variable `$descripteurs` sera un tableau indexé et l'élément avec la clé 1 (l'identifiant du flux que je veux utiliser) sera un descripteur de fichier classique.

Nous aurions pu aussi rediriger le flux d'erreur vers un fichier de *log*, en demandant que les nouvelles erreurs soient ajoutées au fichier actuel :

```
$flux = array( 2 => array('file', '/var/log/fichier.txt', 'a') ) ;
```

L'index 2 indique qu'on utilise le flux standard d'erreur, la valeur 'file' demande que le flux soit redirigé vers un fichier au lieu d'être traité manuellement par notre script PHP,

et les deux paramètres suivants sont l'adresse du fichier à utiliser et son mode d'ouverture.

Si nous avions voulu utiliser les trois flux classiques dans PHP, nous aurions pu avoir la définition suivante :

```
$flux = array(
    0 => array('pipe', 'w') , // Flux d'entrée en écriture
    1 => array('pipe', 'r') , // Flux de sortie en lecture
    2 => array('file', '/var/log/fichier.txt', 'a') // Flux d'erreur en écriture
);
```

On aurait alors eu deux descripteurs de fichiers aux index 0 et 1 dans le tableau `$descripteurs`, et les erreurs éventuelles auraient été signalées dans le fichier `/var/log/fichier.txt`.

### Lecture et écriture

La fonction `popen()` retourne un descripteur et la fonction `proc_open()` renvoie dans son troisième paramètre la liste des descripteurs de fichiers demandés.

Ces descripteurs s'utilisent exactement comme des descripteurs de fichiers. Vous pouvez utiliser les fonctions de lecture et d'écriture décrites dans la partie sur le traitement des fichiers.

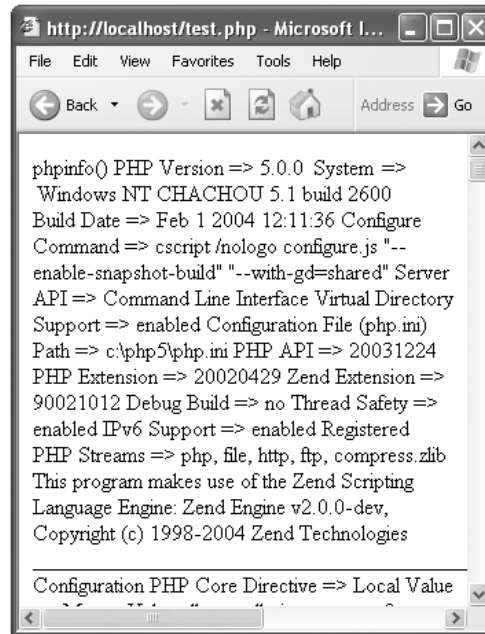
Pour exemple, cette commande exécute un fichier PHP contenant un appel à la fonction `phpinfo()` et affiche le résultat à l'écran. Les erreurs sont stockées dans un fichier de *log*. On peut voir le résultat en figure 14-4.

```
<?php
$flux = array(
    0 => array('file', 'test2.php', 'r') ,
        // Flux d'entrée en lecture. Le fichier contient un phpinfo()
    1 => array('pipe', 'w') , // Flux de sortie en écriture
    2 => array('file', './fichier.txt', 'a') // Flux d'erreur en ajout
);
$commande = 'c:/php5/php.exe';
$proc = proc_open($commande, $flux, $descripteurs);

$fp = $descripteurs[1];
while( $line = fgets($fp, 1024) ) {
    echo htmlspecialchars($line);
}
?>
```

Figure 14-4

Exécution de  
commande avec  
`proc_open()`



### Statut des programmes exécutés

Lors de l'exécution avec `proc_open()`, vous pouvez obtenir toutes les informations disponibles sur le programme lancé avec la fonction `proc_get_status()`. Cette dernière fonction prend comme unique paramètre l'identifiant retourné lors de l'ouverture et retourne un tableau associatif avec différentes informations. On peut voir le résultat du script suivant à la figure 14-5 :

```
<?php
$flux = array(
    0 => array('file', 'test2.php', 'r') ,
    1 => array('pipe', 'w') ,
    2 => array('file', './fichier.txt', 'a')
) ;
$commande = 'c:/php5/php.exe';
$proc = proc_open($commande, $flux, $descripteurs) ;
$info = proc_get_status($proc) ;

echo 'commande executée : ', $info['command'], '<br>' ;
echo 'identifiant processus : ', $info['pid'], '<br>' ;
if ( $info['running'] ) {
    echo 'le programme est toujours en exécution <br>' ;
} else {
    echo 'le programme a fini son exécution <br>' ;
    echo 'sa valeur de retour est : ', $info['exitcode'], '<br>' ;
}
```



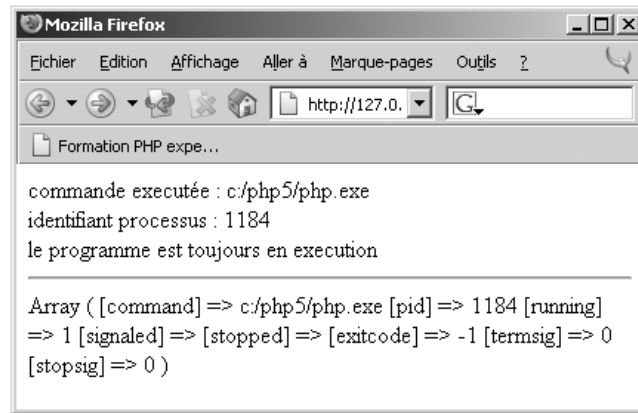
```

if ( $info['signaled'] ) {
    echo 'le programme s est arrêté suite à un signal inconnu <br>' ;
    echo 'le signal reçu est : ', $info['termsig'], '<br>' ;
} elseif ( $info['stopped'] ) {
    echo 'le programme a été stoppé suite à un signal <br>' ;
    echo 'le signal reçu est : ', $info['stopsig'], '<br>' ;
}
echo '<hr>';
print_r($info);
?>

```

Figure 14-5

*Statut des programmes exécutés*



## Fermeture

Comme pour les fichiers, il vous faut fermer les ressources ouvertes dès que vous avez fini de les utiliser. Si vous avez ouvert le processus avec `popen()`, il vous suffit de faire appel à la fonction `pclose()` avec le descripteur comme unique argument. La fonction vous renverra le code de retour du programme (un entier, généralement non nul en cas d'erreur et nul si tout s'est bien passé).

Si en revanche vous avez ouvert des flux avec `proc_open()`, il vous faudra d'abord fermer chaque flux indépendamment avec `fclose()` (comme pour un fichier classique). Ensuite, vous pourrez fermer le processus lui-même avec `proc_close()`, en fournissant comme unique paramètre la valeur retournée par `proc_open()` à l'ouverture.

```

$flux = array(
    0 => array( file", "script.php", "r" ) ,
    1 => array("pipe", "w" ) ,
    2 => array("file", "/var/log/fichier.txt", "a")
) ;
$commande = "/usr/local/bin/php-cli" ;
$proc = proc_open($commande, $flux, $descripteurs) ;

```

```
$fp = $descripteurs[1] ;  
while( $line = fgets($fp, 1024) ) {  
    echo htmlspecialchars($line) ;  
}  
fclose( $descripteurs[0] ) ;  
fclose( $descripteurs[2] ) ;  
proc_close($proc) ;
```

## Sécurité et programmes externes

### Données externes dans la commande

Lancer un programme à partir d'une donnée externe (par exemple un paramètre utilisateur) est un point qui doit être mûrement réfléchi. Il serait en effet facile pour l'utilisateur d'envoyer des paramètres qui seront interprétés de manière non prévue, ou qui exploiteront des vulnérabilités du système. Vous devrez toujours vérifier la cohérence des données utilisées avec celles que vous attendez.

Pour vous aider à minimiser les risques, PHP met deux fonctions à votre disposition : `escapeshellcmd()` et `escapeshellarg()`. Elles vous permettront de protéger les caractères interprétables et les caractères spéciaux dans les paramètres envoyés aux programmes. C'est un peu l'équivalent de `addslashes()` pour les bases de données.

Vous pourrez trouver plus de détails sur ces fonctions et sur l'attention qu'il faut leur porter dans le chapitre sur la sécurité.

### safe\_mode et open\_basedir

Si le `safe_mode` est activé, l'accès aux programmes vous sera par défaut refusé. La directive `safe_mode_exec_dir` contient alors une liste de répertoires dans lesquels se trouvent les seuls programmes utilisables. C'est une manière simple pour l'administrateur de limiter les interactions avec le système aux seuls programmes auxquels il accorde sa confiance.

En effet, il faut bien prendre en compte que PHP ne peut pas surveiller ce que fait un programme lancé. Le programme aura alors accès à toutes les ressources disponibles pour votre utilisateur sur le système. En particulier, la directive `open_basedir` n'aura aucun effet.

## Gestion des sockets réseau

Pour communiquer avec des services TCP/IP comme les serveurs web ou les serveurs Telnet, vous aurez besoin d'ouvrir une *socket* réseau. La gestion des *sockets* n'est pas plus complexe que l'utilisation des fichiers ou celle des processus. En fait, les fonctions sont pour la plupart les mêmes.

## Ouverture

La fonction `fsockopen()` permet d'ouvrir une *socket* en tant que client.

```
fsockopen (adresse, port [,coderreur ,texterreur ,timeout])
```

En lui fournissant une adresse et un port en arguments, elle vous connectera au serveur demandé et renverra un descripteur pour faire les lectures et écritures.

```
<?php
// Se connecte au serveur Web de php.net
$fp = fsockopen ('www.php.net', 80);
// Se connecte pour ouvrir une session telnet
$fp = fsockopen ('192.168.2.6', 23);
?>
```

En cas d'erreur, la fonction renvoie la valeur `FALSE`. Vous pouvez toutefois récupérer un code et un texte d'erreur en fournissant deux variables supplémentaires comme arguments. Le code d'erreur sera retourné dans la première et le texte dans la seconde.

Si vous fournissez un cinquième argument, il sera pris comme un temps d'expiration (*timeout*) en secondes. Si la connexion n'est pas établie dans la limite imposée, PHP considère qu'il y a échec et rend la main.

```
<?php
// Se connecte au serveur Web de php.net
$timeout = 10 ; // 10 secondes pour établir la connexion
$fp = fsockopen ('www.php.net', 80, $erno, $errmsg, $timeout);
if (!$fp) {
    echo "Erreur $erno : $errmsg" ;
}
?>
```

### Adresses IP et noms de domaine

Pour identifier un serveur, vous pouvez utiliser indifféremment son adresse IP ou son nom de domaine. Si vous utilisez le nom de domaine, PHP fera une requête DNS pour chercher la correspondance avec l'adresse IP. Cette requête peut prendre un temps non négligeable à forte charge.

PHP peut de plus utiliser les nouvelles adresses IP version 6 (IPv6). Si vous utilisez une telle adresse, il vous faudra obligatoirement entourer cette adresse IPv6 par des crochets afin qu'elle ne gêne pas l'interprétation du reste de la chaîne.

## Type de sockets

Par défaut, PHP établit des *sockets* TCP (ce sont celles que vous voudrez probablement utiliser, qui servent pour les protocoles mail, pour les serveurs web, pour le FTP et la plupart des services Internet).

Vous pouvez toutefois utiliser d'autres types de transports. En préfixant l'adresse par `udp://` PHP utilisera des *sockets* UDP. Les protocoles `ssl://` et `tls://` sont aussi disponi-

bles si vous avez compilé PHP avec la prise en charge d'openssl pour pouvoir établir des connexions sécurisées. Les *sockets* Unix sont aussi accessibles via le préfixe `unix:`.

#### Attention

Si vous utilisez le protocole UDP, rappelez-vous bien qu'il s'agit d'un protocole déconnecté sans garantie de réception. Tout ce qui est envoyé par le serveur ou par vous ne sera pas forcément reçu à l'autre bout et aucune erreur ne sera retournée en cas de mauvaise réception.

## Lecture et écriture

Les fonctions de lecture et d'écriture sont exactement les mêmes que pour les accès aux fichiers et aux processus : `fgets()`, `fwrite()`, `fread()`, `feof()`, etc. Vous pouvez vous reporter au chapitre 13 concernant les fichiers pour plus de détails.

```
<?php
$fp = fsockopen('www.php.net', 80);
fputs($fp, "GET / HTTP/1.1\r\n") ;
fputs($fp, "Host: www.php.net\r\n") ;
fputs($fp, "Connection: close\r\n") ;
fputs($fp, "\r\n");
while(!feof($fp)) {
    echo fgets($fp,128);
}
?>
```

## Fermeture

Une fois la *socket* utilisée, vous pouvez la fermer avec `fclose()`, exactement comme pour un fichier classique.

## Fonctions de contrôle

### Fonctions bloquantes

Par défaut, les *sockets* sont ouvertes dans un mode dit bloquant. Quand on lit une telle *socket*, PHP attend que les données à lire arrivent ou que la *socket* soit close avant de rendre la main. Si les données mettent du temps avant d'arriver, votre script sera bloqué pendant de longues secondes.

Ce comportement est pratique quand on a besoin d'une donnée et qu'on ne peut rien faire d'autre en l'attendant. Pourtant, dans certains cas, il peut être utile de simplement lire si des données sont disponibles et, sinon, passer à la suite, quitte à refaire une lecture plus tard.

Pour basculer entre les deux méthodes, vous pouvez utiliser la fonction `stream_set_blocking()`.

```
stream_set_blocking (idsocket, mode)
```

Elle prend en paramètres le descripteur de la *socket* et un booléen. Si ce booléen est vrai, alors la *socket* utilise des fonctions bloquantes. Dans le cas contraire, les fonctions seront non bloquantes (elles rendront la main tout de suite s'il n'y a aucune donnée en attente).

Si la connexion est non bloquante, il faudra alors refaire une demande de lecture plus tard pour lire ce qui sera arrivé après. Tant que `feof()` renvoie une valeur fausse, c'est que des données peuvent toujours arriver.

```
$fp = fsockopen ('www.php.net', 80);
stream_set_blocking($fp, TRUE) ;
fputs($fp, "GET / HTTP/1.1\r\n") ;
fputs($fp, "Host: www.php.net\r\n") ;
fputs($fp, "Connection: close\r\n") ;
fputs($fp, "\r\n");
$char = fgetc($fp) ;

// Est équivalent à

$fp = fsockopen ('www.php.net', 80);
fputs($fp, "GET / HTTP/1.1\r\n") ;
fputs($fp, "Host: www.php.net\r\n") ;
fputs($fp, "Connection: close\r\n") ;
fputs($fp, "\r\n");
stream_set_blocking($fp, FALSE) ;
do {
    $char = fgetc($fp) ;
} while( $char === FALSE && !feof($fp)) ;
```

### Temps d'expiration

Nous venons de voir que, parfois, PHP peut attendre plusieurs secondes l'arrivée d'une donnée ou d'une réponse. Au bout d'un certain temps, PHP considère la connexion comme perdue et ferme la *socket*.

Ce temps d'expiration est réglable grâce à la fonction `stream_set_timeout()`. Elle prend en paramètres le descripteur utilisé et un temps d'attente maximal en secondes.

```
<?php
$fp = fsockopen ('www.php.net', 80);
stream_set_blocking($fp, TRUE) ;

// Si le serveur ne donne plus signe de vie pendant 10 secondes,
// on considère la connexion comme perdue
stream_set_timeout($fp, 10) ;

fputs($fp, "GET / HTTP/1.1\r\n") ;
fputs($fp, "Host: www.php.net\r\n") ;
fputs($fp, "Connection: close\r\n") ;
fputs($fp, "\r\n");
$char = fgetc($fp) ;
?>
```

## Statut de la connexion

La fonction `stream_get_meta_data()` permet de connaître l'état d'une *socket*. C'est l'équivalent pour les *sockets* de `proc_get_status()` qui servait pour les exécutions de programme. En fournissant un descripteur en argument, la fonction retourne un tableau associatif avec des informations sur la *socket* en cours :

```
<?php
$fp = fsockopen ('www.php.net', 80);
fputs($fp, "GET / HTTP/1.1\r\n");
fputs($fp, "Host: www.php.net");
fputs($fp, "Connection: close\r\n");
fputs($fp, "\r\n");
sleep(5);
$info = stream_get_meta_data ( $fp );
if ( $info['timed_out'] ) {
    echo 'le serveur ne répond plus, la connexion a été coupée <br>' ;
}
if ( $info['blocked'] ) {
    echo 'la connexion est en mode bloquant <br>' ;
} else {
    echo 'la connexion est en mode non bloquant <br>' ;
}
if ( $info['eof'] ) {
    echo 'le socket a fini d envoyer des données <br>' ;
}
echo 'Il y a ', $info['unread_bytes'], ' octets non écrits <br>' ;
?>
```

### Note

Même si la *socket* a fini d'envoyer des données (la clé `eof` renvoie vrai), il peut encore y avoir des données en attente de lecture. Pour tester si on a lu toutes les données à lire, il faut utiliser la fonction `feof()` décrite plus haut.

## Gestion unifiée des flux

Vous avez pu remarquer que PHP savait gérer les fichiers de manière transparente, qu'ils soient locaux, sur un serveur web ou sur un serveur FTP. La gestion des fichiers est elle-même assez proche de la gestion des *sockets* réseaux ou des exécutions de programmes.

C'est sur ce constat que, depuis sa version 4.3, PHP vous offre une gestion unifiée de tous les flux de données : fichiers locaux, fichiers distants, fichiers compressés, données cryptées, *sockets*, programmes, etc. Vous pouvez alors accéder de manière transparente et avec les mêmes fonctions à tous ces flux de données. Il est même possible de définir ses propres abstractions pour définir un type de flux personnalisé.

Vous avez aussi la possibilité d'associer plusieurs filtres lors de l'utilisation d'un flux. On peut ainsi faire une conversion automatique d'un jeu de caractères vers un autre, pour gérer de manière transparente des fichiers UTF-8.

## Types de flux gérés

Lors de la description de la fonction `fopen()`, dans la gestion des fichiers, nous avons présenté des méthodes pour accéder à des adresses HTTP ou FTP. La syntaxe utilisée permet en fait d'utiliser pratiquement n'importe quel type de flux. Une adresse peut donc correspondre aussi bien à un fichier sur le disque, à un fichier sur le réseau, à une *socket* réseau ou même à un fichier compressé.

La syntaxe générale d'un flux est `transport://cible`. La première partie désigne le protocole ou la fonction d'abstraction utilisée, la deuxième partie est l'adresse du flux via ce protocole. On a par exemple l'adresse du fichier `file://etc/passwd` ou de la page web `http://www.php.net/`. Vous pouvez utiliser n'importe quelle adresse avec les fonctions classiques de lecture et d'écriture (décrites principalement dans la gestion des fichiers) sans vous préoccuper du type d'abstraction en jeu.

### Liste des abstractions gérées

Nous détaillons ici une liste non exhaustive des types de flux gérés afin d'en décrire les limitations et les particularités. Vous pourrez avoir plus ou moins de possibilités suivant votre configuration.

Pour certaines méthodes, des paramètres dits *de contexte* sont indiqués. L'explication des contextes et leur utilisation seront décrits par la suite. De même, des noms de métadonnées peuvent être fournis. Les valeurs correspondantes seront lisibles dans le tableau associatif retourné par la fonction `stream_get_meta_data()`. Une explication de ce que sont ces informations sera donnée plus loin.

#### Fichiers locaux

L'abstraction de fichier local est la plus utilisée. Si vous fournissez une adresse sans préfixe, c'est par défaut ce mode d'accès qui sera utilisé. Vous pouvez toutefois utiliser explicitement les fichiers locaux grâce au préfixe `file://`.

#### Flux compressés

En plus des fichiers classiques, il est possible d'utiliser des flux compressés de manière transparente. Les fichiers compressés avec `gzip` (extension `gz`) sont accessibles via le préfixe `compress.zlib://` et ceux compressés avec `bzip2` (extension `bz2`) avec le préfixe `compress.bzip2://`.

#### Note

Ces possibilités n'existent que si PHP a été compilé avec les modules `zlib` et `bzip2`.

```
<?php
// Exemple d'un chemin d'accès pour la lecture d'un fichier ZIP
$fichier = 'compress.zlib://test.gz';
readfile($fichier) ;

?>
```

Vous pouvez accéder à ces ressources tant en lecture qu'en écriture (mode `w` ou `a`), mais pas simultanément en lecture et en écriture (modes avec le suffixe `+`). Aucune autre fonction ne sera accessible. Pour les effacements de fichiers ou les métadonnées, il vous faudra utiliser les fonctions sans le préfixe de compression (vous opérez alors sur les fichiers et non sur le contenu compressé).

### Fichiers distants

Vous pouvez accéder à des fichiers distants par HTTP ou FTP simplement en utilisant les préfixes `http://` et `ftp://`. Si vous devez spécifier un nom d'utilisateur et un mot de passe, vous pouvez alors utiliser le préfixe `http://nom:pass@` (ou `ftp://nom:pass@`).

Le protocole HTTP est accessible uniquement en lecture, aucun autre type de fonction ne sera autorisé. Pendant cette lecture, seul le contenu de la page vous sera retourné ; les en-têtes seront, eux, accessibles par la clé `http_response_header` dans les métadonnées.

Il est possible de définir les en-têtes envoyés dans la requête HTTP via les options de contexte. La clé `method` décrit la méthode HTTP (GET ou POST). La clé `header` permet de spécifier des en-têtes arbitraires et la clé `content` définit le contenu de la requête (utilisé le plus souvent pour envoyer des informations en POST). Si aucun nom d'agent utilisateur (nom du navigateur, en-tête `User-Agent`) n'est spécifié dans les en-têtes, le paramètre `user_agent` de configuration du `php.ini` sera utilisé.

Le protocole FTP est, lui, utilisable autant en lecture qu'en écriture (modes `w` et `a`), mais pas simultanément en lecture et en écriture (modes avec le suffixe `+`). Vous pouvez aussi utiliser la fonction `unlink()` pour effacer un fichier sur le serveur FTP. Par défaut, si vous ouvrez un fichier existant avec le mode `w`, PHP vous refusera l'accès pour ne pas écraser le fichier existant. Pour autoriser l'écrasement, vous pouvez fournir un booléen vrai à l'option de contexte `overwrite`.

Les protocoles sécurisés HTTPS et FTPS sont accessibles, si vous utilisez le module `openssl`, via les préfixes `https://` et `ftps://`.

Les protocoles HTTP et FTP utilisent tous les deux le protocole réseau TCP. Ils en partagent donc aussi les paramètres et les options. Dans le cas de connexions sécurisées, HTTP et FTP utilisent en interne le protocole SSL. Ils en partagent donc les options de contexte.

#### Note

Ces abstractions ne sont pas disponibles si le paramètre de configuration `allow_url_fopen` est désactivé.



## Flux d'entrée et de sortie PHP

PHP gère, comme tous les programmes, des flux d'entrée et de sortie. Les flux du processus PHP sont accessibles via les adresses `php://stdin` (entrée), `php://stdout` (sortie) et `php://stderr` (erreur).

Plutôt que d'envoyer des données directement sur le flux de sortie du processus PHP, vous voudrez probablement envoyer ces données via le système de sortie classique de PHP. Vous pourrez alors utiliser les filtres de sortie (ces filtres et leur fonctionnement seront décrits au chapitre suivant). Pour envoyer vos données sur le système de sortie de PHP, vous pouvez utiliser l'adresse `php://output`.

L'adresse `php://input` permet quant à elle de lire les données reçues dans la requête d'exécution. Pour une requête web, vous pouvez y lire tous les en-têtes HTTP.

Ces différentes adresses ne sont utilisables qu'en lecture (pour les flux d'entrée) ou en écriture (pour les flux de sortie ou d'erreur).

## Socket réseau

PHP peut vous connecter de manière transparente à une *socket* réseau. Vous la manipulez alors comme vous manipulez un fichier. Les *sockets* gérées peuvent être de type TCP (préfixe `tcp://`), UDP (préfixe `udp://`), Unix (préfixe `unix://` et `udg://`). Par défaut, dans une ouverture réseau (par exemple, via `fsockopen()`), c'est le protocole TCP qui sera utilisé si vous n'utilisez pas de préfixe.

Les adresses seront de type `protocole://adresse:port` (le port n'est pas utilisé pour les *sockets* Unix). Dans la fonction `fsockopen()`, le port doit être spécifié dans un protocole à part (vous pouvez fournir la valeur 0 pour les *sockets* Unix). Si l'adresse est une adresse IPv6, elle devra être entourée de crochets.

### Note

Ces abstractions ne sont pas disponibles si le paramètre de configuration `allow_url_fopen` est désactivé.

## Connexions sécurisées

Si PHP est compilé avec le module `openssl`, vous pouvez utiliser des connexions sécurisées. Vous aurez ainsi accès aux protocoles SSL et TLS via les préfixes `ssl://` et `tls://`.

Ces protocoles étant des surcouches du protocole TCP, ils en partagent les paramètres et les options. Il existe toutefois des options de contexte propres aux protocoles sécurisés.

Si la valeur du nom de `verify_peer` contient un booléen vrai, le certificat SSL sera analysé et vérifié. Si `allow_self_certificate` est vrai, PHP autorisera les certificats autosignés. Les paramètres `cafile` et `capath` permettent de spécifier un fichier de vérification du certificat ou un répertoire contenant ce fichier. La clé `local_cert` peut contenir l'adresse du fichier contenant le certificat local à utiliser, et la clé `passphrase` le mot de passe associé.

La valeur sous le nom `CN_match` permet quant à elle de vérifier que le Common Name correspond à un masque défini.

**Note**

Ces abstractions ne sont pas disponibles si le paramètre de configuration `allow_url_fopen` est désactivé.

### Obtenir la liste des types de flux gérés

Il existe deux fonctions permettant d'obtenir la liste des protocoles et abstractions gérés par votre configuration. La fonction `stream_get_wrappers()` retourne un tableau avec la liste des fonctions d'abstraction d'accès aux fichiers (fichiers locaux, compressés, par protocole HTTP, etc.). La fonction `stream_get_transports()` donne la liste des transports réseaux gérés (`tcp`, `udp`, `ssl`, etc.). En mettant ces deux résultats bout à bout, vous obtiendrez la liste de tous les préfixes utilisables dans les fonctions de gestion de flux. Un exemple des types qui peuvent être pré-enregistrés vous est donné dans la figure 14-6.

```
<?php
$fp = stream_get_wrappers();
print_r($fp);
$fp = stream_get_transports();
print_r($fp);
?>
```

**Figure 14-6**

*Flux et transports gérés*



### Utilisation simple

L'utilisation des abstractions de flux est très similaire à celle décrite dans la gestion des fichiers. Les fonctions en jeu sont globalement les mêmes.

#### Ouverture

Pour ouvrir un flux quelconque, il suffit de faire appel à la fonction `fopen()` comme décrit au début de ce chapitre. Pour utiliser autre chose qu'un fichier local, il vous suffit de spécifier le préfixe associé au type de flux que vous souhaitez.

```
$fp = fopen('/etc/passwd', 'r') ;
$fp = fopen('http://www.php.net/', 'r') ;
```

Dans le cas d'une *socket* réseau utilisée en mode client, vous pouvez aussi utiliser `stream_socket_client()`. Cette fonction s'utilise de la même façon que la fonction `fsockopen()` décrite plus haut, mais le port souhaité se spécifie dans l'adresse au lieu d'un paramètre à part.

```
■ $fp = stream_socket_client('tcp://192.168.0.5:80') ;
```

Vous pouvez aussi vous reporter aux descriptions sur le lancement de programmes externes et les fonctions `popen()` ou `proc_open()` pour gérer les flux de programmes externes.

### Lecture et écriture

Une fois le descripteur ouvert, vous pouvez lire et écrire avec toutes les fonctions décrites dans la gestion des fichiers : `fgetc()`, `fgets()`, `fread()`, `fwrite()`, `feof()`, etc. Il existe toutefois quelques fonctions supplémentaires.

La fonction `stream_get_line()` permet de lire une ligne sur le descripteur, comme `fgets()`. Elle prend tout de même un argument supplémentaire, qui permet de spécifier la chaîne qui sépare les lignes entre elles. De plus, à la différence de `fgets()`, le délimiteur lui-même n'est pas retourné avec la chaîne de caractères résultat.

```
■ $chaîne = stream_get_line($fp, 1024, "\r\n") ;
```

La fonction `stream_copy_to_stream()` permet de connecter deux flux. PHP lira le premier descripteur donné en argument pour écrire dans le second. Vous pouvez optionnellement définir en troisième argument un nombre maximal de caractères à lire (et donc à copier). Le nombre de caractères lus et écrits vous sera retourné.

### Fonctions de contrôle

#### Gestion de tampon

La fonction `stream_set_write_buffer()` permet de définir la taille du tampon sur un flux particulier. Le descripteur de flux est à fournir en premier paramètre et la nouvelle taille en second paramètre. La taille par défaut est de 8 ko.

Pour forcer le tampon à être vidé, on peut utiliser la fonction `fflush()` avec le descripteur de flux comme unique argument.

#### Métadonnées

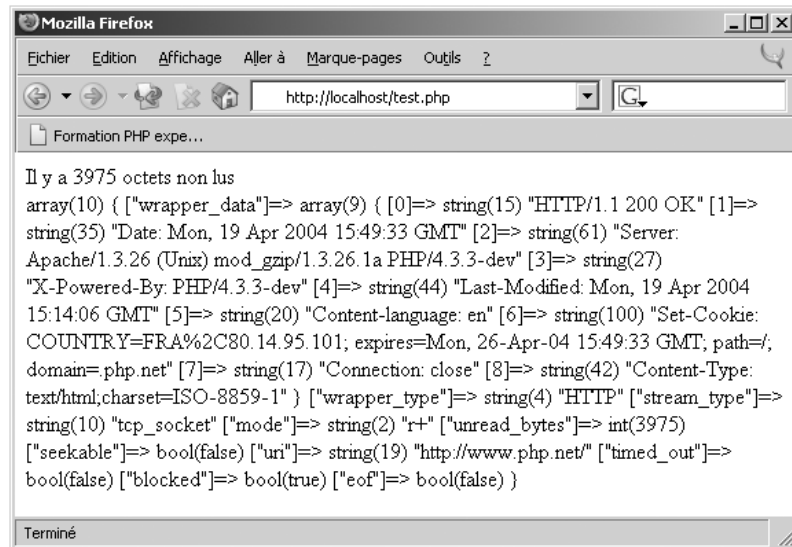
Les métadonnées sont les données annexes à un contenu, par exemple le nombre de caractères restant à lire. La fonction `stream_get_meta_data()` permet de récupérer ces informations en fournissant un descripteur de flux comme unique argument. Les métadonnées sont retournées dans un tableau associatif qui contient les mêmes informations que ce qui a été décrit plus haut pour `socket_get_status()` :

La clé `timed_out` est un booléen qui est vrai s'il s'agit d'une *socket* réseau qui a expiré (le serveur ne répond plus et la connexion a été coupée). La clé `blocked` est un booléen qui est vrai si la connexion est en mode bloquant. La clé `eof` est un booléen vrai quand le flux a fini de recevoir des données (la *socket* est close ou le fichier a été entièrement lu).

Enfin, la clé `unread_bytes` renvoie le nombre de caractères qui restent à lire. Un aperçu des différents paramètres est donné à la figure 14-7

```
<?php
$fp = fopen('http://www.php.net/', 'r') ;
$info = stream_get_meta_data($fp) ;
echo 'Il y a ', $info['unread_bytes'], ' octets non lus <br>' ;
var_dump($info);
?>
```

Figure 14-7  
Métadonnées



#### Note

Même si la *socket* a fini d'envoyer des données (la clé `eof` renvoie vrai), il peut encore y avoir des données en attente de lecture. Pour tester si on a lu toutes les données à lire, il faut utiliser la fonction `feof()` décrite précédemment.

Des informations propres aux fonctions d'abstraction sont toutefois disponibles. Les clés `stream_type` et `wrapper_type` renvoient les types de flux et d'abstraction utilisés (fichier compressé, *socket* TCP, etc.). La clé `wrapper_data` contient les données spécifiques à l'abstraction utilisée (par exemple les en-têtes dans le cas d'une connexion HTTP). Enfin, la clé `filters` contient la liste des filtres actifs sur le flux.

#### Temps d'expiration

Le temps d'expiration permet de couper la communication sur une *socket* quand l'ordinateur d'en face ne répond plus. Classiquement, le temps d'expiration par défaut est de 30 secondes. La fonction `stream_set_timeout()` permet de définir un temps d'expiration

différent. Elle prend un descripteur de flux comme premier argument et un temps en secondes comme second argument.

### Mode bloquant

Quand un flux est défini comme bloquant et qu'on demande une lecture, le script est mis en attente tant que toutes les données demandées ne sont pas disponibles et que le flux n'est pas terminé. En mode non bloquant, la fonction renvoie ce qui est disponible (potentiellement une chaîne vide) sans attendre.

Vous pouvez basculer d'un mode à l'autre en fournissant un descripteur de flux et un booléen à la fonction `stream_set_blocking()`. Si le booléen est vrai, le flux sera en mode bloquant, sinon il sera en mode non bloquant.

### Fermeture

Tout flux ouvert est fermé à l'aide d'une unique fonction : `fclose()`. Elle prend comme unique paramètre le descripteur du flux à fermer.

À la fin du script, PHP ferme tous les flux ouverts. Laisser des connexions ouvertes mais non utilisées consomme toutefois des ressources inutilement et est parfois considéré comme de la mauvaise programmation. Nous vous conseillons de fermer explicitement toutes les ressources ouvertes dès qu'elles ne sont plus utilisées.

### Fonctions réseau

En plus des fonctions générales, les fonctions réseaux donnent accès à quelques fonctions spécifiques.

#### Nom de la socket

La fonction `stream_socket_get_name()` permet de récupérer l'adresse associée à une *socket* réseau. Elle prend en arguments un descripteur de flux et un booléen. Si le booléen est vrai, c'est l'adresse de la *socket* distante qui sera retournée, sinon c'est l'adresse de la *socket* locale qui sera lue.

#### Serveur réseau

Jusqu'à présent, nous n'avons utilisé les *sockets* réseaux à l'aide de `fsockopen()` ou `stream_socket_client()` que pour nous connecter à un serveur existant, pas pour agir en serveur nous-mêmes. Créer un serveur réseau est pourtant possible. Des implémentations de serveurs FTP et HTTP faites en PHP ont même été écrites avec la version 4.

Vous pouvez commencer à écouter sur une *socket* à l'aide de la fonction `stream_socket_server()` en fournissant l'adresse d'écoute en argument. Elle renvoie un identifiant de serveur ou `FALSE` en cas d'erreur. L'exemple suivant crée un serveur qui écoute sur le port HTTP.

Si vous fournissez deux variables comme deuxième et troisième arguments, ces variables contiendront respectivement un code d'erreur et un texte explicatif si le port n'a pas pu être alloué.

```
$serveur = stream_socket_server('tcp://0.0.0.0:80', $no, $txt) ;  
if (! $serveur ) die( "Erreur $no : $txt" ) ;
```

**Note**

Sur la plupart des systèmes, vous (et votre processus PHP) devrez avoir les droits d'administrateur pour créer un serveur sur un port inférieur à 1 024.

Une fois le serveur créé, il nous reste à attendre les connexions. La fonction `stream_socket_accept()` attend une connexion et retourne le descripteur correspondant. Elle prend en paramètre l'identifiant du serveur sur lequel écouter et, optionnellement, un temps d'expiration en secondes (sinon PHP utilisera la valeur par défaut). En fournissant une variable en troisième paramètre, l'adresse de la *socket* distante sera retournée en cas de connexion réussie. Si aucune nouvelle connexion n'était en attente ou n'arrive dans le temps imparti, la fonction renvoie la valeur `FALSE`.

```
$serveur = stream_socket_server('tcp://0.0.0.0:80', $no, $txt) ;  
$fp = stream_socket_accept($serveur, 10) ;  
if (! $fp ) echo "Aucune connexion établie" ;
```

**Autres fonctions**

Les fonctions de gestion de fichiers ou de flux utilisent pour la plupart les abstractions de flux en interne. Les seules contraintes sont celles de l'abstraction utilisée. Ainsi, il est impossible d'utiliser la fonction `file_put_contents()` avec une adresse HTTP parce que l'abstraction correspondante n'accepte que la lecture et pas l'écriture.

Il est par exemple possible d'utiliser `copy()` pour copier un fichier d'un serveur HTTP vers un serveur FTP.

**Contextes**

Les contextes sont des options qui accompagnent un flux. Ils permettent de définir des paramètres pour la connexion. La liste des options de contexte de chaque type d'abstraction est donnée dans la description de chaque type de flux faite plus haut. Pour le protocole HTTP, on peut par exemple envoyer des en-têtes arbitraires dans la requête.

Ces contextes s'utilisent avec les différentes fonctions d'ouverture de flux et les fonctions d'accès rapide. Il faut d'abord créer un contexte, remplir les paramètres, puis le fournir comme dernier paramètre des fonctions de flux.

**Création d'un contexte**

Vous pouvez créer un contexte avec la fonction `stream_context_create()`. Elle prend en argument un tableau avec les options de contexte et renvoie un identifiant de contexte.

L'argument fourni est un tableau associatif à deux dimensions. La clé de première dimension est le nom de l'abstraction utilisée et la clé de deuxième dimension est le nom de

l'option à définir. Par exemple, pour qu'une opération FTP puisse écraser les fichiers déjà existants sur le serveur :

```
$options = array(
    'ftp' => array( 'overwrite' => TRUE ) ;
) ;
$contexte = stream_context_create( $options ) ;
```

### Modification d'un contexte

Vous pouvez modifier un contexte existant à l'aide de la fonction `stream_context_set_option()`. Elle prend en paramètres l'identifiant de flux ou de contexte (au choix), le nom de l'abstraction utilisée, le nom de l'option à modifier et en dernier la valeur à définir. Un booléen vrai est retourné en cas de réussite, un faux en cas d'échec.

```
stream_context_set_option($contexte, 'FTP', 'overwrite', TRUE);
```

### Lecture d'un contexte

Vous pouvez lire les options définies dans un contexte avec la fonction `stream_context_get_options()`. Elle prend en paramètre un identifiant de contexte ou un descripteur de flux et retourne le tableau associatif contenant les diverses options définies.

```
$options = stream_context_get_options( $contexte ) ;
$options = stream_context_get_options( $descripteur ) ;
```

### Ouvertures de flux

Les contextes doivent être fournis à l'ouverture des flux. Pour cela, il vous faut fournir l'identifiant de contexte en dernier paramètre des fonctions `fopen()`, `stream_socket_client()` et `stream_socket_server()`.

```
<?php
// Adresse du fichier FTP à écrire
$adr = 'ftp://ftp.monserveur.com/index.php' ;
// Mode d'ouverture : écriture
$mode = 'w' ;
// Utilisation de la directive de configuration include_path
$include_path = FALSE ;
// Contexte pour écraser les fichiers existants
$options = array(
    'ftp' => array( 'overwrite' => TRUE )
) ;
$contexte = stream_context_create( $options ) ;
// Ouverture
$fp = fopen($adr, $mode, $include_path, $contexte ) ;
?>
```

Les deux fonctions d'ouverture de *socket* utilisent juste avant le contexte un paramètre que nous n'avons pas encore décrit. Il permet de spécifier des paramètres de gestion des *sockets* (drapeaux de connexion). Sauf en de rares cas, vous devriez utiliser la valeur 0 pour une *socket* cliente :

```
// Adresse sur laquelle se connecter
$adr = 'ssl://www.php.net:443';
// Temps maximal d'établissement de la connexion
$timeout = 10 ;
// Paramètre de connexion
$flag = 0 ;
// Contexte
$options = array(
    'ssl' => array('verify_peer' => FALSE )
);
$contexte = stream_context_create( $options );
// Ouverture
$fp = stream_socket_client( $adr, $code_erreur, $texte_erreur,
    $timeout, $flag, $contexte );
```

Pour une *socket* serveur, vous utiliserez l'association de constantes `STREAM_SERVER_BIND` | `STREAM_SERVER_LISTEN` :

```
// Adresse sur laquelle se connecter
$adr = "ssl://127.0.0.1:443" ;
// Temps maximal d'établissement de la connexion
$timeout = 10 ;
// Paramètre de connexion
$flag = STREAM_SERVER_BIND | STREAM_SERVER_LISTEN ;
// Contexte
$options = array(
    'ssl' => array( 'verify_peer' => FALSE )
);
$contexte = stream_context_create( $options );
// Ouverture
$fp = stream_socket_client( $adr, $code_erreur, $texte_erreur,
    $timeout, $flag, $contexte );
```

### Fonctions d'accès rapide

Les fonctions d'accès rapide telles que `readfile()`, `file_get_contents()`, `file_set_contents()` et `file()` peuvent de la même façon recevoir les options de contexte. Il vous suffit d'ajouter l'identifiant de contexte comme dernier paramètre de ces fonctions.

## Filtres

Les filtres permettent d'appliquer automatiquement des fonctions sur les données reçues ou envoyées à travers un flux. Il est possible d'enchaîner autant de filtres que vous souhaitez sur un même flux.



Pour exemple, si on applique la fonction `strtolower()` (qui convertit un texte en minuscules) sur un flux pour le mode écriture, toute donnée écrite sur le flux sera automatiquement convertie en minuscule.

### Utilisation

Pour ajouter un filtre sur un flux, il vous suffit de faire appel aux fonctions `stream_filter_append()` et `stream_filter_prepend()` en fournissant en arguments le descripteur de flux et le nom d'un filtre existant. Si c'est la première fonction qui est choisie, le filtre est ajouté à la fin de la liste des filtres actuels (il sera appliqué en dernier). Si c'est la seconde fonction qui est utilisée, le filtre sera ajouté en début de liste (il sera appliqué en premier).

En fournissant optionnellement un troisième paramètre, vous pouvez définir le mode pour lequel vous souhaitez utiliser le filtre. Le filtre sera utilisé en écriture si vous fournissez la constante `STREAM_FILTER_WRITE`, en lecture si vous fournissez la constante `STREAM_FILTER_READ`, et pour les deux avec `STREAM_FILTER_BOTH`.

### Fonctions d'accès rapide

Les fonctions d'accès rapide comme `file_get_contents()` ne renvoient aucun descripteur de flux à l'utilisateur. Vous pouvez tout de même y appliquer des filtres grâce à une syntaxe spéciale.

L'abstraction utilisée est `php://filter`. Elle accepte deux paramètres : l'adresse du flux à ouvrir et la liste des filtres à utiliser.

Le paramètre nommé `resource` désigne l'adresse du flux à ouvrir. Ainsi, pour ouvrir la page d'accueil du site officiel de PHP, on pourrait utiliser l'adresse suivante :

```
■ php://filter/resource=http://www.php.net/
```

Les paramètres `read` et `write` permettent de spécifier une liste des filtres à utiliser en lecture et en écriture. L'adresse suivante ouvre toujours la page d'accueil de `www.php.net`, mais applique automatiquement les filtres `string.rot13` et `string.toupper` en lecture :

```
$ressource = 'http://www.php.net/' ;
$read = array( 'string.rot13', 'string.toupper' ) ;

$adresse = 'php://filter'
          . '/read='.implode('|', $read)
          . "/resource=$ressource" ;
```

Vous pourrez utiliser cette syntaxe dans toutes les fonctions d'accès rapide décrites plus haut dans ce chapitre.

### Liste des filtres définis

Par défaut, certains filtres sont prédéfinis. On peut y trouver `string.tolower` et `string.toupper` (qui convertissent respectivement les caractères en minuscules et majuscules), `string.rot13` (codage *rot13* : le N remplace le A, le O remplace le B, le P

remplace le C, et ainsi de suite), `string.base64` (équivalent de `base64_decode()` et `base64_encode()`) et `string.quoted_printable` (codage principalement utilisé dans les e-mails).

### Ajouter un filtre personnalisé

La liste des filtres prédéfinis est très réduite ; elle est là plutôt à titre d'exemple. Il est possible de l'étendre à volonté grâce à la fonction `stream_filter_register()`. Cette dernière fonction prend deux paramètres : le nom du filtre à créer et le nom d'une classe implémentant les fonctionnalités du filtre.

```
stream_filter_register('monfiltre', 'maclassedefiltre');
```

La classe doit être une dérivée de la classe `php_user_filter`. Elle doit implémenter obligatoirement trois méthodes : `filter()`, `onclose()` et `oncreate()`.

```
class maclassedefiltre extends php_user_filter {  
  
    function filter($in, $out, &$consomme, $ferme) {  
        /* conversion */  
    }  
    function onclose() {  
        /* Libération des ressources à la fermeture */  
    }  
    function oncreate() {  
        /* Initialisations à l'ouverture */  
    }  
}
```

La première méthode, `filter()`, sert à faire les conversions. Les deux premiers arguments fournis sont deux ressources ; la première permet de lire la chaîne à filtrer et la seconde permet d'écrire le résultat du filtre (la chaîne convertie). Le troisième paramètre est un entier passé par référence qui représente le nombre de caractères lus sur le flux ; vous devez l'incrémenter du nombre de caractères que vous avez traités en entrée. Si vous traitez les caractères par deux et qu'il y en ait trois reçus vous pouvez donc ne traiter que les deux premiers et n'incrémenter la variable que de 2. Le quatrième et dernier argument est un booléen : il sera vrai quand le flux sera sur le point de se fermer, c'est-à-dire que PHP est en train de faire son dernier appel au filtre.

La fonction doit retourner un entier qui correspond à la réussite de l'opération. La constante `PSFS_PASS_ON` indique que les données ont été traitées avec succès, `PSFS_FEED_ME` indique que rien n'a été retourné et que le filtre attend des données supplémentaires avant d'agir, enfin `PSFS_ERR_FATAL` indique une erreur fatale dans le processus et que le filtre ne peut continuer.

Vous pouvez utiliser le modèle suivant pour gérer les deux ressources fournies en paramètres :

```
function filter($in, $out, &$consomme, $ferme) {  
  
    // Lit une chaîne de caractères sur l'entrée et récupère un objet  
    while ($donnee = stream_bucket_make_writeable($in)) {
```

```
// Convertit la chaîne lue ($donnee->data)
$donnee->data = strtoupper($donnee->data) ;

// Ajoute la chaîne convertie à la sortie
stream_bucket_append($out, $donnee) ;

// Incrémente le nombre de caractères lus ($donnee->datalen)
$consomme += $donnee->datalen ;

}

// Valeur de retour indiquant que les données
// ont été traitées avec succès
return PSFS_PASS_ON;

}
```

Les fonctions `oncreate()` et `onclose()` permettent de faire les initialisations nécessaires et de libérer les éventuelles ressources utilisées par le filtre. Elles sont appelées respectivement avant l'utilisation du filtre et juste après (une fois le tampon d'écriture vidé).

## Types personnalisés

PHP fournit par défaut les abstractions les plus utiles dans le domaine web : HTTP, FTP, SSL, etc. Si toutefois la quantité d'abstractions gérées n'était pas suffisante pour vos besoins, il est possible d'en créer de nouvelles. La procédure est similaire à la création de filtres personnalisés : il vous faut créer une classe implémentant un certain nombre de méthodes définies, puis l'enregistrer à l'aide de `stream_wrapper_register()`.

Cette dernière fonction prend en arguments un nom de protocole (la partie avant le caractère deux points) et un nom de classe. Elle renvoie un booléen qui sera vrai en cas de réussite et faux si une abstraction de ce nom existe déjà.

La classe à créer doit implémenter les fonctions suivantes : `stream_open()`, `stream_read()`, `stream_write()`, `stream_tell()`, `stream_seek()`, `stream_stat()` et `stream_eof()`.

La méthode `stream_open()` doit accepter quatre arguments. Les deux premiers sont l'adresse du flux et le mode d'ouverture. Il est important de noter qu'il vous appartient de vérifier que le flux est effectivement accessible par le mode demandé. Vous devrez renvoyer un booléen à vrai en cas d'ouverture réussie et à faux en cas d'échec.

Le troisième paramètre est un entier qui correspond aux constantes `STREAM_USE_PATH`, `STREAM_REPORT_ERRORS`, ou à la composition des deux. Si `STREAM_USE_PATH` est présente et que l'adresse soit une adresse relative, vous pouvez utiliser la directive de configuration `include_path` pour chercher la ressource demandée. Si `STREAM_REPORT_ERRORS` est définie, vous devrez lancer vous-même des erreurs à l'aide de `trigger_error()` en cas de problèmes. Dans le cas contraire, vous ne devrez lancer aucune erreur et laisser PHP le faire.

Le quatrième argument est une variable passée par référence. Si le troisième paramètre est composé avec `STREAM_USE_PATH` et si une adresse relative a été fournie, vous devrez y affecter l'adresse réelle de la ressource utilisée.

Les autres méthodes à implémenter correspondent à leur équivalent en gestion de fichiers. `stream_read()` lit une chaîne de caractères et prend un nombre de caractères à lire en entier. `stream_write()` écrit la chaîne de caractères reçue en argument vers le flux. `stream_eof()`, `stream_tell()`, `stream_seek()`, `stream_stat()` et `stream_close()` fonctionnent exactement comme `feof()`, `ftell()`, `fseek()`, `fstat()` et `fclose()`, mais sans le premier paramètre (qui est le descripteur dans les fonctions de gestion de fichier).

```
class monabstraction {
    function stream_open($adresse, $mode, $options, &$adr_absolue) {
        /* Ouverture du flux */
    }
    function stream_close() {
        /* Fermeture du flux */
    }
    function stream_read($nombre) {
        /* Lit le flux pour $nombre caracteres */
    }
    function stream_writer($donnee) {
        /* Écrit la donnée dans le flux */
    }
    function stream_eof() {
        /* Renvoie vrai si toutes les données sont lues */
    }
    function stream_tell() {
        /* Renvoie la position actuelle dans le flux */
    }
    function stream_seek($position, $type_de_position) {
        /* Change de position dans le flux */
    }

    function stream_stat() {
        /* Renvoie les informations sur le fichier */
    }
}

stream_wrapper_register('abs', 'monabstraction');

$fp = fopen('abs://monadresse', 'r');
```

Des méthodes pour le traitement des fichiers et répertoires eux-mêmes existent aussi. Les méthodes `rmdir()`, `mkdir()`, `unlink()`, `rename()`, `dir_opendir()`, `dir_readdir()`, `dir_rewinddir()`, `dir_closedir()`, `stream_lock()`, `url_stat()` fonctionnent comme leur équivalent en fonction prédéfinie.

## Cas d'application

### *Système de paiement en ligne*

#### Contexte

Vous utilisez actuellement pour votre site de commerce un service de paiement en ligne classique. Une fois la commande réalisée, vous redirigez le client vers une page de votre banque fournisseur. Le client remplit les informations de paiement sur le site de la banque et ce dernier vous renvoie le résultat en fin de procédure.

Cette situation vous convient à peu près pour votre site web, mais vous êtes en train de faire une application de prise de commande par téléphone. Bien qu'il soit possible de rediriger les standardistes vers le site web de la banque pour noter les informations du client, cela s'avère peu pratique : vous ne pouvez plus garder une cohérence dans l'interface (par exemple pour avoir des raccourcis clavier qui passent d'une étape à l'autre), et surtout cette procédure vous impose de demander les informations de paiement du client à la fin de la commande (ou de les noter dans un coin pour les recopier à la main par la suite). Dans la pratique, il est dur d'imposer une telle contrainte au client ; beaucoup interviennent au dernier moment pour changer un détail ou un autre, ce qui ne serait pas possible une fois la redirection vers la banque faite.

#### Réalisation

Vous avez donc décidé d'utiliser un procédé plus avancé qui vous permet de valider et d'effectuer un paiement simplement sans avoir à passer par l'interface web de la banque. Votre contact bancaire vous a fourni un programme qui se connecte tout seul à la banque de manière sécurisée. Pour effectuer les paiements, il vous faudra l'exécuter en donnant en paramètres les informations comme le numéro de carte de crédit et la date d'expiration.

La première partie de votre script va définir quatre constantes. La première est le chemin d'accès vers le programme de la banque. Le spécifier ainsi en début de fichier vous permettra de le modifier facilement par la suite. Les trois autres constantes définissent le résultat du paiement. On se contente de faire la différence entre un paiement réussi, un paiement qui a échoué à cause des informations fournies par le client et une erreur interne (problème de connexion à la banque par exemple).

```
<?php
class paiement {
    const BANQUE = "/usr/local/banque/programme" ;
    const PAY_REUSSI = 1 ;
    const PB_UTILISATEUR = 2 ;
    const PB_INTERNE = 4 ;
}
```

Les méthodes suivantes vont nous permettre de définir les informations client (numéro de carte de crédit, date d'expiration, etc.) :

```
<?php
public function setNumeroCarte($numero) {
    $this->numero = $numero ;
}
```

```
    }  
    public function setExpiration($annee, $mois) {  
        $this->annee = $annee ;  
        $this->mois = $mois ;  
    }  
}
```

Enfin, nous faisons une fonction qui déclenche le paiement vers la banque. Pour cela, nous utilisons les outils d'exécution de programme.

```
public lancePaiement() {  
    // Préparation de la commande à lancer  
    $commande = self::BANQUE ;  
    $commande .= ' '.escapeshellarg($this->numero) ;  
    $commande .= ' '.escapeshellarg($this->annee) ;  
    $commande .= ' '.escapeshellarg($this->mois) ;  
  
    // Lancement de la commande  
    $banque = popen($commande, "r")  
    if ( !$banque) {  
        // Exécution impossible, problème interne  
        return self::PB_INTERNE ;  
    }  
  
    // Lecture du message renvoyé par la banque  
    $resultat = fread($banque, 2) ;  
    $message = '' ;  
    while ( !feof($banque) ) $message = fread($banque, 1024) ;  
    $retour = pclose($banque) ;  
  
    // Résultat de la transaction  
    if (!$retour) {  
        // Il y a eu un problème lors de l'exécution  
        // ou lors de la connexion  
        return self::PB_INTERNE ;  
    } elseif($resultat == 'OK') {  
        // Tout s'est bien passé,  
        // le message est le code d'autorisation de la banque  
        $this->autorisation = $message ;  
    } else {  
        // Les informations de paiement sont erronées  
        // Le message fourni est une explication de la banque  
        $this->erreur = $message ;  
    }  
}
```

Voici maintenant le code de la classe de paiement en entier, et la façon de l'utiliser :

```
<?php  
class paiement {  
    const BANQUE = "/usr/local/banque/programme" ;  
    const PAY_REUSSI = 1 ;
```

```
const PB_UTILISATEUR = 2 ;
const PB_INTERNE = 4 ;

public function setNumeroCarte($numero) {
    $this->numero = $numero ;
}

public function setExpiration($annee, $mois) {
    $this->annee = $annee ;
    $this->mois = $mois ;
}

public lancePaiement() {
    // Préparation de la commande à lancer
    $commande = self::BANQUE ;
    // On échappe tous les caractères spéciaux
    // pour éviter des problèmes de sécurité
    // si un des standardistes n'est pas de confiance
    $commande .= ' '.escapeshellarg($this->numero) ;
    $commande .= ' '.escapeshellarg($this->annee) ;
    $commande .= ' '.escapeshellarg($this->mois) ;

    // Lancement de la commande
    $banque = popen($commande, "r")
    if ( !$banque) {
        // Exécution impossible, problème interne
        return self::PB_INTERNE ;
    }

    // Lecture du message renvoyé par la banque
    $resultat = fread($banque, 2) ;
    $message = '' ;
    while ( !feof($banque) ) $message = fread($banque, 1024) ;
    $retour = pclose($banque) ;

    // Résultat de la transaction
    if (!$retour) {
        // Il y a eu un problème lors de l'exécution
        // ou lors de la connexion
        return self::PB_INTERNE ;
    } elseif($resultat == 'OK') {
        // Tout s'est bien passé,
        // le message est le code d'autorisation de la banque
        $this->autorisation = $message ;
        return self::PAY_REUSSI ;
    } else {
        // les informations de paiement sont erronées
        // le message fourni est une explication de la banque
        $this->erreur = $message ;
        return self::PB_UTILISATEUR ;
    }
}
```

```
public function getMessageErreur() {
    return $this->erreur ;
}
public function getAutorisationPaiement() {
    return $this->autorisation ;
}
}

$p = new paiement() ;
$p->setNumeroCarte('1245346978') ;
$p->setExpiration('2004', '06') ;
$resultat = $p->paiement() ;
if ($resultat == paiement::PAY_REUSSI) {
    echo "Tout s'est bien passé, l'autorisation est "
        . $p->getautorisationPaiement() ;
} elseif($resultat == paiement::PB_INTERNE) {
    echo "Il y a eu un problème interne, réessayez plus tard" ;
} else { // $resultat == paiement::PB_UTILISATEUR
    echo "Les informations fournies sont refusées : "
        . $p->getMessageErreur() ;
}
}
```

## *Sauvegardes automatiques pour interface réseau*

### Contexte

Il y a une semaine, le disque dur d'une secrétaire a subi une panne. Il a été remplacé, mais toutes les données ont été perdues. Le disque réseau mis en place pour les sauvegardes n'est pas assez utilisé par les services internes et tout n'a pas pu être récupéré.

Comprenant qu'il est difficile d'obtenir des utilisateurs qu'ils se contraignent à archiver leurs fichiers de manière centralisée, vous avez décidé d'effectuer des sauvegardes des dossiers personnels de tous les postes chaque nuit.

Comme les administrateurs ne peuvent pas physiquement être là pour rallumer manuellement tous les postes pendant la sauvegarde automatique, juste avant l'ouverture des bureaux, le service informatique a eu la tâche de développer un outil dédié.

### Réalisation

La solution retenue a été d'utiliser le Wake On LAN. Il s'agit d'une fonctionnalité de certaines cartes réseau qui permet d'allumer le micro-ordinateur à distance. Il faut pour cela lui envoyer par réseau certaines informations.

Comme les machines sont éteintes, elles n'ont pas d'adresses IP. Il faut donc les référencer par leur adresse MAC. Il s'agit d'un identifiant unique que possède chaque carte réseau. Le tableau suivant établit la liste des adresses MAC des postes à sauvegarder :

```
$mac = array(
    '00:07:1D:22:A1:05' ,
```



```
'00:12:AB:32:50:05' ,  
'00:07:1D:23:FB:73' ,  
'00:07:1D:22:18:C2' ,  
);
```

La fonction suivante permet de convertir une adresse MAC sous sa représentation humaine vers sa représentation en octets :

```
function convertMac($adresse) {  
    $adresse_mac = explode(':', $adresse_mac);  
    $octets = '';  
    foreach($adresse_mac as $a) {  
        $octets .= chr(hexdec($a));  
    }  
    return $octets ;  
}
```

Il reste maintenant à envoyer sur le réseau l'information nécessaire pour réveiller ces différents postes. Il existe plusieurs manières de le faire ; nous utiliserons un paquet UDP que nous enverrons à tout le réseau (mais seules les cartes visées par le contenu du paquet déclencheront l'éveil).

```
function reveil($adresse_mac) {  
    // Conversion de l'adresse MAC vers sa forme réelle  
    $adresse_mac = convertMac($adresse_mac) ;  
    // Construction du message de réveil :  
    $msg = '' ;  
    for($i = 0 ; $i < 6 ; $i++) {  
        $msg .= chr(255) ;  
    }  
    for ($i = 0; $i < 16 ; $i++) {  
        $msg .= $adresse_mac ;  
    }  
    // Ouverture de la connexion (udp en broadcast, port 9)  
    $cx = stream_socket_client("udp://255.255.255.255:12287") ;  
    fwrite($cx, $msg) ;  
    fclose($cx) ;  
}
```

Une fois les postes réveillés, il suffit d'utiliser un partage de fichiers prédéfini pour recopier les fichiers sur le serveur de sauvegarde. En allumant les machines juste avant l'ouverture des bureaux, il n'y aura aucune dépense inutile. Dans le cas contraire, on pourrait penser à un programme qui éteint les micros à heure fixe (certains BIOS PC le font).

```
foreach($mac as $poste) {  
    reveil($poste) ;  
}
```

**Note**

L'implémentation faite ici n'est qu'un exemple pour vous montrer les possibilités des *sockets* ; elle n'est pas directement fonctionnelle et nécessitera des adaptations selon votre matériel.

## Conversion entre jeux de caractères

### Contexte

Vous développez un moteur d'indexation pour vos documents internes. Votre moteur fonctionnait parfaitement jusqu'à présent, mais avec l'avènement du XML, vous commencez à voir apparaître des fichiers avec le codage caractère UTF-8. Votre moteur ne gère que l'ISO-8859-1 et n'arrive donc pas à utiliser les termes accentués dans ces documents.

Le code source du moteur de recherche est assez complexe et contient de nombreuses optimisations pour les performances. Modifier ce code coûte cher et favorise l'apparition d'erreurs. Vous souhaitez donc une solution douce qui permette de modifier le minimum.

### Réalisation

Pour gérer ce problème de codage, vous avez décidé de mettre à profit les filtres de flux PHP. Tous les accès de votre moteur vers le disque d'archive ou les différents sites web seront filtrés par un petit script qui convertira d'abord les documents vers le bon codage.

Nous allons donc créer un nouveau type de protocole, qui sera utilisé par le moteur : `search://`. Il suffira d'ajouter ces neuf caractères dans la fonction du moteur qui fait appel à `fopen()` pour lire les fichiers. Les modifications sont donc extrêmement mineures.

```
class utf8_to_iso88591 {  
    }  
    stream_wrapper_register( 'search', 'utf8_to_iso88591' );
```

La fonction d'ouverture sera chargée de détecter le type de codage caractères utilisé par les fichiers. Pour être complet, on devrait vérifier les en-têtes HTTP dans le cas d'un fichier web, puis s'ils n'existent pas, vérifier la présence d'une déclaration XML ou de balise HTML `<meta>` dans le fichier, et finalement, si rien d'autre n'est présent, essayer de faire une détection basée sur la fréquence de certains caractères. Dans le cadre de cette application, on sait que les fichiers classiques sont tous en ISO-8859-1 et que seuls les fichiers XML risquent d'être en UTF-8.

```
private $contenu ;  
private $lu ;  
function stream_open($adresse, $mode, $options, &$adr_absolue) {  
    // On retire le préfixe « search: »  
    $url = substr($adresse, 9) ;
```

```

// Doit-on utiliser l'include_path ?
$use_path = $options & STREAM_USE_PATH ;
$this->contenu = @file_get_contents($url,$use_path) ;
$this->lu = 0 ;
// On vérifie si on a accédé correctement au fichier,
// sinon on rend la main
if ($this->contenu === FALSE) return FALSE ;
// On vérifie s'il s'agit d'un fichier XML
if (substr($url, -4) == '.xml') {
    // C'est un XML, on vérifie la déclaration XML
    $debut = strpos($this->contenu, '<?xml') ;
    if ($debut!==FALSE) {
        // On vérifie si c'est de l'UTF-8
        $fin = strpos($this->contenu, '?>', $debut+1) ;
        $declarationXML = substr($this->contenu, $debut, $fin) ;
        if ( !strpos($declarationXML, 'encoding="ISO-8859-1"')
            && !strpos($declarationXML, "encoding='ISO-8859-1'") ) {
            // C'est de l'utf-8, on transforme en iso-8859-1
            $this->contenu = utf8_decode($this->contenu) ;
        }
    }
}
return TRUE ;
}

```

Le reste des méthodes de la classe est alors très simple, il suffit de lire la propriété qui définit le contenu du fichier :

```

function stream_close() {
    // Rien à faire
}
function stream_read($nombre) {
    $texte = substr($this->contenu, $this->lu, $nombre) ;
    $this->lu += $nombre ;
    return $texte ;
}
function stream_writer($donnee) {
    // Pas pris en charge, le moteur de recherche ne fait que lire
}
function stream_eof() {
    return empty($this->contenu) ;
}
function stream_tell() {
    return $this->lu ;
}
function stream_seek($position, $type) {
    if ( $type == SEEK_CUR ) {
        $position += $this->lu ;
    } elseif( $type == SEEK_END ) {

```

```
        $position += strlen($this->contenu) ;
    }
    $this->lu = $position ;
}
function stream_stat() {
    /* Non implémenté */
}
```

Il y aurait eu d'autres moyens de procéder, notamment de remplacer directement la fonction `fopen()` du moteur de recherche plutôt que de créer une abstraction. L'avantage de cette méthode est de permettre un grand nombre d'évolutions par la suite. Si demain il faut pouvoir filtrer les mots-clés offensants ou permettre l'écriture par le moteur de recherche (par exemple, un moteur qui réécrit les tags `<meta>` des pages HTML), il y aura peu à modifier.



# 15

## Flux de sortie PHP

---

Nous avons vu au chapitre précédent comment gérer les différents flux d'un programme. PHP est lui-même un programme, avec un flux d'entrée et un flux de sortie. Le premier est géré en interne avec le serveur web, mais vous avez encore tout le contrôle sur le flux de sortie. Nous allons voir dans ce chapitre comment nous pouvons interagir avec ce flux de sortie. Nous verrons qu'il est possible de compresser ce flux pour envoyer une page moins lourde au navigateur et ainsi améliorer la rapidité de chargement. Nous verrons aussi qu'il est possible d'effectuer une conversion automatique du jeu de caractères envoyé en sortie, ce qui est pratique quand vous gérez une application avec plusieurs langues.

### Principes et utilisations

Gérer le flux de sortie de PHP permet d'appliquer des filtres sur le texte ou le résultat avant qu'il soit envoyé au navigateur voire d'en différer l'envoi.

#### *Principe de fonctionnement*

PHP permet de traiter son flux de sortie grâce à un principe de tampon. Au lieu d'envoyer directement ce qui est affiché au serveur web (et donc au navigateur), PHP le stocke en interne et l'envoie par petits blocs. Ce comportement évite à PHP de consommer trop de ressources en déclenchant un envoi pour quelques caractères seulement.

Les développeurs de PHP ont mis ce comportement à contribution pour permettre aux utilisateurs d'intervenir sur ce tampon. Il est ainsi possible de définir soi-même quand le contenu sera envoyé au serveur web ou d'appliquer des opérations sur le contenu du tampon avant qu'il soit envoyé.

## Exemples d'utilisation

### Compression des pages

L'utilisation la plus fréquente du tampon de sortie est de compresser le contenu des pages avant de l'envoyer au navigateur. Le protocole HTTP prévoit en effet de pouvoir compresser avec le format zip les fichiers avant de les envoyer.

La gestion de cette compression a deux attraits. Elle permet tout d'abord de réduire la taille des pages et donc de soulager la bande passante de votre serveur. La bande passante est souvent un des goulots d'étranglement des serveurs web et, en compressant les pages, vous pouvez en diminuer la consommation d'un facteur 2 à 10. Le deuxième avantage de la compression, c'est que la page est aussi plus rapide à télécharger (puisque de plus petite taille). Vous améliorez ainsi nettement le confort de visite des utilisateurs bas débit : ils n'auront plus désormais ce long délai d'attente pour afficher la page.

### Ajout d'en-tête ou cookie

La deuxième utilisation la plus fréquente de ce tampon consiste à différer l'envoi jusqu'à ce que le script soit complètement exécuté pour pouvoir ajouter des en-têtes et des cookies n'importe quand dans le script.

En effet, il est normalement impossible d'ajouter des en-têtes ou des cookies une fois que le contenu de la page a été envoyé (voir les chapitres 8 et 10 pour plus de détails sur les réponses HTTP et ce problème en particulier). En revanche, tant que PHP n'a pas envoyé de contenu au client, on peut définir ses propres en-têtes et cookies sans limitation.

En faisant garder en mémoire le contenu au lieu de l'envoyer directement, on peut alors modifier les en-têtes même si on a eu un affichage auparavant. Les en-têtes ne seront figés que lorsque PHP aura tout envoyé, c'est-à-dire à la fin du script.

### Conversion entre jeux de caractères

Pouvoir effectuer des actions sur le tampon avant son envoi permet d'effectuer de nombreuses conversions. Une des plus utiles dans un contexte international est probablement la conversion d'un jeu de caractères vers un autre, par exemple du national ISO-8859-1 vers l'international UTF-8. Pour plus de détails sur les codages de caractères, reportez-vous au chapitre 5, qui décrit les traitements de chaînes de caractères.

### Récupérer la sortie dans une variable

Une dernière utilisation courante du tampon de sortie est de récupérer dans une variable tout ce qui a été envoyé, pour la traiter plus tard. Ce comportement est principalement adopté quand on exécute un script externe : tout ce qu'il envoie est mis en tampon au lieu d'être passé au serveur web. Une fois l'exécution finie, on récupère le contenu du tampon dans une variable, comme si on avait fait une simple affectation.

## Gestion du tampon de sortie

### *Début et arrêt de la mise en tampon*

Pour initialiser un tampon de sortie, il faut demander à PHP de mettre tout ce qu'on envoie à l'affichage dans un coin de mémoire afin de pouvoir l'utiliser par la suite. C'est le rôle de la fonction `ob_start()`. Elle renvoie `TRUE` ou `FALSE` selon que PHP accepte ou non la mise en tampon.

Le tampon est automatiquement arrêté à la fin du script ; son contenu est alors directement envoyé vers la sortie. Il est toutefois possible de demander l'arrêt de la mise en tampon par avance. Pour cela, vous pouvez utiliser les fonctions `ob_end_flush()` et `ob_end_clean()`. La première fonction arrête le tampon et envoie le contenu au serveur web (et donc au navigateur du visiteur), la seconde vide le contenu, sans rien envoyer au visiteur.

```
<?php
ob_start() ;
// Demande que tout affichage soit mis en tampon
echo 'essai d\'affichage' ;
// Le texte est mis en tampon
// Le visiteur ne reçoit rien pour l'instant
setcookie('nom', 'valeur') ;
// On peut encore envoyer un cookie sans erreur
// parce que le serveur web n'a toujours envoyé aucun contenu
// Le bloc d'en-tête n'est toujours pas finalisé
ob_end_flush() ;
// PHP envoie alors le contenu du tampon au serveur web
// La mise en tampon est arrêtée
// Le client reçoit : essai d'affichage
// Le cookie est bien reçu aussi
echo 'après affichage' ;
// ce texte est envoyé directement, sans passer par le tampon
?>
```

Ces trois fonctions permettent uniquement de différer l'envoi. Elles sont utiles pour ceux qui ne peuvent pas gérer les cookies et les en-têtes uniquement en début de script. Elles ne permettent pas plus d'interactions.

### *Récupération du contenu*

Pour pouvoir agir sur le contenu, nous avons un jeu de trois autres fonctions : `ob_flush()`, `ob_get_contents()` et `ob_clean()`.

La première, `ob_flush()`, demande à PHP d'envoyer le contenu de son tampon au serveur web sans attendre. Contrairement à `ob_end_flush()`, ce qui est envoyé par la suite continue à être mis en tampon.

```
<?php
ob_start() ;
```



```
// Demande que tout affichage soit mis en tampon
echo 'essai d\'affichage' ;
// Le texte est mis en tampon
// Le visiteur ne reçoit rien pour l'instant
ob_flush() ;
// Le contenu est envoyé,
// Le client reçoit : essai d'affichage
setcookie('nom', 'valeur') ;
// Le cookie ne peut plus être envoyé
// car du contenu a déjà été fourni,
// le bloc d'en-tête est déjà finalisé
ob_end_flush() ;
// PHP envoie alors le contenu du tampon au serveur web
// Le seul contenu restant est le message d'erreur du cookie
?>
```

La fonction `ob_get_contents()` permet, elle, de récupérer le contenu du tampon au lieu de l'envoyer au serveur web. Attention, il ne s'agit que de lire le contenu du tampon, pas de le déplacer. Ce qui a été lu reste toujours en attente d'envoi vers le serveur web.

```
<?php
ob_start() ;
// Demande que tout affichage soit mis en tampon
echo 'essai d\'affichage' ;
// Le texte est mis en tampon
// Le visiteur ne reçoit rien pour l'instant
$aaffiche = ob_get_contents() ;
// $aaffiche contient maintenant : essai d'affichage
ob_end_flush() ;
// PHP envoie alors le contenu du tampon au serveur web
// La mise en tampon est arrêtée
// Le client reçoit : essai d'affichage
echo $aaffiche ;
// Le client reçoit une deuxième fois : essai d'affichage
?>
```

Pour effacer le contenu du tampon (par exemple après l'avoir lu et récupéré dans une variable) il faut faire appel à `ob_clean()`. Tout le contenu du tampon encore non envoyé est alors effacé, il ne pourra plus être ni récupéré ni utilisé.

```
<?php
ob_start() ;
// Demande que tout affichage soit mis en tampon
echo 'essai d\'affichage';
// Le texte est mis en tampon
// Le visiteur ne reçoit rien pour l'instant
ob_clean() ;
// Le tampon est effacé
// Il ne contient plus rien
ob_end_flush() ;
```

```
// PHP envoie alors le contenu du tampon au serveur web
// La mise en tampon est arrêtée
// Le client ne reçoit rien car le tampon était vide
?>
```

## Imbrication de tampons

PHP ouvre un tampon à chaque fois que vous faites appel à la fonction `ob_start()`. Il est alors possible d'imbriquer les gestions de tampon. Les fonctions de gestion s'appliqueront alors toujours au tampon de niveau supérieur (le dernier ouvert non refermé).

```
<?php
ob_start();
// On démarre la première mise en tampon
echo 'a';
// On envoie 'a' dans le premier tampon
ob_start();
// On démarre un deuxième tampon
echo 'b';
// On envoie 'b' dans le deuxième tampon
$deuxieme = ob_get_contents();
// On lit le contenu du deuxième tampon ("b")
// et on le copie dans une variable
ob_end_flush();
// On arrête le deuxième tampon
// Son contenu est envoyé vers la sortie,
// c'est-à-dire vers le tampon de niveau supérieur
// Le premier tampon contient maintenant 'ab'
echo 'c' ;
// On envoie 'c' dans le premier tampon
$premier = ob_get_contents();
// On copie le contenu du premier tampon dans une variable
ob_end_clean()
// On vide le premier tampon de son contenu et on l'arrête
echo $deuxieme , $premier ;
// Le client reçoit 'bab'
?>
```

## Informations sur le tampon

Les fonctions précédentes sont toutes « naïves ». Elles ne connaissent rien sur l'état du tampon ou sur son contenu. Il est parfois nécessaire de connaître un peu plus d'informations lors d'un script.

Deux fonctions nous permettent d'obtenir un peu plus de détails sur les tampons : `ob_get_length()` et `ob_get_level()`. La première retourne la taille en octets du contenu du dernier tampon ouvert. La seconde retourne le nombre de tampons encore ouverts, zéro si tous ont été refermés.

## Filtres automatiques

Pour l'instant, nous nous sommes contentés de traiter à la main le contenu du tampon. Pour y appliquer un filtre ou le transformer, il faudrait le récupérer avec `ob_get_contents()`, le modifier puis le renvoyer seulement une fois le tampon fermé. PHP permet toutefois une procédure plus automatique.

Les filtres les plus courants sont prédéfinis par PHP et directement utilisables. Vous pouvez lancer ainsi une conversion entre jeux de caractères ou une compression de pages à l'aide d'une seule commande, sans rien gérer à la main.

### Compression des pages avec `zlib`

Le protocole HTTP permet au serveur web d'envoyer une page compressée au navigateur pour économiser de la bande passante et accélérer les téléchargements. Cette compression est faite avec le module `zlib` (gestion de fichiers `zip`), il vous faut donc l'avoir activé pour pouvoir utiliser cette fonctionnalité (soit avoir compilé PHP avec `--with-zlib`, soit décommenter l'extension dans votre `php.ini` si vous avez une distribution toute faite comme sous Microsoft Windows).

PHP détecte tout seul si le client accepte ou non ce mode d'envoi et agit en conséquence (si ce n'est pas le cas, ce gestionnaire ne compressera pas le contenu). Pour démarrer ce gestionnaire de tampon, ajoutez simplement `ob_start('ob_gzhandler')` en début de script. `ob_gzhandler()` est une fonction interne au module `zlib`, qui permet de compresser une chaîne de caractères dans le cadre du protocole HTTP.

Il n'est pas nécessaire de vous préoccuper davantage de ce tampon une fois lancé ; il s'arrêtera seul à la fin du script. Pensez en revanche à le lancer avant tout affichage et avant toute autre mise en tampon.

```
<?php
ob_start('ob_gzhandler') ;
/* Vous pouvez maintenant écrire le reste
du script normalement*/
```

#### Note

La compression des pages n'est pas gratuite, elle consomme des ressources processeur. Vérifiez donc que vous pouvez vous permettre cette surconsommation de processeur avant d'activer la compression.

### Configuration

Il est possible de configurer certains paramètres pour la compression des pages via des directives du `php.ini`. La directive de configuration `zlib.output_compression_level` détermine le niveau de compression à utiliser. Il s'agit d'une valeur entre 0 et 9 (0 ne compressant pas et 9 étant la compression la plus forte). Les valeurs les plus importantes consomment des ressources processeur très importantes pour un gain minime par rapport aux

valeurs intermédiaires. Il est conseillé de laisser le réglage par défaut, ou de choisir une valeur entre 2 et 6.

#### Attention

Cette directive n'est pas toujours présente dans le `php.ini`. Il faut parfois l'écrire soi-même. Vérifiez qu'elle a bien été prise en compte avec un `phpinfo()` (voir figure 15-1).

## zlib

<b>ZLib Support</b>	enabled
<b>Compiled Version</b>	1.1.4
<b>Linked Version</b>	1.1.4

Directive	Local Value	Master Value
<code>zlib.output_compression</code>	On	On
<code>zlib.output_compression_level</code>	9	9
<code>zlib.output_handler</code>	<i>no value</i>	<i>no value</i>

Figure 15-1

Configuration

## Conversion entre jeux de caractères

Il existe deux filtres prédéfinis dans PHP, qui vous permettront de faire une conversion automatique d'un jeu de caractères vers un autre. Grâce à une simple ligne au début de vos scripts, vous pourrez changer de jeu de caractères sans modifier vos données ou la façon dont elles sont traitées. C'est particulièrement utile si votre application doit gérer plusieurs langues utilisant des alphabets différents.

Le premier filtre utilise le module `mbstring`. Il vous sera particulièrement utile si vous utilisez différents jeux de caractères ou des jeux de caractères internationaux (qui stockent un caractère sur plusieurs octets). Pour l'utiliser, vous devrez avoir activé le module `mbstring`.

Le second filtre se base sur le module `iconv`. Les possibilités d'`iconv` sont beaucoup moins importantes que celles de `mbstring`, puisque le module se contente de faire une conversion mais n'étend pas les fonctionnalités des fonctions PHP habituelles. Vous vous en servirez si vous n'utilisez que des jeux de caractères où un caractère se code sur un octet. Dans ce cas, c'est probablement la solution la plus avantageuse. Pour utiliser le filtre du module `iconv`, il faudra avoir activé ce module dans votre configuration.

Vous trouverez plus d'informations sur les deux modules mis en œuvre au chapitre 5, qui traite plus en détail des manipulations de chaînes de caractères.

## Utilisation du module mbstring

Pour utiliser la conversion de codage `mbstring`, vous devez avoir activé le module correspondant et faire appel à `ob_start('mb_output_handler')` en début de script.

```
<?php
ob_start('mb_output_handler') ;
// Vous pouvez maintenant envoyer écrire $$$ ??? $$$ le reste du script normalement
```

PHP utilise alors la directive de configuration `http_output` pour savoir vers quel codage effectuer la conversion (si la directive contient la valeur `pass`, alors aucune conversion n'est faite). Vous pouvez trouver cette directive, ainsi que les suivantes dans votre fichier de configuration `php.ini`. La conversion elle-même n'a lieu que si aucun codage de caractères n'a été spécifié dans l'en-tête `Content-Type` et s'il s'agit d'un fichier texte (`Content-Type` commençant par `text/`). Pour plus de facilité, vous pouvez aussi définir ce type de configuration via la fonction `ini_set()` ou directement via la fonction `mb_http_output()`.

## Utilisation du module iconv

De même que pour le module `mbstring`, il est possible de convertir les pages d'un jeu de caractères vers un autre avec le module `iconv`. Le fonctionnement est similaire : le nom de la fonction de rappel est `ob_iconv_handler`. Les jeux de caractères à utiliser sont à définir avec `iconv_set_encoding()` ou via les directives de configuration du `php.ini` (`iconv.internal_encoding` et `iconv.output_encoding`).

La fonction `iconv_set_encoding()` attend deux paramètres : le type de codage (`internal_encoding` pour le jeu utilisé en interne, `output_encoding` pour le jeu destination) et le nom du jeu de caractères (`ISO-8859-1` ou `UTF-8` par exemple). Les directives de configuration prennent, elles, directement le nom du jeu de caractères comme valeur.

```
<?php
// Exemple de changement de jeu français vers international (UTF-8)
iconv_set_encoding('internal_encoding', 'ISO-8859-1')
iconv_set_encoding('output_encoding', 'UTF-8') ;
ob_start('ob_iconv_handler') ;

/* Reste du script normalement */
```

### Note

Le gestionnaire `iconv` ne peut pas être activé en même temps que le gestionnaire `mbstring`.

## Filtres utilisateur

Les filtres prédéfinis de PHP couvrent la plupart des utilisations courantes. Si vous avez un besoin spécifique, il est toutefois possible de monter votre propre filtre automatique à l'aide d'une fonction de rappel.

Si vous fournissez un nom de fonction en paramètre à `ob_start()`, PHP lui fournira le contenu du tampon à chaque `ob_flush()` ou `ob_end_flush()` et enverra le résultat sur la sortie (ou le tampon parent).

```
<?php
function a2b( $avecA ) {
    $avecB = str_replace('a', 'b', $avecA) ;
    return $avecB ;
}
ob_start( 'a2b' ) ;
echo 'ababab' ;
ob_end_flush() ;
// Affiche 'bbbbbb' ;
?>
```

#### Note

Comme pour toutes les fonctions de rappel, il est aussi possible de fournir une méthode plutôt qu'une fonction. Il faut alors fournir un tableau de deux éléments : le premier est l'objet à utiliser (ou le nom de la classe s'il s'agit d'une méthode statique) et le second est le nom de la méthode.

## Automatisation

Il est possible d'automatiser l'activation de gestionnaires de tampon via le fichier de configuration `php.ini`.

Si vous spécifiez un nom de fonction pour la directive `output_buffering`, PHP lancera tout seul un `ob_start()` implicite en début de script avec la fonction en question comme filtre. Ce comportement est particulièrement utile si vous voulez appliquer un filtre à plusieurs scripts sans avoir à modifier toutes les sources.

Prêtez attention toutefois à bien définir cette fonction avant de faire un quelconque affichage, sinon PHP ne saura pas quoi exécuter et signalera une erreur. Les gestionnaires comme `ob_mb_handler`, `ob_iconv_handler` ou `ob_gzhandler` peuvent aussi être utilisés.

### Cas du gestionnaire zlib

Si vous utilisez la compression des pages avec le gestionnaire `gzhandler` du module `zlib`, il est aussi possible de l'automatiser via une directive du `php.ini` : `zlib.output_compression`. Si elle est activée, alors PHP démarrera seul la compression des pages sans avoir besoin qu'on lui fournisse `ob_start('ob_gzhandler')` à l'exécution. Ce comportement vous permet d'activer ou de désactiver la compression de tout un site sans toucher aux scripts. C'est la méthode conseillée pour utiliser la compression HTTP.

#### Attention

L'activation de la directive de configuration et le démarrage du tampon manuellement sont exclusifs l'un de l'autre. Si vous utilisez les deux, votre contenu sera compressé deux fois, ce qui le rendra illisible par les navigateurs (sans aucun gain de taille).

## Tampon interne de PHP

### *Délai avant affichage*

PHP utilise par défaut un petit tampon pour ses propres besoins, mais ne le laisse pas accessible directement. Il est impossible de le lire ou de le modifier.

Il est en revanche nécessaire de connaître cette information, car elle peut influencer sur divers comportements. Il est ainsi possible, si l'exécution d'une page dure longtemps, que le dernier caractère affiché sur le navigateur ne soit pas le dernier caractère que le script a envoyé à PHP. Le débogage peut parfois s'en trouver compliqué puisqu'on croit que le script s'arrête plus tôt que ce n'est effectivement le cas.

Par défaut, PHP utilise un tampon de 4 ko et envoie le contenu vers le serveur web quand il est plein ou que le script est fini. Ce comportement lui permet d'économiser des ressources en diminuant le nombre d'échanges avec le serveur web.

Vous pouvez modifier la taille par défaut du tampon avec la directive de configuration `output_buffering`. Si vous spécifiez une valeur numérique, PHP l'interprétera comme une taille en octets pour le tampon. Si vous spécifiez `0n`, PHP mettra tout le contenu de la page en tampon, vous permettant de modifier les en-têtes ou cookies à tout endroit de la page tant que vous n'utilisez pas la fonction `flush()` (voir plus bas pour plus de détails). Si vous spécifiez `0ff`, PHP n'utilisera aucun tampon et enverra directement ses résultats au serveur web.

### *Vider le tampon*

Il est possible de forcer PHP à envoyer tout son tampon interne vers la sortie avec la fonction `flush()`, sans paramètres.

Pour plus de simplicité, il est aussi possible de demander à PHP de vider son tampon à chaque fois que le script envoie quelque chose. Il suffit alors d'activer la directive `implicit_flush` dans le `php.ini`. Il est aussi possible de l'activer pendant l'exécution avec `ob_implicit_flush()`. Faites attention toutefois aux conséquences, car cette fonction désactive automatiquement tous les tampons utilisateurs en cours comme si on avait fait un `ob_end_flush()`.

### *Autres tampons en jeu*

Si vous avez encore l'impression que PHP n'envoie pas tous les caractères dès qu'ils sont disponibles, c'est probablement dû à votre serveur web ou à votre navigateur. En effet, le serveur web gère lui aussi un petit tampon pour son propre usage. Ce tampon peut même être important si vous utilisez certains modules comme le `mod_gzip` pour Apache. Certaines versions de Microsoft Internet Explorer retardent aussi l'affichage tant que la page n'est pas complète ou qu'elle n'a pas envoyé 256 octets.

# 16

## Envoyer et recevoir des e-mails

---

Le courrier électronique est devenu l'un des principaux moyens de communication dans les entreprises. Que ce soit dans la gestion d'un site ou pour un progiciel, l'*e-mail* constitue l'un des éléments indispensables à l'amélioration du confort de l'utilisateur. Dans l'absolu, l'internaute peut ne plus avoir besoin de venir sur un site pour suivre un achat ou chercher des nouveautés ; il est automatiquement prévenu par e-mail et peut se rendre sur le site au bon endroit, au bon moment et dans les meilleurs délais. On parle alors de technique de *PUSH* : l'information est envoyée vers l'utilisateur au lieu d'attendre qu'il vienne la chercher. Un autre service très utile est la réalisation d'un *webmail* permettant à tous les utilisateurs d'un extranet d'accéder à leur courriel de n'importe où. Enfin, il est possible de déclencher des actions à la réception d'un e-mail pour déclencher une sauvegarde par exemple. Nous verrons dans ce chapitre comment fonctionne la gestion des e-mails avec PHP.

### De l'utilité de gérer des e-mails

Si vous faites partie des personnes ayant déjà acheté sur Internet, vous aurez sûrement remarqué que les sites commerçants vous envoient souvent par e-mail une confirmation de votre commande. Certains disposent même d'un système permettant de vous informer des étapes de traitement de votre commande.

Ce type de service permet de gérer l'aspect relation client (*Customer Relationship Management*). En voici quelques cas d'utilisation courants :

- envoyer une lettre d'information (*newsletter*) personnalisée ;



- tenir ses clients informés des mises à jour ;
- disposer d'un gestionnaire de courrier électronique permettant de consulter ses messages de n'importe où (webmail) ;
- faire vivre un forum, permettre d'avertir automatiquement une personne ayant posté un texte sur un forum qu'une réponse lui a été faite ;
- être tenu informé de comportements anormaux ou d'erreurs sur le site web.

Comme nous le verrons en détail au cours de ce chapitre, PHP vous permet d'envoyer des e-mails, autant sous format texte que sous format HTML.

Dans le cadre de l'envoi d'une lettre d'information (sollicitée), les retours et les statistiques tendent à montrer que les campagnes de publicité au format HTML ont un impact nettement plus important que les campagnes en mode texte, et cela bien que les clients de messagerie ne prennent pas tous en charge les e-mails HTML.

#### L'abus et le SPAM

Il faut toutefois faire très attention à l'utilisation que vous faites des e-mails. Le *SPAM* (envoi de courrier non sollicité), interdit par la loi, rapporte rarement des clients et, au contraire, joue plutôt en votre défaveur.

## Webmail Open Source

La gestion des e-mails en PHP a mobilisé beaucoup de développeurs sur des projets Open Source. La difficulté réside dans le choix des bibliothèques utilisées. Nous allons ici présenter deux outils de webmail. En fin de chapitre, nous reviendrons sur une bibliothèque dont le but sera de simplifier vos développements impliquant des envois d'e-mails poussés.

### Nocc

Nocc (*No Carbon Copy*) est un webmail très simple à utiliser et à installer. Son utilisation se base soit sur POP3, soit sur IMAP. Datant de plusieurs années, le projet est mature.

Vous trouverez plus de détails sur ce projet à l'adresse : <http://nocc.sourceforge.net/>.

### IMP

IMP (*Internet Messaging Program*) est un webmail PHP très performant, compatible avec IMAP et POP3. Il requiert PHP 4.1.0 (ou plus) et Horde 2.0 (ou plus). Vous pouvez également gérer un carnet d'adresses en installant Turba.

Ce projet est, avec SquirrelMail, la crème des crèmes en matière de webmail, mais n'est pas toujours évident à installer.

Pour plus d'informations, reportez-vous à l'adresse : <http://www.horde.org/imp/>

Figure 16-1  
Le webmail NOCC

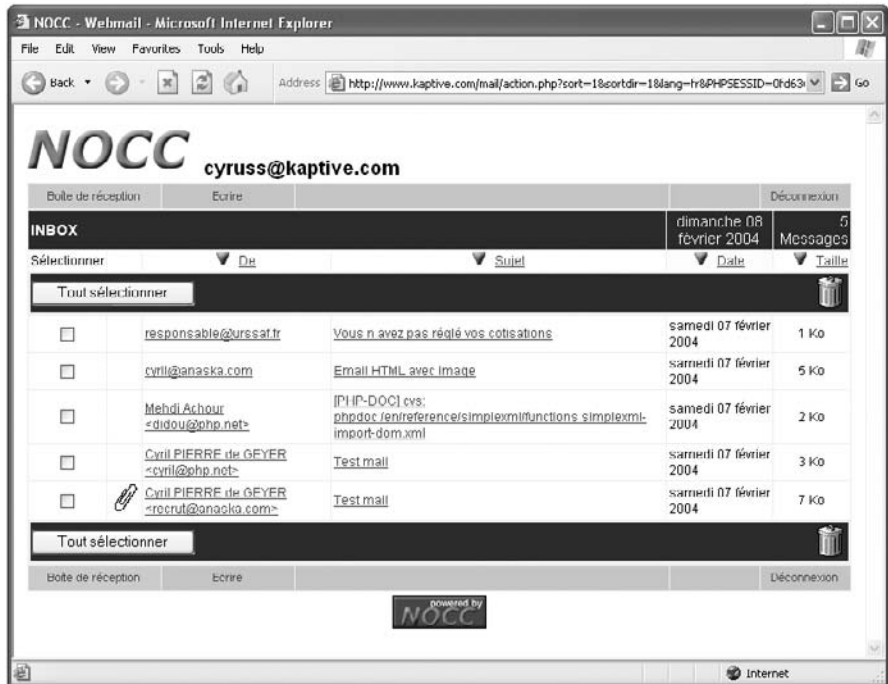
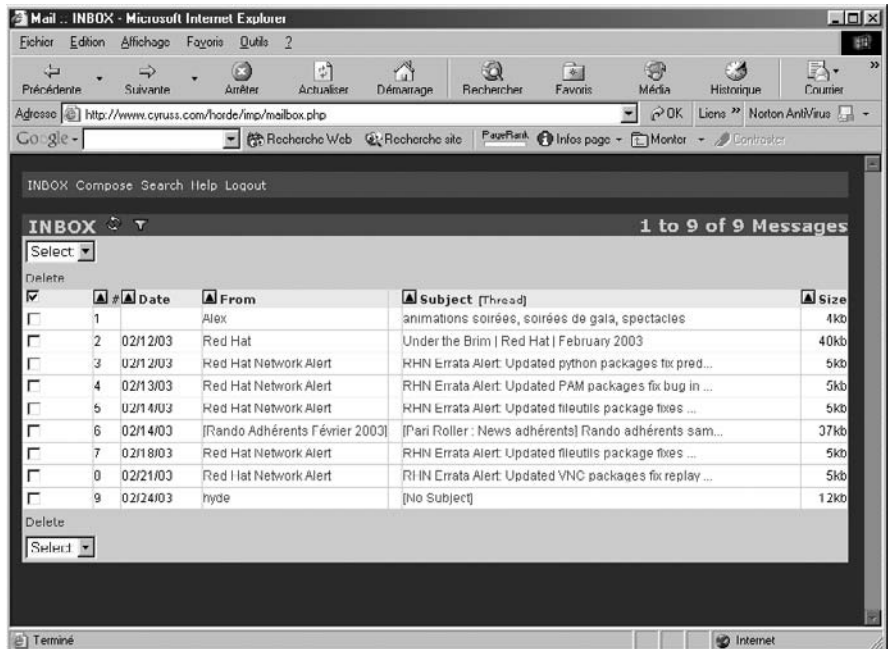


Figure 16-2  
Le webmail IMP



## Mise en œuvre

### Prérequis techniques

Pour que la fonction d'envoi d'e-mail fonctionne correctement, vous devez avoir spécifié un serveur de messagerie dans le fichier `php.ini`. Il doit déjà exister une section semblable à celle-ci :

```
[mail function]
SMTP=mail.php.net ;for win32 only
sendmail_from=cyril@php.net;for win32 only
;sendmail_path=;for unix only
```

### Sous Microsoft Windows

Pour un serveur sous Microsoft Windows, SMTP doit indiquer l'adresse du serveur SMTP de votre fournisseur d'accès (généralement de la forme `mail.<domaine>` ou `smtp.<domaine>`).

Pour le même type de serveur, `sendmail_from` doit indiquer l'adresse e-mail qui sera utilisée par défaut comme adresse source de l'e-mail.

### Sous un système Unix

Sous un système de type Unix, il faut disposer d'un serveur de messagerie de type `sendmail`.

La localisation du programme de `sendmail` (habituellement `/usr/sbin/sendmail` ou `/usr/lib/sendmail`) est effectuée automatiquement. Le script de préconfiguration de PHP, configure, essaie de repérer la présence de `sendmail`, et affecte ce résultat par défaut lors de la compilation. En cas de problème de localisation, vous pouvez donner une nouvelle valeur à la directive de configuration `sendmail_path` dans le fichier `php.ini`.

Si votre système n'utilise pas `sendmail`, il fournit probablement un programme équivalent qui en émule l'interface. La ligne ci-dessous est celle que vous pourriez avoir pour un serveur `qmail` :

```
sendmail_path = "/var/qmail/bin/sendmail"
```

### Pour utiliser les fonctions IMAP

Si vous souhaitez utiliser les fonctions IMAP de PHP, vous devez compiler PHP avec l'option `--with-imap`, ou « décommenter » la ligne correspondante dans le `php.ini` pour Windows.

Pour pouvoir compiler le module IMAP sous Unix, vous devrez avoir installé les fichiers de développement du client de l'université de Washington, que vous pouvez trouver à l'adresse `ftp://ftp.cac.washington.edu/imap/`. Une fois le client compilé, vous devrez copier le fichier `c-client/c-client.a` dans `/usr/local/lib` (ou tout autre répertoire de bibliothèques). Il vous faudra aussi copier les fichiers `c-client/rfc822.h`, `mail.h` et `linkage.h` dans `/usr/local/include` (ou dans le répertoire correspondant).

## Anatomie d'un e-mail

Dans un premier temps, nous allons nous intéresser au courrier électronique au format texte. Nous verrons plus tard dans ce chapitre que les e-mails au format HTML sont plus complexes à utiliser.

Un e-mail est constitué de deux parties : les en-têtes et un corps, le texte du message. Les en-têtes sont assimilables à des données administratives, que nous allons diviser arbitrairement en deux catégories pour une meilleure compréhension.

Une première catégorie contient toutes les informations spécifiques au transport (l'adresse de l'expéditeur, celle du destinataire, etc.). On pourra assimiler cette partie à une enveloppe. Comme un courrier standard, elle permettra de véhiculer le message et sera enrichie par les différents acteurs de l'acheminement.

Une seconde partie enregistre toutes les données nécessaires à la manipulation de l'e-mail. Cette partie n'est pas nécessaire au transport, mais on y trouve le sujet, une liste de destinataires, la date d'envoi, le type de contenu, etc. On pourra l'assimiler à la lettre contenue dans l'enveloppe.

Les en-têtes sont standardisés et se retrouvent en début de message. Les applications utilisent le format d'en-tête défini dans la norme RFC 822, qui est le dénominateur commun de la structure d'un courrier.

Le code source suivant est un exemple d'e-mail simple envoyé de `cyruss@cyruss.com` vers `cyril@php.net` :

```
From: cyruss@cyruss.com Tue Oct 1 12:00:50 2004
Return-Path: cyruss@cyruss.com
Received: from php.net (pb2.pair.com [216.92.131.5])by champagne.nexen.net (Postfix)
  with SMTP id DF92F22727for <cyril@php.net>; Tue, 4 Mar 2004 12:00:50 +0100 (CET)
Date: Tue, 1 Mar 2004 12:07:03 +0100
Message-ID: NGBBKCEFALFGGMKAKBPHOECCFFAA.cyruss@cyruss.com
To: cyril@php.net
Subject: Anatomie_d'un_courrier_électronique.
From: "===Cyruss===" cyruss@cyruss.com
Reply-to : cyruss@cyruss.com
```

Corps de texte : ligne 1 !

Seconde ligne : Corps du texte

Chaque en-tête est une ligne composée d'un nom de champ, d'un séparateur (caractère :) suivi de la valeur du champ et d'une fin de ligne :

```
Reply-to: cyruss@cyruss.com
```

**Les RFC utiles**

RFC 821 : *Simple Mail Transfer Protocol* (SMTP)

RFC 822 : *Standard for ARPA Internet text messages*

RFC 2060 : *Internet Message Access Protocol* (IMAP)

RFC 1939 : *Post Office Protocol Version 3* (POP3)

RFC 2076 : *Common Internet Message Headers*.

RFC 2045, RFC 2046, RFC 2047, RFC 2048 et RFC 2049 : *Multipurpose Internet Mail Extensions* (MIME)

Vous trouverez toutes ces spécifications à l'adresse <http://www.rfc.net>.

## Envoyer des e-mails

La gestion des envois d'e-mail en PHP est extrêmement simple. On utilise la fonction `mail()` en lui passant en arguments :

- l'adresse électronique du destinataire ;
- le sujet du courrier ;
- le texte du courrier.

```
<?php
// On envoie un message à cyril@php.net
mail('cyril@php.net', 'Sujet', "Texte du message\nLigne 2\n");
?>
```

Pour revenir à la ligne dans l'e-mail, vous pouvez utiliser, si votre texte est entre guillemets, le code du retour chariot, `\n` : il sera transformé par PHP en un caractère de fin de ligne. Si vous entourez le texte d'apostrophes, les caractères spéciaux ne seront pas remplacés et pourraient empêcher la compréhension de votre demande par le serveur de messagerie.

**Remarque**

Suite à des abus (utilisation pour du SPAM ou envoi de courrier anonyme), sur certains hébergements mutualisés, la fonction `mail()` est soit remplacée, soit recodée, soit tout simplement enlevée. En cas de soucis, regardez la documentation ou la FAQ de votre hébergeur.

## Envoyer un e-mail à plusieurs personnes

Pour envoyer un e-mail à plusieurs personnes, il faut séparer les adresses des destinataires par des virgules dans le premier paramètre.

```
<?php
$destinataires = 'cyril@php.net,ab@anaska.com';
// On sépare les destinataires par une virgule.
$sujet = 'Titre du message';
mail($destinataires, $sujet, "Texte\nLigne 2");
?>
```

## Changer l'expéditeur

Pour changer l'expéditeur du message, une solution est de changer la directive de configuration `sendmail_from`, discutée plus haut. Vous pouvez le faire dans le fichier de configuration global (`php.ini`) ou à l'exécution avec la fonction `ini_set()`.

Si vous souhaitez une méthode plus souple, vous pouvez redéfinir l'en-tête correspondant pendant l'envoi de l'e-mail en ajoutant un quatrième argument à la fonction `mail()`. Cet argument optionnel comprend une chaîne de caractères qui sera ajoutée à la fin des en-têtes. Typiquement, cela permet d'insérer des en-têtes supplémentaires.

L'en-tête `From` définit l'expéditeur du message. Si vous ne la définissez pas, PHP le fait seul à partir de `sendmail_from`.

```
<?php
$destinataire = 'cyril@php.net,ab@anaska.com';
$sujet = 'Vous n\'avez pas réglé vos cotisations';
$entete = "From: responsable@urssaf.fr\n";
mail($destinataire, $sujet, "Texte\n Ligne 2",$entete);
?>
```

### Remarque

Nous voyons ici qu'il est extrêmement facile pour quelqu'un de malintentionné d'envoyer un e-mail en se faisant passer pour quelqu'un d'autre. Ne vous fiez donc jamais à l'expéditeur pour authentifier un e-mail. Seule l'adresse IP du serveur de messagerie expéditeur peut vous donner une indication sur l'origine. Seules les identités basées sur des signatures (clé GPG) ou sur des certificats (X509) ne peuvent pas être facilement usurpées.

## Changer l'adresse de retour

Pour changer l'adresse de retour, il faut ajouter la directive suivante dans l'en-tête :

```
Reply-to: Adresse_e-mail \n
```

On ajoute cet en-tête à ceux déjà envoyés dans le quatrième argument de la fonction `mail()`.

```
<?php
$destinataire = 'cyril@php.net,ab@anaska.com';
$sujet = 'Vous n\'avez pas réglé vos cotisations';
$entetes = "From: responsable@urssaf.fr\n";
// On ajoute maintenant à la variable $entete la directive Reply-to
$entetes .= "Reply-to: adresseretur@urssaf.fr\n";
mail($destinataire, $sujet, "Texte\nLigne 2",$entetes);
?>
```

## Ajouter des personnes en copie

Les e-mails disposent de trois modes d'adressage : l'envoi à un destinataire défini, l'envoi d'une copie carbone (cc, *Carbon Copy*) et enfin l'envoi d'une copie carbone en mode caché (bcc, *Blind Carbon Copy*).

### Note

Dans son utilisation normale, la copie indique que la personne n'est pas le destinataire principal, mais qu'on souhaite le tenir informé. La copie carbone cachée permet de tenir une personne au courant sans que le destinataire principal en soit informé.

Pour ajouter des destinataires de ce type, il faut ajouter une des directives suivantes dans l'en-tête :

```
Cc: Adresse_e-mail \n
Bcc: Adresse_e-mail \n
```

Comme les autres en-têtes, Cc: et Bcc: sont sensibles à la casse et la première lettre doit être en majuscule.

```
<?php
$destinataires = 'cyril@php.net,ab@anaska.com';
$sujet = 'Vous n\'avez pas réglé vos cotisations';
$entetes = "From: responsable@urssaf.fr \n";
$entetes .= "Reply-to: adresseretour@urssaf.fr\n";
$entetes .= "Cc: secretaire@urssaf.fr \n";
$entetes .= "Bcc: contentieux@urssaf.fr \n";
/* Ici notre message sera envoyé en copie à secretaire@urssaf.fr
et en copie cachée à contentieux@urssaf.fr */
mail($destinataires, $sujet, "Texte\nLigne 2",$entetes);
?>
```

### Note

De la même façon que pour envoyer un e-mail à plusieurs personnes, on utilise une virgule pour séparer les différentes adresses électroniques des destinataires en copie (Cc) et en copie cachée (Bcc).

## Modifier la priorité d'un message

Les messages peuvent prendre trois niveaux de priorité via l'ajout de la directive X-Priority dans l'en-tête :

- 5 (basse) ;
- 3 (normale) ;
- 1 (urgent).

```
<?php
$destinataire = 'cyril@php.net';
```





## Type de contenu (Content-Type)

Le paramètre `Content-type` spécifie le contenu du message. Par défaut, sa valeur est `text/plain`.

Les principaux types de contenu régulièrement utilisés sont les suivants :

- `image/jpeg`, `image/png`, `image/gif` : formats d'images jpeg, png et gif.
- `text/plain` : texte pur sans mise en forme.
- `text/html` : message au format HTML.
- `multipart/mixed` : il s'agit d'un jeu générique composé de parties. Il est utilisé quand les parties du corps sont indépendantes et ont besoin d'être liées dans un ordre particulier.
- `text/enriched` : texte avec mise en forme, format initié par AOL.
- `application/octet-stream` : flux binaire opaque, valeur pour un type non textuel inconnu.

Le champ d'en-tête `Content-Type` sert donc à spécifier le type et le sous-type des données contenues dans le corps du message.

Les types non reconnus doivent être traités comme `application/octet-stream` pour indiquer que le corps du message contient des données binaires.

## Jeu de caractères

Pour les données textuelles, on peut également définir le jeu de caractères utilisé avec le paramètre `charset`. Il s'ajoute dans l'en-tête `Content-Type`, séparé du reste par un point-virgule. Généralement, vous voudrez utiliser l'ISO-859-1 ou l'ISO-8859-15, qui contient le symbole « euro » :

```
Content-Type: texte/plain; charset="iso-8859-15"
```

## Codages de transport

Le protocole SMTP a certaines limites qui peuvent être problématiques dans le cadre de l'envoi de certains messages (par exemple de type MIME) :

- limitation des messages électroniques à des données US-ASCII 7 bits ;
- lignes ne contenant pas plus de 1 000 caractères.

C'est la directive `Content-transfer-encoding` qui permet de passer outre cette limitation. Elle permet de définir une transformation sous forme d'encodage du corps du message. Le message est codé lors de l'envoi, transféré, puis décodé par le client à la lecture. Le mécanisme est transparent pour l'utilisateur s'il dispose d'un client de messagerie compatible MIME (les clients de messagerie qui n'acceptent pas MIME deviennent très rares).

## Version MIME

La directive `MIME-Version` permet d'identifier la version MIME du navigateur. Elle est toujours en version 1.0 à l'heure actuelle. Les informations relatives à la version MIME doivent être placées avant tous les autres en-têtes.

## Envoyer des e-mails au format HTML

Envoyer un e-mail au format HTML n'est pas extrêmement compliqué, mais nécessite une certaine rigueur. Pour plus de simplicité dans le développement, on utilise généralement l'une des nombreuses bibliothèques développées pour cette gestion. Vous trouverez l'une d'elles détaillée en fin de ce chapitre.

### Les paramètres

Comme nous l'avons vu, un e-mail au format HTML implique de définir un en-tête particulier. Nous allons utiliser la directive `Content-Type` dans notre en-tête et lui donner la valeur `text/html`.

Nous avons ici construit un e-mail simple au format HTML :

```
<?php
// Définition du destinataire et de l'expéditeur
$destinataire = 'cyril@kaptive.com';
$exp = 'toto@toto.com';
// Définition du corps du message
$html=
'<html><body>' .
'<h1>E-mail au format HTML</h1>'.
'<b><u>Voici un document HTML</u></b><br>'.
'On peut jouer sur les polices, '.
'les tailles et <font color="red">même les couleurs</font>'.
'</body></html>';

mail($destinataire,
    'E-mail au format HTML',
    $html,
    "From: $exp\nReply-To: $exp\nContent-Type: text/html \n");
?>
```

### Gérer les images

La gestion des images dans l'envoi d'e-mails est un sujet important. Plusieurs possibilités s'offrent au développeur. Soit les images sont jointes à l'e-mail, soit on indique dans le code HTML des références absolues vers un emplacement sur le web (<http://www.anaska.com/images/logo.jpg>). Bien sûr, chaque méthode dispose d'avantages et d'inconvénients.

Joindre vos images dans le courrier électronique peut engendrer un e-mail de taille considérable. Une personne ne disposant pas d'une connexion haut débit risque de voir d'un

assez mauvais œil un téléchargement durant plusieurs minutes. De plus, la boîte de réception de votre destinataire risque d'être inutilement occupée s'il reçoit plusieurs e-mails avec les mêmes images (par exemple pour un fond de lettre).

L'autre solution consiste à insérer des liens absolus vers des images hébergées sur votre serveur. Dans ce cas, le client ne télécharge que le fichier HTML, ce qui réduit considérablement l'utilisation de la bande passante. En revanche, l'envers de la médaille est que si vous utilisez cette méthode dans le cadre d'un envoi de lettre d'information, votre serveur devra supporter un nombre de demandes parfois conséquentes. La solution la plus utilisée consiste à temporiser vos envois d'e-mails afin que les clients sollicitent les images sur le serveur à des heures différentes. Logiquement, votre serveur répondra mieux à 10 000 demandes d'images espacées dans la journée qu'à 10 000 demandes dans un intervalle court !

#### Attention

Ne pas inclure les images dans l'e-mail peut être problématique dans le cas d'une lecture sans connexion à Internet. De plus, certains clients de messagerie (comme Mozilla Thunderbird) n'affichent pas les images externes par défaut, pour des raisons de sécurité et de respect de la vie privée. En effet, grâce à ces images, certaines sociétés tiennent à jour des statistiques sur leur lectorat ou les personnes qui lisent leurs e-mails.

Vous trouverez ci-dessous un message au format HTML contenant des images appelées sur un serveur distant.

```
<?php
// Définition du destinataire et de l'expéditeur
$destinataire = 'cyril@php.net';
$exp = 'contact@anaska.com';
// Définition du corps du message
$html=
'<html><head>
<title>Anaska, Formation aux nouvelles technologies </title>
</head>
<body background="http://www.anaska.com/images/fond.gif">
<p>Test</p>
</body>
</html>';

mail($destinataire,
    'E-mail au format HTML',
    $html,
    "From: $exp\nReply-To: $exp\nContent-Type: text/html\n");
?>
```

Une autre méthode consiste à exploiter la balise HTML `<base href="http://www.location.com">` qui permet d'associer tous les liens relatifs à l'adresse donnée.

```
<?php
// Définition du destinataire et de l'expéditeur
$destinataire = 'cyril@php.net';
$exp = 'contact@anaska.com';
// Définition du corps du message
$html=
'<html><head>
<!-- Ici, on utilise une balise HTML permettant de définir une adresse
comme étant point de départ pour les liens relatifs -->
<base href="http://www.anaska.com/">
<title>Anaska, Formation aux nouvelles technologies</title>
</head>
<body background="images/fond.gif">

Test
</body></html>';

mail($destinataire,
    'E-mail au format HTML',
    $html,
    "From: $exp\nReply-To: $exp\nContent-Type: text/html\n");
?>
```

La méthode consistant à insérer des images dans l'e-mail lui-même implique de comprendre les manipulations relatives aux pièces jointes.

## Envoyer des pièces jointes

Maintenant que nous avons vu comment envoyer un e-mail au format HTML, nous pouvons nous demander comment attacher un fichier à cet e-mail. Comme pour la gestion des e-mails au format HTML, nous vous recommandons, pour gagner du temps, d'utiliser l'une des bibliothèques Open Source (voir en fin de chapitre) permettant de gérer facilement ces envois.

Pour gérer l'envoi manuellement, il faut simplement construire un message qui soit conforme au format MIME 1.0 (*Multipurpose Internet Mail Extension*) ; dans l'en-tête du mail, on indique qu'il s'agit d'un mail composé de différentes parties : textes et fichiers attachés. On doit séparer chacune de ces parties par une ligne de délimitation unique (*boundary* en anglais). Chaque partie sera indépendante des précédentes et on précisera pour chacune le type de données dont il s'agit (texte, image, etc.).

### Créer la délimitation unique

Vous aurez remarqué cette ligne dans l'en-tête de l'exemple précédent :

```
Content-type:multipart/mixed;boundary="01fedfd1kss12544sssfdfdfd"
```

Cette directive indique que les différentes parties de l'e-mail (fichiers, texte, images, etc.) seront séparées par la ligne de délimitation unique : "--01fedfd1kss12544sssfdfdfd".

L'unicité de cette ligne de délimitation est primordiale. Si ce n'était pas le cas, on pourrait, par exemple, en essayant de joindre un e-mail dans un autre message, se retrouver avec des conflits lors de la réception.

Pour notre part, nous allons créer un délimiteur en utilisant les fonctions `rand()`, `uniqid()` et `md5()`. Ce délimiteur ne sera pas réellement unique, mais la probabilité d'obtenir deux fois la même chaîne de caractères est tellement faible qu'elle peut être considérée comme négligeable.

```
<?php
$delim = md5(uniqid(md_rand()));
```

#### Note

Il est possible de vérifier que le délimiteur créé n'est pas déjà présent afin d'être certain que tout passera bien.

## Construction du message

Maintenant que nous disposons de notre frontière, nous allons pouvoir ajouter des parties à notre message.

Commençons par décrire les en-têtes :

```
// Type du format MIME utilisé
$head = "MIME-Version: 1.0\n";
// Type de contenu et frontière entre parties.
$head .= "Content-Type:multipart/mixed; boundary=\"$delim\"\n";
$head .= "\n";
```

Le champ d'en-tête `MIME-Version` doit être placé en premier, avant les autres en-têtes MIME. On indique donc ici que l'on va créer un message contenant plusieurs parties de types différents (`Content-Type:multipart/mixed`) et que le séparateur de ces messages sera `boundary=$delim`. La dernière ligne du bloc d'en-têtes est vide : cela déclenche la fin des en-têtes et le début du bloc de contenu.

On peut alors ajouter un message à destination des messageries électroniques ne comprenant par le type MIME.

```
// Message à destination des logiciels ne lisant pas le type MIME
$msg .= "Ce message est au format MIME ... \n";
$msg .= "\n";
```

Nous allons maintenant indiquer que nous rentrons dans la première partie du message en insérant le délimiteur unique créé plus haut.

```
//- Première partie du message,
// on indique la frontière.
$msg .= "--$delim\n";
```

Pour chaque partie, on indiquera le type de contenu. Commençons par mettre du texte normal. On indique également comment il sera codé.

```
// On indique le type de contenu du message
$msg .="Content-Type: text/plain; charset=\"iso-8859-1\"\n";
$msg .="Content-Transfer-Encoding:8bit\n";
```

Il faut alors insérer une ligne vide entre l'en-tête et le message, puis écrire ce dernier.

```
$msg .="\n";
$msg .="Ceci est un exemple d e-mail avec un fichier joint\n";
$msg .="\n";
```

Nous allons maintenant insérer notre fichier joint. La première étape va consister à en récupérer le contenu avant de l'insérer dans le message.

```
//- Seconde partie du message : le fichier joint
$fichier = "./image.gif";
$attache = file_get_contents($fichier);
```

Il nous faut alors convertir le contenu du fichier pour être conforme au format RFC 2045. Pour cela, nous utiliserons les fonctions `chunk_split()` et `base64_encode()`.

```
$attache = chunk_split(base64_encode($attache));
```

Notre fichier est prêt et au bon format, nous pouvons maintenant commencer notre partie du message : tout d'abord, le délimiteur unique puis, comme ci-dessus, le type et la façon dont il est codé.

```
$msg .="--$delim\n";
$msg .="Content-Type: image/gif; name=\"".$fichier."\n";
$msg .="Content-Transfer-Encoding: base64\n";
```

Pour les fichiers joints comme pour les images, nous pouvons demander qu'ils soient affichés dans le corps de l'e-mail lorsque cela est possible. On n'oubliera pas d'insérer une ligne vide ensuite. Pour joindre l'image en tant que fichier à sauvegarder, on aurait utilisé `attachment` à la place de `inline`.

```
$msg .="Content-Disposition: inline; filename=\"".$fichier."\n";
$msg .="\n";
```

On insère alors notre fichier ici.

```
$msg .=$attache . "\n";
$msg .="\n";
```

Notre message est terminé. Il ne nous reste plus qu'à l'indiquer en ajoutant notre délimiteur et deux tirets :

```
$msg .="--$delim--\n";
```

On finira comme d'habitude en utilisant la fonction `mail()` de PHP.

```
$dest = "cyril@php.net";
$exp = "cyruss@cyruss.com";
mail($dest,"Image",$msg,"Reply-to:$exp\nFrom: $exp\n".$head);
?>
```

Voici le script complet :

```
<?php
$delim = md5(uniqid(rand()));
// Type du format MIME utilisé
$head = "MIME-Version: 1.0\n";
// Type de contenu et frontière entre parties
$head .= "Content-Type:multipart/mixed; boundary=\"\$delim\" \n ";
$head .= " \n";

// Message à destination des logiciels ne lisant pas le type MIME
$msg .= "Ce message est au format MIME ... \n ";
$msg .= "\n";

//- Première partie du message
// on indique la frontière
$msg .= "--$delim\n";
// On indique le type de contenu du message
$msg .= "Content-Type: text/plain; charset=\"iso-8859-1\"\n";
$msg .= "Content-Transfer-Encoding:8bit\n";
$msg .= "\n";
$msg .= "Ceci est un exemple d e-mail avec un fichier joint\n";
$msg .= "\n";

//- Seconde partie du message : le fichier joint
$fichier = "image.gif";
$attache = file_get_contents($fichier);
$attache = chunk_split(base64_encode($attache));
$msg .= "--$delim\n";
$msg .= "Content-Type: image/gif; name=\"\$fichier\"\n";
$msg .= "Content-Transfer-Encoding: base64\n";
$msg .= "Content-Disposition: inline; filename=\"\$fichier\"\n";
$msg .= "\n";
$msg .= $attache . "\n";
$msg .= "\n";
$msg .= "--$delim--";
$dest = "cyril@anaska.com";
$exp = "cyrucc@cyrucc.com";

mail($dest,"Image",$msg,"Reply-to:$exp\nFrom: $exp\n".$head);
?>
```

### E-mail HTML avec images attachées

Envoyer un e-mail au format HTML avec des images attachées est assez similaire à l'envoi d'un e-mail avec un ou plusieurs fichiers joints. Dans le premier bloc, il faut signifier que le type est text/html. Ensuite, dans chaque fichier joint, il faut ajouter un

en-tête Content-ID renseignant sur l'identifiant du fichier dans le mail, et préciser cet identifiant dans les balises `` en les faisant précéder de `cid`.

```
<?php
// Construction de l'en-tête
$delim = md5(uniqid(rand()));
$entete = "MIME-Version: 1.0\n";
$entete .= "Content-Type: multipart/related;boundary=\"\$delim\"\n";
$entete .= "\n";

// Construction du message
$msg = " Ce message est au format MIME.\n";

// Le code HTML
$msg .= "--$delim\n";
$msg .= "Content-Type: text/html; charset=\"iso-8859-1\"\n";
$msg .= "Content-Transfer-Encoding:8bit\n";
$msg .= "\n";
$msg .= "<html><head><title>Titre</title></head><body><p>" ;
$msg .= "Voici l'image :<img src=\"cid:image1\" alt=\"\">" ;
$msg .= "</p></body></html>\n";
$msg .= "\n";
```

À ce stade, on vient de créer le message HTML en lui-même. Il reste maintenant à insérer les images. Il nous faudra respecter le nom `cid:image1` lors de l'insertion de notre fichier joint.

Dans un cas réel, on ajoutera autant de fois que nécessaire le code d'accrochage d'images ci-dessous.

```
// Accrochage de l'image
$fichier = "image.gif";
$fichier = file_get_contents($fichier);
$fichier = chunk_split(base64_encode($fichier));

$msg .= "--$delim\n";
$msg .= "Content-Type: image/gif; name=\"\$fichier\"\n";
$msg .= "Content-Transfer-Encoding: base64\n";
$msg .= "Content-ID: <image1>\n";
$msg .= "\n";
$msg .= $fichier . "\n";
$msg .= "\n\n";
// Fin d'accrochage d'image

$msg .= "--$delim--\n";
```



```

$destinataire = "cyril@kaptive.com";
$exp = "cyril@anaska.com";
mail($destinataire,
    "E-mail HTML avec image",$msg,
    "Reply-to: $exp\nFrom: $exp\n".$entete);
?>

```

## Recevoir des e-mails

### POP 3

Le protocole POP (*Post Office Protocol*) permet de récupérer son courrier sur un serveur distant. Ce serveur est souvent nommé `mail.nomdedomaine.com`, `pop.nomdedomaine.com` ou `pop3.nomdedomaine.com`.

Le protocole POP permet de relever le courrier présent dans son compte. Tous les fichiers sont téléchargés et donc accessibles localement. Une illustration de l'architecture est donnée à la figure 16-3.

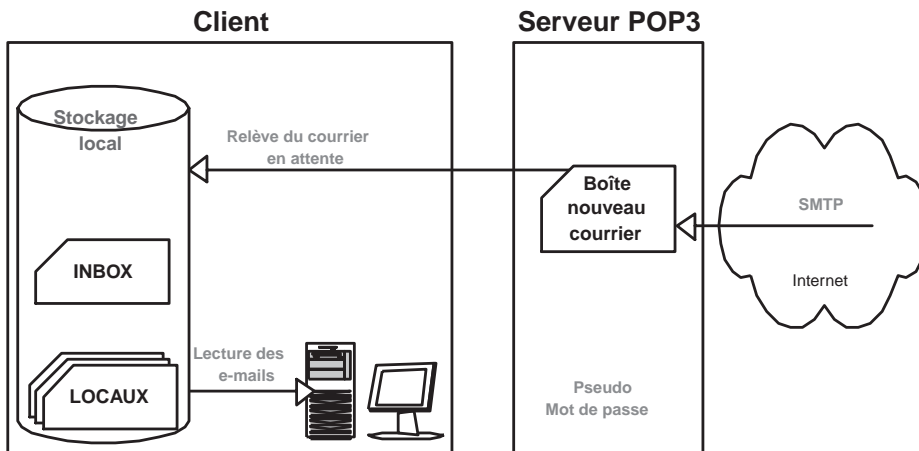


Figure 16-3

Fonctionnement de POP

Le protocole POP3 gère l'authentification à l'aide d'un nom d'utilisateur et d'un mot de passe. Il n'est en revanche pas sécurisé car les mots de passe, au même titre que les e-mails, circulent en clair (de manière non chiffrée) sur le réseau. D'autre part, le protocole POP3 bloque la boîte aux lettres lors de la consultation, ce qui signifie qu'une consultation simultanée par deux utilisateurs d'une même boîte est impossible. Une fois les e-mails récupérés, ils sont généralement supprimés de la boîte aux lettres.

## IMAP 4

Le protocole IMAP (*Internet Mail Access Protocol*) est un protocole alternatif à POP3, mais offrant beaucoup plus de possibilités. Il permet de :

- gérer plusieurs accès simultanés ;
- gérer plusieurs boîtes aux lettres ;
- trier le courrier selon plus de critères ;
- stocker les messages sur le serveur dans une hiérarchie de répertoires.

IMAP peut assurer le traitement hors ligne mais sa force particulière réside dans ses opérations en ligne et en mode déconnecté. En mode en ligne, les courriers sont délivrés au serveur de mail. Le client de messagerie ne les copie pas tous en local avant de les supprimer du serveur. La méthode s'inscrit dans un modèle interactif client-serveur. Le client peut demander de ne récupérer que les en-têtes des messages, les corps de certains messages, voire ne sélectionner que les messages répondant à certains critères. Les messages sur le serveur sont marqués par différents drapeaux d'état (supprimé, répondu, etc.) et restent sur place jusqu'à ce que l'utilisateur demande explicitement leur élimination.

IMAP est conçu pour permettre la manipulation de boîtes aux lettres distantes comme si elles étaient locales. L'architecture IMAP est illustrée à la figure 16-4.

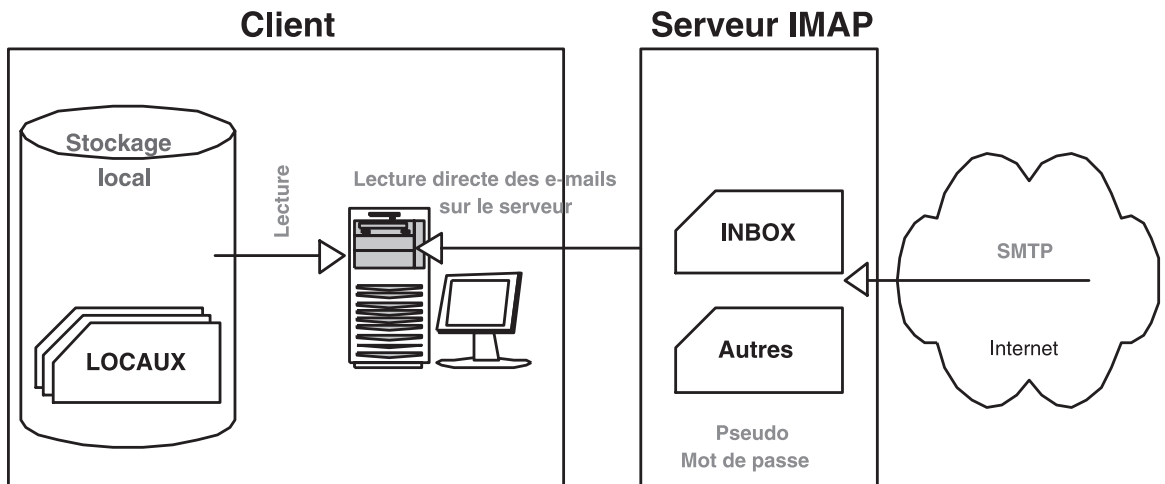


Figure 16-4

Fonctionnement du protocole IMAP

## Comparaison IMAP/POP

Tableau 16-1 Comparatif POP/IMAP

Caractéristiques	IMAP	POP
opérations hors ligne	+	+
Messages accessibles en tout point du réseau.	+	+
Protocoles ouverts	+	+
Clients disponibles pour PC, Mac et station Unix	+	+
Simplicité	+	+
Implémentation		+
Choix de logiciels clients		+
État des messages	+	
Stockage et récupération des messages	+	
Accès à et gestion des boîtes aux lettres multiples	+	
Accès aux données autres que les courriers électroniques, telles que les nouvelles Usenet, les documents	+	

### Ouvrir un flux vers une boîte aux lettres

La première étape d'une connexion va consister à ouvrir un canal vers votre boîte aux lettres. Pour cela, on va utiliser la fonction `imap_open()` en lui indiquant :

- l'adresse de la boîte aux lettres spécifiée de la manière suivante : `{nom_de_serveur:port/protocole}` ;
- le nom d'utilisateur ;
- son mot de passe ;
- éventuellement, on peut spécifier un mode d'ouverture.

```
<?php
// Connexion à un serveur IMAP
$log = "pseudo";
$passe = "mot_de_passe";
$boite = imap_open("{localhost:143}INBOX",$log,$passe);
// Connexion à un serveur POP
$boite2 = imap_open("{localhost:110/pop3}INBOX",$log,$passe);
// Connexion à un serveur de news
$boite3 = imap_open("{news.php.net:119/nntp}php.internals","", "");
?>
```

**Note**

Contrairement à ce que leur nom laisse penser, les fonctions préfixées par `imap_` ne sont pas réservées à la gestion du protocole IMAP. Elles peuvent aussi être utilisées pour gérer le protocole POP.

**Lire les en-têtes de ses e-mails**

Une fois la connexion au serveur établie, on peut commencer à lire les e-mails. Une bonne pratique consiste à ne lire que les en-têtes dans un premier temps : ils nous aideront à décider quels sont les messages intéressants à lire (et éviter de télécharger les autres). Pour cela, on utilise la fonction `imap_headers()`, qui prend en paramètre l'identifiant du canal que nous avons ouvert.

Il est également possible de ne lire qu'un seul en-tête via la fonction `imap_header()`, qui prend en argument le canal ouvert ainsi que le numéro du message.

```
<?php
// Connexion à un serveur IMAP
$log = "pseudo";
$pass = "mot_de_passe";
$boite = imap_open("{localhost:143}INBOX",$log,$pass);
// Récupération de tous les en-têtes
$tableau_entetes = imap_headers($boite);
// Affichons l'ensemble du tableau.
print_r($tableau_entetes);
?>
```

Il est aussi possible de connaître le nombre de messages disponibles sur sa boîte aux lettres avec la fonction `imap_num_msg($boite)` ;.

Dans la section « Cas d'application », nous verrons comment gérer automatiquement des désabonnements à une lettre d'information en utilisant les fonctions IMAP. Nous allons maintenant voir un bref résumé du fonctionnement des principales fonctions IMAP :

- `imap_body()` lit le corps d'un message.
- `imap_check()` vérifie le courrier de la boîte aux lettres courante.
- `imap_close()` ferme le canal de communication avec le serveur.
- `imap_delete()` marque le fichier pour l'effacement, dans la boîte aux lettres courante.
- `imap_expunge()` efface tous les messages marqués pour l'effacement.
- `imap_header()` lit les en-têtes d'un message.
- `imap_headers()` retourne les en-têtes de tous les messages d'une boîte aux lettres.
- `imap_open()` ouvre un canal de communication vers une boîte aux lettres.
- `imap_reopen()` ouvre un nouveau canal de communication vers une boîte aux lettres.
- `imap_sort()` trie des messages.
- `imap_uid()` retourne l'identifiant d'un message.

L'exemple qui suit, tiré de l'étude de cas présentée en fin de chapitre, nous montre comment se connecter à un serveur de messagerie et lire les messages, permettant éventuellement de les traiter.

```
<?php
$email = "newsletter@kaptive.com";
$password = "pass";
$mailbox = imap_open("{imap.kaptive.com:143}INBOX", $boite, $pass);

$check = imap_check($mailbox);
$numMessages = imap_num_msg($mailbox);
for($index=1; $index <= $numMessages; $index++){
    $header = imap_header($mailbox, $index);
    echo "<b>$header->Subject</b><br>\n";
    $from = $header->from[0];
    echo "Provenance : | $from |\n<br>";
    $corps = imap_body($mailbox,$index);
    echo "Corps du message : | $corps |\n<br>";
    if (eregi(".*a detruire.*",$header->Subject) {
        imap_delete($mailbox,$index,0);
    }
    echo "</table>\n";
}

imap_expunge($mailbox);
imap_close($mailbox);
?>
```

De la même façon, il est possible de lire un message présent sur un serveur de *news* :

```
<?php
// Connexion à un serveur de news

$b = imap_open("{news.php.net:119/nntp}php.windows", $log, $passe);
$tableau_entetes = imap_headerinfo($b,22870);

// Affichons l'ensemble du tableau.
print_r($tableau_entetes);
?>
```

## Astuces et sécurité

### Lancer un script à la réception

Sous Linux, il est possible de déclencher l'exécution d'un script PHP à la suite de la réception d'e-mails. Ce type de fonctionnalité a l'avantage de vous ouvrir de nombreuses perspectives. Vous pourrez, sur un simple envoi d'e-mail, commander une sauvegarde d'une base de données puis vous l'envoyer par courrier électronique, déclencher une mise à jour de votre site, valider une commande et automatiser l'envoi de factures, etc.

Le plus simple est de placer un fichier `.forward` à la racine de votre répertoire personnel. Ce fichier contiendra la ligne suivante :

```
#!/usr/local/bin/php -q ~/script.php"
```

Ainsi, lors de la prochaine réception d'un message, le script PHP `script.php` sera exécuté. Assurez-vous cependant de disposer de l'interpréteur PHP en ligne de commande.

#### Attention

Pour que la directive soit prise en compte, il faut configurer votre `sendmail` pour que l'exécution de scripts soit acceptée dans les `.forward`.

## Vérification d'une adresse e-mail

Sur la plupart des sites Internet nécessitant une inscription, on demande aux utilisateurs de donner leur adresse électronique. Il n'est pas rare de recevoir une fausse adresse. Vous pouvez vérifier la cohérence des informations soumises (syntaxe de l'adresse fournie et présence d'un serveur de mail à l'adresse indiquée), mais le seul moyen d'être certain de l'existence d'un *e-mail* est d'y envoyer un message.

Nous vous proposons ci-après deux scripts permettant de vérifier un e-mail. Les deux sont complémentaires : le premier vérifie la syntaxe générale de l'adresse, et le second vérifie que le domaine spécifié existe et sait gérer les e-mails.

#### Note

L'expression rationnelle utilisée est là à titre d'exemple. Elle n'est pas parfaite et ne couvre pas tous les cas possibles. À notre décharge, l'expression théorique couvrirait à peu près une page complète de cet ouvrage. Celle-ci est une bonne approximation pour ce qui nous préoccupe.

```
<?php
function is_email($m) {
    return preg_match(/^[\w.-]+@([\w-]+\.)+[a-z]{2,6}$/i, $m);
}

function verifie_email($email){
    list($compte,$domaine)=split("@",$email,2);
    // la partie suivante ne fonctionne que sous Unix / Linux
    // Une alternative peut être trouvée dans PEAR avec NET_DNS
    if ( !checkdnsrr($domaine,"MX") &&
        !checkdnsrr($domaine,"A")){
        return "Ce domaine n'accepte pas les emails";
    }
    return $email;
}
?>
```

Une autre alternative consiste à utiliser les fonctionnalités de filtre de PHP. Cette solution vous évite la construction d'une expression régulière hasardeuse.

```
<?php
// Validation
$mail = filter_input(INPUT_POST, 'me1', FILTER_VALIDATE_EMAIL) ;

if ($mail == FALSE){
    echo 'Le mail fourni n est pas valide' ;
} else{
    echo "La variable est une adresse email valide : $mail";
}
?>
```

## ***Espacer vos envois en masse***

Gérer un système d'envoi en masse d'information peut être délicat. Il ne faut jamais perdre de vue la charge serveur qui va résulter de votre lettre d'information. Privilégiez donc des envois temporisés si vous voulez répartir la charge dans le temps.

Une société de sécurité très compétente a un jour fait l'erreur d'envoyer tous ses e-mails promotionnels en une fois. Ladite société avait développé un outil permettant de tester les vulnérabilités d'une machine. Dans une lettre d'information envoyée à plusieurs milliers de chefs de projet américains, elle proposait de tester gratuitement l'outil sur une de leurs machines. La publicité était bien faite, la lettre bien rédigée, et des centaines de connexions simultanées arrivaient sur le programme de test. Le système, qui n'était pas prévu pour supporter une telle charge, s'est mis à planter. Le second effet intéressant à noter vient des tranches horaires. Aux États-Unis, les régions n'ont pas toutes la même tranche horaire. Avec une activité professionnelle démarrant en général à 9 heures, il était possible d'apercevoir régulièrement des milliers de personnes se connecter en quelques instants. Elles ouvraient leur boîte aux lettres en arrivant, visionnaient leurs e-mails et cliquaient sur le lien.

## **Bibliothèques Open Source**

### ***HTML Mime mail par phpguru.org***

Comme vous avez pu le constater, autant la création d'e-mail en mode texte est simple, autant la gestion des e-mails au format HTML avec des fichiers attachés peut être complexe. C'est pour cette raison que nous allons vous présenter une bibliothèque permettant de se simplifier la vie. Il s'agit de la bibliothèque HTML Mime mail de phpguru qui vous offre une interface simple pour l'envoi d'e-mails.

#### **Les possibilités de la bibliothèque**

Parmi les possibilités de cette bibliothèque on trouvera, entre autres :

- envoi d'un ou plusieurs fichier(s) attaché(s) ;

- inclusion d'un ou plusieurs destinataire(s) en CC (copie carbone) ;
- envoi à un ou plusieurs destinataire(s) en BCC (copie carbone invisible) ;
- envoi d'un e-mail au format HTML avec équivalent texte pour les clients de messagerie ne prenant pas en charge ce format ;
- envoi d'un e-mail avec des images intégrées.

## Installation

La première étape consiste à télécharger le fichier sur le site <http://phpguru.org/mime.mail.html>

Une fois l'archive décompactée, vous devrez déplacer le fichier `htmlMimeMail5.php` dans un répertoire accessible depuis vos scripts. Il vous suffira de l'inclure avec `include_once()` ou `require_once()`.

## Utilisation de la bibliothèque

À travers l'exemple qui suit, on verra les principales méthodes de la bibliothèque `htmlMimeMail`.

```
include('htmlMimeMail5/htmlMimeMail5.php');
/* Exemple permettant de voir comment
utiliser la classe pour envoyer un e-mail
au format HTML en incluant des images */

// Création de l'objet
$mail=new htmlMimeMail5();

// On récupère le contenu de l'email HTML
$html=file_get_contents('exemple.html');
// Et celui de son équivalent texte
$text=file_get_contents('exemple.txt');

// On inclut une image qui servira de fond à notre e-mail
$mail->addEmbeddedImage(new fileEmbeddedImage('background.gif'));

// on ajoute les contenus à l'e-mail
$mail->setHTML($html);
$mail->setText($text);

// On définit quelques caractéristiques de l'e-mail
$mail->setReturnPath('cyril@php.net');
$mail->setFrom('"Cyril PIERRE de GEYER" <cyril@php.net>');
$mail->setSubject('Test mail');

// Envoi du message, définition du récepteur
$mail->send(array('cyruss@cyruss.com'));
```



## Les méthodes

Définir les champs principaux de l'e-mail

La première étape consiste à créer une instance de l'objet Mail. Pour cela, on utilise le constructeur de la classe :

```
// Création de l'objet
$mail = new htmlMimeMail();
```

On définit ensuite le sujet du message via la méthode `setSubject()` :

```
// Définit la ligne de sujet de l'e-mail
$mail->setSubject('Votre e-mail !');
```

La méthode `setFrom()` nous permettra de définir l'expéditeur de ce message. Son appel est obligatoire.

```
// Définit l'expéditeur de l'e-mail. Appel obligatoire
$mail->setFrom( "Cyril PIERRE de GEYER "<recrut@anaska.com>');
```

On définira alors les destinataires de la copie carbone via la méthode `setCc()` et les destinataires de la copie cachée via la méthode `setBcc()` :

```
// un seul destinataire en Cc
$mail->setCc( "cyril@php.net" );
// un seul destinataire en Bcc
$mail->setBcc("cyril@php.net");
```

On peut également définir une adresse de réponse différente de celle de l'expéditeur.

```
$mail->setReturnPath( "contact@kaptive.com" );
```

Une fois tous ces champs remplis, on va s'attaquer au contenu de l'e-mail à proprement parler. Pour cela, on utilise la méthode `setHTML()` qui prend en paramètre un seul argument si vous souhaitez envoyer un message au format texte :

```
// envoi d'un message texte
$mail->setHtml('Contenu du message au format texte');
```

et deux si vous souhaitez envoyer un e-mail au format HTML :

```
// envoi d'un e-mail texte
$texte = 'Contenu du message au format texte' ;
$html = '<html><body><b>He111looo !!</b></body></html>' ;
$mail->setHtml($html, $texte);
```

## Gestion plus poussée

Rien n'empêche d'attacher un fichier à notre e-mail. Pour cela, nous utiliserons la méthode `addAttachment()`, qui prend trois paramètres : le fichier, son nom et son type.

```
$attache = file_get_contents("../im.gif");
$mail->addAttachment($attache, 'im.gif', 'image/gif');
```

## Envoyer l'e-mail

Enfin, pour envoyer votre e-mail, il faut utiliser la méthode `send()`, qui prend en paramètre un tableau contenant la liste des destinataires. Un second paramètre permet de définir par quel mode vous envoyez votre e-mail : soit la fonction `mail()` de PHP :

```
$result = $mail->send(array('cyril@php.net'), 'smtp');
```

soit directement par SMTP :

```
$result = $mail->send(array('cyril@php.net'), 'mail');
```

## Envoyer un e-mail HTML contenant des images

Pour envoyer un e-mail au format HTML contenant des images, deux possibilités s'offrent à vous. La première consiste à créer votre fichier HTML et toutes ses images dans un répertoire. Vous pourrez alors indiquer à la méthode `setHtml()` un troisième paramètre, qui sera l'adresse du répertoire où se trouvent toutes les images.

```
$html = file_get_contents('monfichier.html');  
$text = file_get_contents('monfichier.txt');  
$mail->setHtml($html, $text, './');
```

Sinon, il vous suffit de faire correspondre le nom de vos images, quand vous les ajoutez à l'objet, au nom indiqué dans votre fichier.

```
$mail->addHtmlImage($background, 'background.gif', 'image/gif');  
$html = "<html><body><img src=background.gif>test</body></html>";  
$mail->setHtml($html, $text);
```

## Définir le jeu de caractères

Vous pouvez également définir le jeu de caractères que vous souhaitez utiliser via la méthode `setTextCharset()` et `setHtmlCharset()`. Le jeu de caractères par défaut de cette bibliothèque est l'ISO-8859-1. Vous voudrez probablement utiliser l'ISO-8859-15 si vous avez besoin du symbole euro.

# Cas d'application

## *Gestion d'une lettre d'information*

### Contexte

Dans le cadre de votre politique de CRM (gestion de la relation client), vous envoyez une lettre d'information régulière à tous les clients de votre site marchand. Comme vous êtes soucieux de respecter les règles de la CNIL en matière de respect de la vie privée, vous autorisez les abonnés à se désinscrire. Jusqu'ici, vous aviez effectué toutes ces désinscriptions manuellement, mais la croissance de votre activité ne vous le permet plus. Vous souhaiteriez donc mettre en place un système automatisé gérant cet aspect de désabonnement.

## Réalisation

Pour éviter de traiter tous les désabonnements manuellement, on utilisera les fonctionnalités IMAP de PHP associées à un mécanisme de déclenchement répétitif ou événementiel.

Pour cela, on propose aux utilisateurs de renvoyer un message sur une adresse en leur demandant d'indiquer un mot-clé dans le message.

On automatise en exécutant un script qui va traiter les messages à intervalles réguliers. Si on souhaite se baser sur des déclenchements événementiels, on exécutera le même script à réception d'un e-mail (voir astuce relative aux `.forward`).

Ici, notre script permettra de désabonner automatiquement un utilisateur qui répond à une lettre d'information en indiquant dans le titre « à détruire » ou « destruction » dans le corps du message.

Le script commence donc par se connecter à la boîte aux lettres via la fonction `imap_open()` et consulte le nombre de messages via `imap_num_msg()`. Pour chaque message, il consulte la zone d'en-tête via la fonction `imap_header()`, lit le corps du message via `imap_body()` et enfin, s'il rencontre un titre contenant « à détruire » ou un corps contenant « destruction », il efface le message.

```
<?php
$login = 'newsletter';
$password = 'pass';
$mailbox = imap_open("{imap.kaptive.com:143}INBOX", $login, $pass);

$check = imap_check($mailbox);
$nMessages = imap_num_msg($mailbox);
echo "<table border=\"1\">
for($index=1; $index <= $nMessages; $index++){
    $header = imap_header($mailbox, $index);
    echo "<tr><td>$header->Subject</td></tr>\n";
    $from = $header->from[0];
    echo "<tr><td>$from</td></tr>\n";
    $corps = imap_body($mailbox,$index);
    echo "<tr><td>$corps</td></tr>\n";
    if (
        strpos($header->subject, "a detruire")!==FALSE
        || strpos($corps, "destruction")!==FALSE
    ) {
        imap_delete($mailbox,$index,0);
    }
    echo "</table>\n";
}

imap_expunge($mailbox);
imap_close($mailbox);
?>
```

# 17

## Travailler avec une base de données

---

De nombreuses applications PHP utilisent une base de données. Bien que la quasi-totalité des SGBD (Systèmes de gestion de base de données) soit supportée, le plus couramment utilisé avec PHP est MySQL. Des systèmes permettant d'installer automatiquement une plate-forme Apache/PHP/MySQL, tels que WAMPServer et EasyPHP, ont grandement participé à rendre ce couple très populaire.

Dans ce chapitre, nous allons commencer par une présentation du langage SQL (orientée MySQL). Le chapitre suivant s'attardera sur la liaison entre PHP et les bases de données.

MySQL, bien que souvent décrié, est doté de nombreux avantages et ses fonctionnalités sont généralement tout à fait suffisantes et adaptées à la majorité des applications. Cela est d'autant plus vrai que MySQL a rattrapé son retard fonctionnel dans la version 5 : prise en charge des requêtes imbriquées, des clés étrangères, des *triggers* et des transactions.

### Utilisation d'un SGBD

#### *Qu'est-ce qu'un SGBD ?*

Une base de données est un ensemble de données organisé de manière que l'on puisse aisément accéder à son contenu et en obtenir de l'information (recoupements, analyses, etc.). Souvent nommé SGBD, un système de gestion de bases de données est un serveur logiciel indépendant qui gère les accès à ces bases de données et fournit des interfaces simples pour les manipuler.

Pour dialoguer avec un SGBD, on utilise un langage nommé SQL (*Structured Query Language*). Chaque SGBD propose quelques fonctions SQL spécifiques, mais tous tendent à se rapprocher des normes définies par l'ANSI (American National Standard Institute).

Il existe plusieurs types de SGBD, les plus courants étant les SGBD relationnels (MySQL, PostgreSQL, Oracle...). D'autres modèles de bases de données existent, tels que le modèle hiérarchique, le modèle réseau, le modèle objet et le modèle déductif. Au cours de ce chapitre, nous nous intéresserons uniquement aux SGBD relationnels, dont le sigle est SGBDR.

### Travailler avec un SGBD relationnel

Une organisation des données sous forme de tables définit le modèle relationnel. Concrètement, les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). On appelle attribut le nom des colonnes. Un attribut est identifié par un nom et un type. On appelle enregistrement (ou *tuple*) une ligne du tableau.

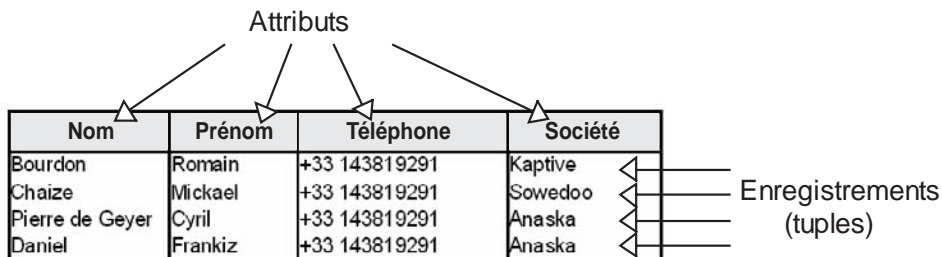


Figure 17-1

*Attributs et enregistrements*

On utilise une base de données pour stocker des informations sur lesquelles il nous sera possible d'effectuer des sélections complexes. Dans notre exemple en figure 17-1, on pourrait aisément sélectionner les personnes travaillant dans la société Anaska. Une base de données pourrait également servir à stocker des logs, des messages d'un forum et une base d'utilisateurs, etc. Notre base de données est dite relationnelle car nous pourrions par exemple relier un utilisateur aux messages qu'il a postés sur un forum et les logs nous permettraient de connaître ses heures de passage. Les tables peuvent être reliées entre elles comme le montre la figure 17-2.

Généralement, le lien entre les tables est réalisé par ce qu'on appelle des clés. Nous traiterons ce thème plus loin dans ce chapitre.

L'utilisation d'une base de données n'est pas forcément adaptée à tous les cas de figure. Si vous n'avez pas besoin de manipuler les données, un stockage sous forme de fichier peut se révéler nettement plus performant.

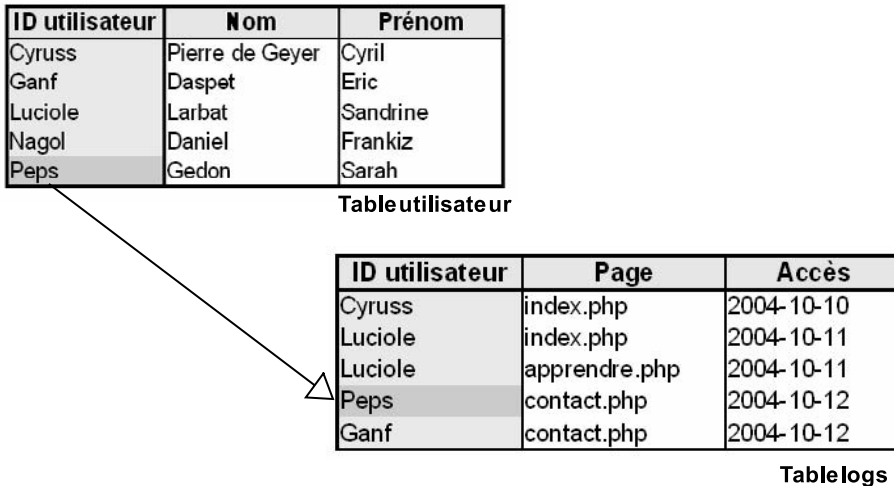


Figure 17-2

Relation entre tables

## Présentation de MySQL

MySQL est un serveur de gestion de bases de données (SGBD) dont les principaux atouts sont la rapidité, la robustesse et la facilité d'utilisation. Son moteur est basé sur la norme ANSI SQL 92, tout en y apportant quelques fonctions spécifiques. Il est disponible sous deux licences, la licence GPL (*General Public License*) des projets GNU et FSF (*Free Software Foundation*) et une licence propriétaire moins contraignante mais payante.

### Note

On notera que MySQL n'est pas compatible à 100 % avec la norme SQL 92. Il s'agit d'un sous-jeu restreint.

Vous trouverez plus d'informations sur <http://www.mysql.com/>.

## Points forts/points faibles

### Vitesse et performances

Le serveur MySQL est réputé très rapide à moyenne et à faible charge. À très forte charge ou avec un ratio écriture/lecture important, d'autres SGBD comme Oracle sont probablement mieux adaptés.

MySQL est tout de même utilisé sur de gros systèmes avec des bases de données de plusieurs gigaoctets de données. Ces utilisations sont faites avec des accès principalement en lecture et gardent une très bonne réactivité par rapport à d'autres systèmes.

MySQL, avec son mécanisme de réplication, permet une grande scalabilité des performances. De nombreux sites à très fort trafic l'utilisent en mettant un serveur maître pour gérer les écritures (INSERT, UPDATE, DELETE) et des serveurs esclaves pour gérer les lectures.

### Connectivité

On peut se connecter et travailler sur une base MySQL en utilisant des interfaces écrites en langages C, Perl, C++, Java, Python, PHP. Utiliser MySQL vous garantit donc une forte connectivité avec l'environnement extérieur. La relation entre MySQL et PHP est historiquement forte, puisque MySQL lui doit en partie son succès.

### Coût

MySQL est distribué gratuitement sous licence GPL. Il est cependant possible de ne pas être restreint aux conditions de la licence GPL en achetant une licence propriétaire auprès de la société MySQL AB (<http://www.mysql.com/company/>).

### Hébergement

Les hébergeurs web proposent en grande majorité à leurs clients le couple PHP/MySQL. Il est ainsi possible de trouver des hébergements de qualité à des prix raisonnables. Il est en revanche plus difficile de trouver des hébergeurs proposant les couples PHP/SQL Server, PHP/Oracle ou PHP/PostgreSQL, et les coûts seront souvent plus élevés.

### Portabilité

MySQL s'exécute sur de nombreux systèmes d'exploitation dont Unix, Microsoft Windows, GNU/Linux ou IBM OS/2.

### Réplication de données

La réplication de données sert à améliorer la solidité et la vitesse de votre application. Pour la solidité, vous pouvez avoir une copie sur un serveur de sauvegarde, qui prend le relais immédiatement si le serveur principal rencontre des problèmes. L'amélioration de la vitesse est obtenue en envoyant les requêtes qui ne nécessitent pas d'accès en écriture vers un serveur esclave.

Depuis la version 3.23.15, MySQL accepte la réplication monodirectionnelle, en interne. Un serveur sert de maître, et les autres serveurs d'esclaves.

### Accès aux sources

Les sources étant fournies, il est possible d'améliorer ou de personnaliser MySQL. Vous avez aussi l'assurance de pouvoir trouver des compétences pour garantir l'assistance technique ou l'évolution de votre système. Vous n'êtes pas enchaîné à un éditeur qui détient les sources d'une composante essentielle de votre application.

## Fonctionnalités

Un des points négatifs de MySQL est le manque de fonctionnalités. Les développeurs travaillent toutefois à combler ce retard depuis la version 4.1. La version 5.0 comble toute différence avec d'autres SGBD plus complets.

Les fonctionnalités disponibles sont toutefois largement suffisantes dans le cas qui nous intéresse le plus ici, c'est-à-dire pour des applications web.

### Requêtes imbriquées

Les requêtes imbriquées sont maintenant reconnues par MySQL 4.1. Il s'agissait d'un manque par rapport aux bases de données proposées par IBM, Microsoft ou Oracle.

### Transactions

Une transaction est une unité logique qui contient un ou plusieurs blocs SQL exécutés par un utilisateur. Toutes les requêtes doivent être exécutées, sans quoi tout est annulé. L'atomicité signifie que soit toutes les requêtes sont effectuées, soit aucune ne l'est. Ce doit par exemple être le cas dans les transactions bancaires : on n'imagine pas que votre compte soit débité sans que votre achat soit payé au vendeur.

### Clés étrangères et intégrité référentielle

Dans une base de données relationnelle, les clés étrangères permettent de symboliser une relation entre deux tables. Cette relation permet d'assurer une intégrité référentielle, c'est-à-dire de vérifier qu'à une référence dans une table correspond bien toujours une entrée dans la table référencée. Cela permet de garder la cohérence et l'intégrité des données.

À partir de MySQL version 3.23.44, les tables de type InnoDB reconnaissent les vérifications d'intégrité référentielle. Pour les autres types de tables, le serveur MySQL accepte la syntaxe `FOREIGN KEY` dans la commande `CREATE TABLE`, mais ne la prend pas en compte.

### Procédures stockées et déclencheurs

Une procédure stockée est une liste de commandes qui peuvent être compilées et stockées sur le serveur. Une fois que cela est fait, les clients n'ont pas besoin de soumettre à nouveau toute la commande mais font simplement référence à la procédure stockée. Cela se traduit par des performances bien meilleures, car les commandes n'ont pas à être analysées plusieurs fois, et ainsi bien moins d'informations transitent sur le réseau. Il y a aussi un gain réel d'efficacité et de clarté puisqu'on évite de répliquer ces commandes plusieurs fois. Le concept peut être rapproché de celui des fonctions que vous créez en PHP.



Un déclencheur (*trigger*) est une procédure stockée qui est activée lorsqu'un événement particulier survient. Vous pouvez par exemple installer une procédure stockée qui est déclenchée dès qu'une ligne est effacée dans une table d'achat, pour que la commande d'un client soit automatiquement effacée si tous ses achats sont effacés.

Ces fonctionnalités sont implémentées avec MySQL version 5.

## Vues

Les vues sont la plupart du temps utiles pour donner aux utilisateurs l'accès à un ensemble de relations représentées par une table. Une vue est une table virtuelle ; les données de la vue sont en fait des champs de différentes tables mis ensemble, ou des résultats d'opérations sur ces champs. Il s'agit par exemple de définir les sélections qui seront le plus souvent faites et de les présenter comme si elles étaient dans une table dédiée.

Beaucoup de bases de données SQL ne permettent pas de mettre à jour les enregistrements d'une vue ; vous devez alors faire les mises à jour dans les tables séparément.

Les vues sont implémentées avec MySQL version 5.

## Types de tables MySQL

MySQL fournit une base de données qui adapte ses capacités et ses performances à vos besoins. MySQL met plusieurs types de tables (appelés également moteurs) à votre disposition et vous permet de les permuter facilement.

MySQL gère deux différents types de tables : les tables transactionnelles (InnoDB et BDB) et les tables non transactionnelles (HEAP, ISAM, MERGE, et MyISAM). Selon votre installation de MySQL, vous disposerez de tout ou partie des moteurs disponibles. Si vous souhaitez en ajouter un, il vous faudra recompiler MySQL.

Les avantages des tables transactionnelles (TST) sont les suivants :

- Ces tables sont plus sûres : même si MySQL s'arrête à la suite d'une erreur interne ou un problème matériel, vous pouvez récupérer vos données, soit par un recouvrement automatique, soit à partir d'une sauvegarde combinée avec le log binaire.
- Vous pouvez combiner plusieurs commandes et les accepter toutes d'un seul coup avec la commande COMMIT.
- Vous pouvez utiliser ROLLBACK pour ignorer vos modifications (si vous n'êtes pas en mode auto-commit).
- Si une mise à jour échoue, tous vos changements seront annulés. (avec les tables non transactionnelles (NTST), tous les changements opérés sont permanents).

Quant aux tables non transactionnelles (NTST), leurs avantages sont les suivants :

- Elles sont plus rapides puisqu'il n'y a pas de traitement des transactions.
- Elles utilisent moins d'espace disque pour la même raison.
- Elles utilisent moins de mémoire pour les mises à jour.

Vous pouvez combiner les tables TST et NTST dans la même requête pour obtenir le meilleur des deux types.

## ISAM (Indexed Sequential Access Method)

### Avantages

Son fonctionnement repose sur le postulat suivant : la base de données sera bien plus souvent interrogée que mise à jour. Par conséquent, les opérations de lecture seront très rapides et ne nécessiteront que peu de ressources serveur.

### Inconvénients

ISAM n'accepte pas les transactions et n'est pas tolérant aux pannes : si votre disque dur tombe en panne, les fichiers de données seront irrécupérables. N'utilisez donc pas ISAM dans une application critique, à moins d'employer des techniques de réplication efficaces.

## MyISAM

MyISAM est le format par défaut de MySQL. Si vous essayez de créer une table dont le type n'existe pas, MyISAM sera utilisé à la place.

Il s'agit d'une extension du type ISAM. En plus de fournir un certain nombre de fonctions de gestion des champs et d'indexation non disponibles dans ISAM, MyISAM utilise un mécanisme de verrouillage des tables pour optimiser plusieurs opérations de lecture et d'écriture simultanées.

En contrepartie, vous devez exécuter la commande `OPTIMIZE TABLE` de temps en temps pour récupérer l'espace occupé inutilement par les algorithmes de mise à jour.

Cette rapidité de consultation des informations est sans doute l'une des principales raisons de la popularité de MySQL dans le domaine du développement relatif à Internet.

## HEAP

Les tables HEAP utilisent un index de hachage et sont stockées en mémoire. Du fait de ce maintien en mémoire, HEAP est plus rapide que MyISAM, mais les données sont volatiles et seront perdues si elles ne sont pas sauvegardées sur disque.

Les tables HEAP sont très utiles lorsque vous voulez utiliser une instruction `SELECT` imbriquée pour sélectionner et manipuler les données.

## InnoDB et Berkeley DB

InnoDB a été conçu pour maximiser les performances lors du traitement de grandes quantités de données.

### Avantages

Les principaux avantages sont :

- la gestion des transactions ;
- la gestion des clés étrangères ;
- la capacité de restauration après crash.

Ces types de tables étant les seuls à reconnaître les transactions et les clés étrangères, ils seront vos seuls choix si vous avez besoin de ces fonctionnalités.

### Inconvénients

Les tables InnoDB et BDB sont plus lentes que les tables HEAP, ISAM et MyISAM.

## Définir le type de vos tables

Avec la plupart des SGBD, il n'est possible de définir le moteur utilisé qu'au niveau de la base. Avec MySQL, il est possible de définir table par table le moteur utilisé. L'élément qui rend possible une telle flexibilité est une extension fournie par MySQL à l'ANSI SQL, à savoir le paramètre `TYPE`. MySQL vous permet de spécifier des moteurs de base de données au niveau des tables, si bien qu'on les qualifie parfois de formats de tables.

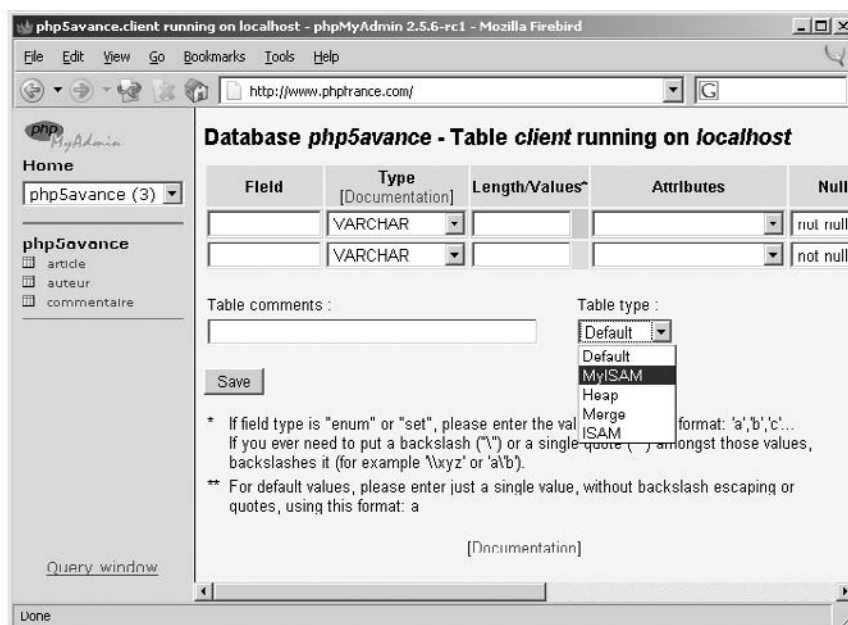
L'exemple de code suivant montre comment créer des tables qui utilisent respectivement les moteurs MyISAM, ISAM et HEAP.

```
CREATE TABLE table1myisam (  
    id INT NOT NULL AUTO_INCREMENT,  
    nom VARCHAR(25)  
) TYPE=MyISAM  
CREATE TABLE table2isam (  
    id INT NOT NULL AUTO_INCREMENT,  
    nom VARCHAR(25)  
) TYPE=ISAM  
CREATE TABLE table3heap (  
    id INT NOT NULL AUTO_INCREMENT,  
    nom VARCHAR(25)  
) TYPE=HEAP
```

Avec phpMyAdmin (application écrite en PHP permettant d'administrer, depuis un navigateur, des bases de données MySQL), il suffit de choisir le moteur de la table lors de sa création.

Figure 17-3

Choix du moteur de table avec phpMyAdmin



Vous pouvez également utiliser la commande `ALTER TABLE` pour transférer une table existante d'un moteur à un autre. Le code suivant illustre l'utilisation de `ALTER TABLE` pour transférer une table `MyISAM` vers le moteur `InnoDB` :

```
ALTER TABLE table1myisam CHANGE TYPE=InnoDB
```

Connaître le moteur utilisé sur une table

Vous pouvez utiliser la commande `SHOW TABLE` pour déterminer quel moteur gère une table donnée. `SHOW TABLE` renvoie un jeu de résultats vous permettant d'obtenir de nombreuses informations. Le nom du moteur de base de données figure dans le champ `Type`.

```
SHOW TABLE STATUS FROM table1
```

Optimiser

MySQL vous permet donc d'adapter votre base de données à vos besoins spécifiques. Si vous avez besoin d'utiliser les transactions, vous pouvez utiliser un moteur transactionnel. Si à l'inverse vous êtes préoccupé par la rapidité, utilisez un moteur non transactionnel, comme `MyISAM`.

## Outils d'administration Open Source

Le principal type d'outils d'administration que l'on peut attendre du couple MySQL/PHP est un gestionnaire de bases de données. Il existe de nombreux outils, mais la référence est de loin `phpMyAdmin`. Cet outil est le principal d'une famille permettant de

gérer différentes bases de données (phpOracleAdmin, phpSybaseAdmin, phpPgAdmin, etc.). Si vous préférez un outil disponible en local sur votre machine, vous pouvez utiliser les logiciels *MySQL Query Browser* et *MySQL Administrator* développés par MySQL AB respectivement pour la gestion des requêtes et pour l'administration de votre serveur.

Pour créer et concevoir votre base de données, vous pouvez utiliser l'outil de MySQL « MySQL WorkBench » ou DbDesigner (téléchargeable à l'adresse <http://www.fabforce.net/dbdesigner4/>). Ce dernier vous permet de travailler sur le modèle conceptuel de votre base de données.

## phpMyAdmin

phpMyAdmin est un outil développé en PHP destiné à faciliter la gestion d'un ensemble de bases de données MySQL et cela à l'aide d'un simple navigateur. Il s'agit d'un des outils Open Source phares gravitant autour de PHP et ses fonctionnalités sont très poussées.

phpMyAdmin permet de gérer l'ensemble d'un serveur MySQL aussi bien qu'une simple base de données. Il permet d'opérer facilement les tâches d'administration courantes, de créer une structure de tables rapidement ou de tester les requêtes pendant le développement.

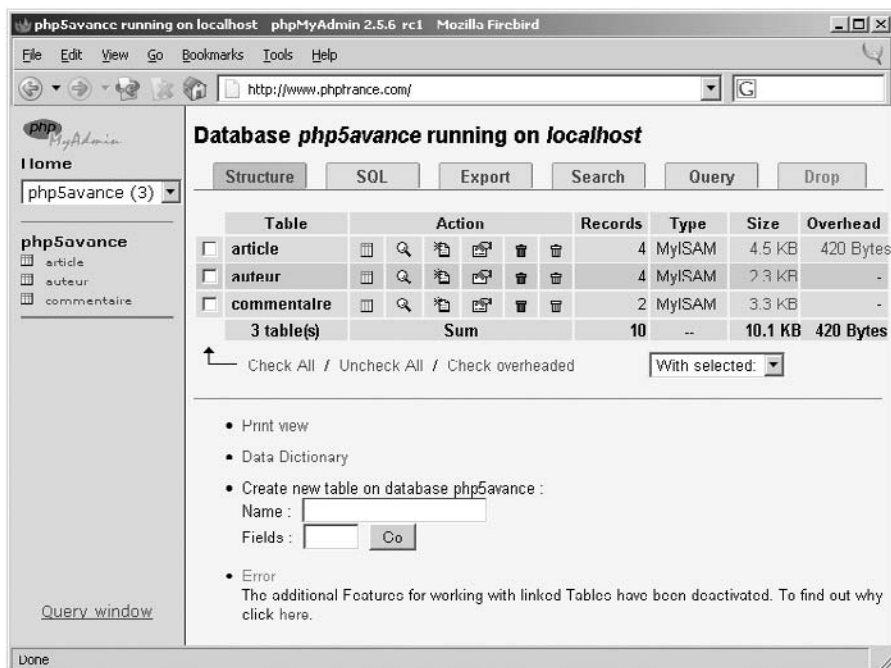
Son installation se fait de façon classique : téléchargez l'archive sur le site <http://www.phpmyadmin.net/>, décompressez-la et placez les fichiers dans un répertoire présent dans l'arborescence web. Ouvrez ensuite le fichier `config.inc.php` avec votre éditeur et modifiez les valeurs suivantes :

- `$host` - indiquez ici le nom de votre serveur de bases de données ou son adresse IP ; le plus souvent, phpMyAdmin étant installé sur la machine hébergeant le serveur MySQL, vous aurez simplement à spécifier `localhost`.
- `$user` - indiquez le nom de l'utilisateur MySQL que vous souhaitez utiliser.
- `$password` - indiquez le mot de passe correspondant à l'utilisateur que vous souhaitez utiliser.
- `$auth_type` - il s'agit du mode d'authentification que vous souhaitez utiliser. Soit vous servez de l'utilisateur MySQL défini dans ce fichier de configuration, soit vous utilisez l'authentification `http` qui vous demande un couple utilisateur/mot de passe et qui vous authentifie comme étant cet utilisateur MySQL.

```
$cfg['Servers'][$i]['host']      = 'localhost';
$cfg['Servers'][$i]['auth_type'] = 'http';
$cfg['Servers'][$i]['user']     = 'eurofacturier_user';
$cfg['Servers'][$i]['password'] = 'mon_mot_de_passe';
```

Il est possible de définir de nombreux autres paramètres dans le fichier de configuration, mais nous nous limitons au nécessaire pour s'adapter aux besoins de ce chapitre.

Figure 17-4  
*phpMyAdmin*



#### Note

Autoriser les connexions à phpMyAdmin en provenance d'Internet implique de définir une politique de sécurité pour éviter qu'une personne mal intentionnée ne vole ou ne détruise vos informations. Une protection faite via les contrôles d'accès du serveur web est recommandée. Consultez le chapitre dédié à la sécurité pour plus d'informations.

## Les commandes SQL

Nous allons revoir ici les principales commandes SQL (*Structured Query Language*), qui vont nous permettre de travailler avec MySQL. Cette revue d'instructions est loin d'être exhaustive ; elle présente simplement les manipulations courantes nécessaires. Si vous ne connaissez pas le langage SQL et si vous avez besoin d'opérer certaines tâches, vous pouvez essayer de le faire avec phpMyAdmin. L'outil vous montre en effet à chaque fois la requête SQL qu'il a exécutée, vous pouvez donc la recopier dans un fichier personnel de formation pour pouvoir ensuite la retrouver facilement et l'adapter selon vos besoins.

## Créer une base de données

Tout d'abord, pour créer une base de données, il faut disposer des droits d'accès adéquats. Nous supposons que vous disposez des pouvoirs nécessaires. Sur un hébergement mutualisé, vous serez probablement restreint à une (ou plusieurs) base préalablement créée par votre hébergeur ; vous pouvez donc ignorer cette étape.

Pour créer une base de données nommée *eurofactory*, vous aurez à exécuter la requête suivante :

```
CREATE DATABASE eurofactory;
```

Sous phpMyAdmin, il vous suffira d'entrer le nom de la base que vous souhaitez créer :

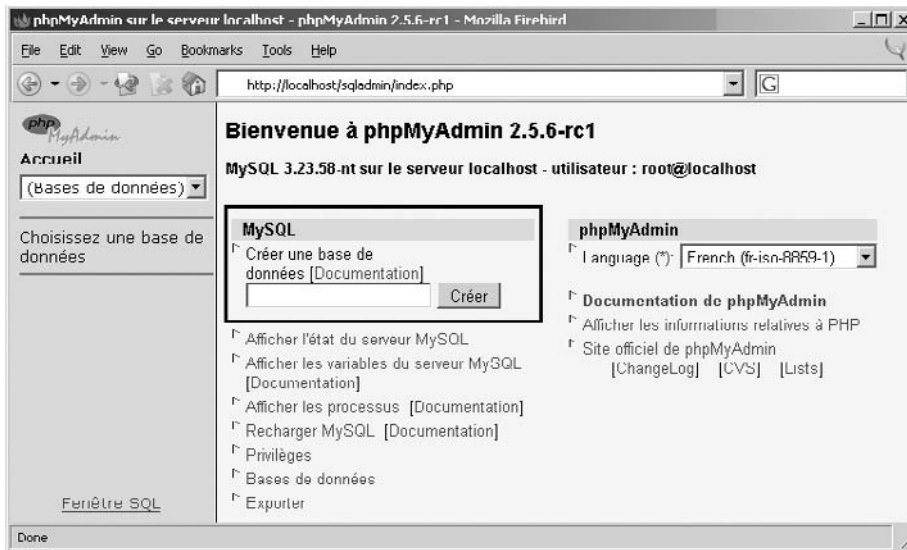


Figure 17-5

Création d'une base dans phpMyAdmin

Si vous créez une base dans l'optique de l'associer au développement d'une application, il est recommandé de créer un utilisateur propre aux scripts PHP qui s'y connecteront. Ainsi, vous pourrez donner des droits spécifiques à cet utilisateur (ne pas donner les droits de supprimer des tables par exemple).

### Attention

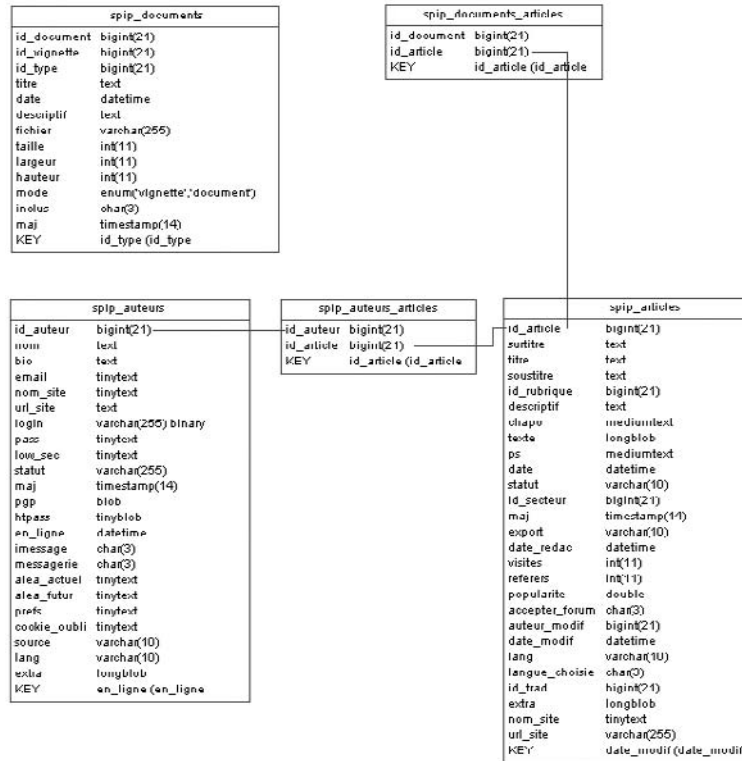
Se servir d'un utilisateur identique sur toutes vos bases implique que toutes les tables seront accessibles avec ce nom d'utilisateur. En cas de récupération de votre couple utilisateur/mot de passe par un tiers, vous risquez donc la compromission de l'ensemble de vos données sur toutes ces tables.

## Créer des tables

Une fois votre base de données créée, il est nécessaire de créer les tables que vous avez précédemment définies dans votre modèle physique de données (MPD, voir exemple à la figure 17-6).

Figure 17-6

Modèle physique de données (MPD)



### Note

Travailler avec un modèle conceptuel de données (MCD) ou un modèle physique de données (MPD) engendre un gain de temps non négligeable. Inutile d'aller chercher le nom ou le type d'un champ dans phpMyAdmin, il vous suffit de regarder votre MCD imprimé. Remarquez qu'il est possible de faire du *reverse engineering* (ingénierie inverse) pour créer un MCD à partir d'un *dump* SQL (exportation de la base de données, généralement pour sauvegarde).

La syntaxe générale de la création d'une table est :

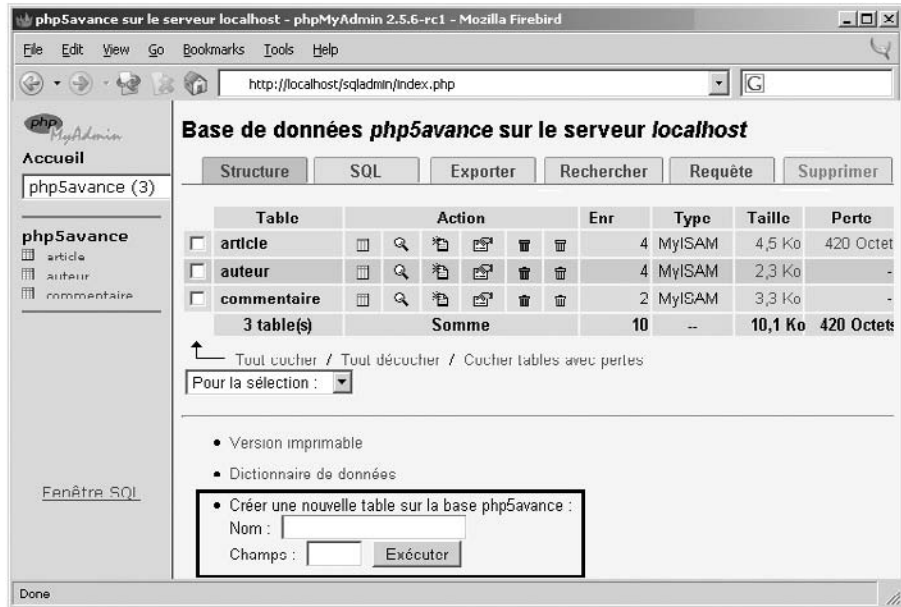
```
CREATE TABLE client ( id int(11), s char(60) )
```

Entre les parenthèses, on indiquera tous les attributs de la table. Cela peut vite devenir complexe si votre table contient de nombreux champs. La manipulation est bien plus



évidente avec phpMyAdmin, ainsi que décrit dans la figure 17-7. Utiliser une interface graphique pour vos créations (ou pour produire un code que vous recopiez dans vos scripts) vous simplifiera beaucoup le travail.

**Figure 17-7**  
Création de table  
dans phpMyAdmin



Avec phpMyAdmin, vous commencez par indiquer le nom de la table et le nombre de champs désirés avant de passer à un écran de saisie vous demandant de nommer et de définir les champs souhaités (voir figure 17-8).

Champ	Type [Documentation]	Taille	Attributs	Null	Défaut <sup>***</sup>	Extra	Prim	Index	Unique	...	Texte
login	VARCHAR			not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
categorie	TINYINT			not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
	VARCHAR			not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>
	VARCHAR			not null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="checkbox"/>

**Figure 17-8**  
Définition des champs

## Principaux types de champs

### Entiers

Les principaux types sont TINYINT, INT et BIGINT, qui sont respectivement codés sur 1, 4 et 8 octets.

Par défaut, les entiers seront signés (ils peuvent être positifs ou négatifs). Si vous souhaitez avoir un maximum en valeur absolue plus élevé et si vous n'avez pas besoin de nombres négatifs, il est possible de définir votre entier comme non signé en ajoutant le paramètre `unsigned`. Les extrêmes vont alors de -128 à 127 pour les `TINYINT` signés et de 0 à 255 pour les non signés.

### Nombres décimaux

Le tableau 17-1 détaille les différentes possibilités que vous pouvez rencontrer.

**Tableau 17-1 Les types réels**

	Structure
<b>real</b>	[(taille,nb_decim)] synonyme de double
<b>double</b>	[(taille,nb_decim)] nombre à virgule 8 octets ;
<b>float</b>	[(taille,nb_decim)] nombre à virgule 4 octets;
<b>decimal</b>	(taille,nb_decim) nombre stocké comme une chaîne
<b>numeric</b>	(taille,nb_decim) synonyme de décimal.

### Types de dates et d'heures

Les types de dates et d'heures sont `DATETIME`, `TIME`, `YEAR`, `DATE` et `TIMESTAMP` (voir tableau 17-2).

Le type `DATETIME` est prévu pour stocker une date et une heure. Le format utilisé est `AAAA-MM-JJ HH:MM:SS`.

Le type `DATE` est prévu pour stocker seulement une date. Le format est `AAAA-MM-JJ`.

Le type `TIMESTAMP` est un champ automatique. Il est fait pour garder trace la date et l'heure de la dernière mise à jour (utilisation de `INSERT`, `REPLACE` ou `UPDATE`) et sera automatiquement rempli. De ce fait, vous ne devriez y accéder qu'en lecture.

**Tableau 17-2 Structure des champs temporels**

	Structure
<b>date</b>	YYYY-MM-DD
<b>year</b>	YYYY
<b>time</b>	HH:MM:SS
<b>timestamp</b>	YYYYMMDDHHMMSS
<b>datetime</b>	YYYY-MM-DD HH:MM:SS

#### Traitement des dates

Pour plus d'informations, voir le chapitre 7 traitant des fonctions usuelles et, plus précisément, la partie sur la gestion des dates.

## Types chaînes de caractères

Deux principaux types permettent de manipuler les chaînes de caractères de taille limitée : les types CHAR et VARCHAR. La longueur d'une colonne CHAR est fixée à la longueur que vous avez définie lors de la création de la table. La longueur peut être n'importe quelle valeur entre 1 et 255.

Un champ de type CHAR a une longueur définie. Un CHAR(50) occupera toujours 50 octets, même si vous n'y entrez qu'une chaîne de deux caractères. Pour utiliser des champs de taille dynamique, vous pouvez spécifier à la place un champ de type VARCHAR. La taille indiquée sera alors une taille maximale.

La différence entre les deux types de champ se fait sur le stockage. Elle se voit sur les vitesses de lecture et d'écriture (un champ de type CHAR peut permettre au serveur de faire des manipulations plus rapides si tous les champs sont de taille fixe) et dans l'espace utilisé (un champ de type VARCHAR pourra utiliser moins de place si vous n'allez pas jusqu'à sa taille maximum). Dans la pratique, un texte comme un titre trouvera sa place dans un champ de type VARCHAR, à l'inverse, un numéro de commande de taille fixe trouvera sa place dans un champ de type CHAR.

### Remarque

Le serveur peut lui-même faire les optimisations qu'il juge nécessaires. Il est tout à fait possible qu'en interne, il décide tout de même d'utiliser un champ de type dynamique ou un champ de type fixe ; en donnant une spécification adaptée, vous l'aidez à faire le meilleur choix possible.

## Champs de grande taille

Les champs de type chaînes de caractères étant limités à 255 caractères, on utilise les champs de types BLOB ou TEXT pour gérer les chaînes, textes ou données binaires de plus grande taille. Une valeur de type BLOB est un objet binaire de grande taille qui peut contenir une quantité variable de données. Les quatre types BLOB (TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB) ne diffèrent que par la taille maximale de données qu'ils peuvent contenir.

**Tableau 17-3 Taille maximale des champs de grande taille**

	Nombre d'octets	Nombre de caractères
Tinyblob/tinytext	2 <sup>8</sup> -1	255
blob/text	2 <sup>16</sup> -1	65535
mediumblob/mediumtext	2 <sup>24</sup> -1	16777215
longblob/longtext	2 <sup>32</sup> -1	4294967295 (+/-4Go)

Les quatre types TEXT (TINYTEXT, TEXT, MEDIUMTEXT et LONGTEXT) correspondent aux types BLOB équivalents et ont les mêmes contraintes de stockage. Les types BLOB sont faits pour gérer des données binaires quelconques, les types TEXT sont faits pour gérer des textes lisibles par un humain. Les tris et les sélections sur ces derniers ne sont pas sensibles à la casse.

**Attention**

Une erreur commune consiste à stocker des images ou des fichiers lourds directement dans votre base de données. Ce n'est pas forcément une bonne idée, car cela surcharge la base alors que vous pourriez ne stocker, par exemple, que le nom et l'emplacement du fichier. D'un point de vue général, si vous n'utilisez pas ces grosses données comme critères de tri dans vos requêtes, vous avez tout intérêt à les laisser sur le système de fichiers.

**Gestion de clé primaire**

Pour référencer un enregistrement, on utilise généralement une clé primaire. Une clé primaire est une colonne, ou un ensemble de colonnes, permettant d'identifier de manière unique un enregistrement dans la table. Il en découle que deux enregistrements ne pourront jamais avoir la même clé primaire.

**Remarque**

Une clé primaire simple fonctionne un peu comme un numéro de sécurité sociale, c'est-à-dire qu'elle est unique et donc ne référence qu'un seul enregistrement.

Il faut donc la choisir de façon à être sûr d'éviter ce type de comportement. D'ailleurs, si vous essayez d'insérer un enregistrement composé d'une clé primaire déjà existante, vous vous heurterez à un message d'erreur.

**Que mettre en clé primaire ?**

Comme nous l'avons dit, une clé primaire désigne de manière unique un enregistrement de la table. Plusieurs cas de figure sont possibles :

- Une (ou plusieurs) colonne(s) de votre table forme(nt) une clé unique identifiant un enregistrement. Dans ce cas, la solution consiste à choisir cette colonne ou cet ensemble de colonnes comme clé primaire.
- Aucune colonne ou association de colonnes de votre table ne permet d'identifier de manière unique un enregistrement. Dans ce cas, on choisit généralement d'ajouter comme clé primaire un index numérique (entier positif) incrémenté automatiquement. L'inconvénient de cette méthode est que vous ajoutez à votre table une information qui n'a pas de sens propre.

## Définition d'une clé primaire

La définition d'une clé primaire s'effectue généralement lors de la création de la table, mais doit être pensée lors de la phase de conception. Le code suivant illustre la définition d'une clé primaire sur plusieurs champs :

```
CREATE TABLE utilisateur (  
    nom VARCHAR(40) NOT NULL,  
    prenom VARCHAR(40),  
    login VARCHAR(40) NOT NULL,  
    email VARCHAR(40),  
    adresse TINYTEXT,  
    PRIMARY KEY (nom,login)  
);
```

Si la clé primaire est constituée d'un seul champ, on peut la définir lors de la définition de celui-ci :

```
CREATE TABLE articles (  
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    titre VARCHAR(200),  
    corps TEXT,  
    FULLTEXT (titre,corps)  
);
```

## Champ auto-incrémenté

Nous venons de voir que MySQL gère un type de clé primaire spécial nommé auto-incrément. Il s'agit d'un entier qui est géré en interne par le SGBD. À chaque insertion, MySQL calcule un nouveau numéro unique qu'il affecte à la ligne insérée. Pour calculer ce numéro, MySQL se contente d'incrémenter le dernier nombre utilisé dans la table comme clé primaire. Ce comportement permet de bénéficier facilement d'une clé primaire simple quand l'utilisation de nombreuses colonnes est complexe, voire impossible. Sous d'autres SGBD, cette fonctionnalité est gérée par les séquences (mais différemment).

Un champ auto-incrémenté ne peut être qu'un entier positif non nul défini comme clé primaire d'une table. Vous pouvez le définir comme auto-incrémenté en ajoutant la mention `AUTO_INCREMENT` lors de la définition du champ.

```
CREATE TABLE articles (  
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    titre VARCHAR(200),  
    corps TEXT,  
    FULLTEXT (titre,corps)  
);
```

Un des avantages de ce type de champs est que, par la suite, vous n'aurez pas à vous en préoccuper lors des insertions. Il suffira de ne rien spécifier et MySQL renseignera lui-même le champ avec une valeur unique pour chaque enregistrement.

**Note**

En général, s'il est possible de définir une clé primaire sur un champ d'une table (ou une série de champs). Certains experts vous conseillent d'éviter d'utiliser les champs auto-incrémentés qui n'ont aucune signification réelle par rapport à vos données et de préférer une clé primaire composée d'un ou de plusieurs champs « significatifs ». Si vous avez une table qui liste vos utilisateurs, une bonne clé primaire pourrait être le login de la personne, ou son adresse électronique.

## Modifier des tables

```
ALTER TABLE nom_de_table
ADD [COLUMN] definition_de_creation
ou
ADD INDEX [nom_index] (index_nom_colonne,...)
ou
ADD PRIMARY KEY (index_nom_colonne,...)
ou
ADD UNIQUE [nom_index] (index_nom_colonne,...)
ou
ADD FULLTEXT [nom_index] (index_nom_colonne,...)
```

**Note**

Les champs entourés de crochets indiquent leur caractère optionnel.

Vous pouvez changer la structure d'une table existante en utilisant la commande SQL ALTER. Par exemple, vous pouvez ajouter des champs, ajouter des index pour optimiser les recherches, voire renommer la table.

Pour utiliser cette commande, vous devez disposer des droits adéquats, c'est-à-dire des droits ALTER, INSERT et CREATE sur la table en question.

Changez le nom d'une table :

```
mysql> ALTER TABLE table1 RENAME table2 ;
```

Ajoutez une colonne :

```
mysql> ALTER TABLE table1 ADD date_achat TIMESTAMP;
```

Ajoutez un index :

```
mysql> ALTER TABLE table1 ADD INDEX (date_achat);
```

Ajoutez un index et une clé primaire :

```
mysql> ALTER TABLE table1 ADD INDEX (date_achat),
      ADD PRIMARY KEY(id);
```

### Mode de fonctionnement de ALTER TABLE

La commande ALTER TABLE sert à modifier le schéma d'une table existante. Cela permet généralement d'ajouter des champs pour permettre une évolution de l'application.

Pour ne pas perdre vos données existantes, MySQL fonctionne de la manière suivante :

- Il crée une table A comprenant les changements voulus.
- Il copie les enregistrements de l'ancienne table vers la table A.
- L'ancienne table est renommée B.
- La table A est renommée avec le nom de votre ancienne table.
- La table B est supprimée.

Si une seule de ces étapes pose un problème, l'ensemble est annulé. Cette méthode de fonctionnement vous permet de ne pas perdre vos informations en cas de problème et de revenir à l'état initial.

### Supprimer des tables

```
DROP TABLE [IF EXISTS]
nom_de_table [, nom_de_table2,...]
```

DROP TABLE supprime une (ou plusieurs) table(s). Il est recommandé d'être prudent avec cette commande, car toutes les données et la structure de la table sont perdues.

```
mysql> DROP TABLE table1;
```

Si la table n'existe pas, un message d'erreur est retourné. Pour éviter cela, on peut utiliser le mot réservé IF EXISTS.

```
mysql> DROP TABLE IF EXISTS table1;
```

### Insérer des données (INSERT)

```
INSERT [INTO] nom_de_table [(nom_colonne,...)] VALUES (...),...
ou
INSERT [INTO] nom_de_table [(nom_colonne,...)] SELECT ...
ou
INSERT [INTO] nom_de_table SET nom_colonne=(expression)...
```

Une fois vos tables créées, il convient de les remplir. La commande SQL INSERT permet d'insérer de nouveaux enregistrements dans une table. Il existe trois méthodes pour utiliser cette commande.

#### Insertion standard

```
INSERT [INTO] nom_de_table [(nom_colonne,...)] VALUES (...),...
```

Il s'agit d'insérer de nouveaux enregistrements en fonction des valeurs spécifiées par VALUES.

```
mysql> INSERT INTO table1 (champ1, champ2) VALUES ('aa','bb');
```

Si vous ne spécifiez pas les champs pour lesquels vous voulez ajouter des données, vous devez alors insérer vos valeurs dans l'ordre respectif de leur présence dans la définition de la table.

```
mysql> INSERT INTO table1 VALUES ('aa','bb','cc');
```

### Insertion via une sous-requête

```
INSERT [INTO] nom_de_table [(nom_colonne,...)] SELECT ...
```

Il est également possible de faire une insertion en se servant d'une sous-requête utilisant la commande SELECT.

```
mysql> INSERT INTO table (champ1, champ2)
      SELECT table2.id, table2.txt FROM table2 WHERE table2.id > 1;
```

#### Note

Cette syntaxe est spécifique à MySQL et ne se retrouve pas obligatoirement dans les autres SGBDR.

### Insertion non complète via SET

```
INSERT [INTO] nom_de_table SET nom_colonne=(expression)...
```

On peut effectuer des enregistrements en indiquant les noms des champs suivis de leur valeur. Cette syntaxe permet aussi de voir plus facilement quelle valeur est associée à quelle information.

```
mysql> INSERT INTO table1 SET champs1=15, champ2='hello';
```

### Insertion multiple en une passe

Il est parfois nécessaire d'insérer plusieurs enregistrements dans une table en limitant le nombre de requêtes. Pour cela, on peut utiliser la syntaxe suivante :

```
INSERT INTO table (champ1,champ2) VALUES
('va111', 'va112'), ('va121', 'va122'), ('va131', 'va132')
```

#### Attention

Cette syntaxe est cependant à utiliser avec prudence car vous ne pourrez plus accéder au dernier identifiant inséré automatiquement via `mysql_insert_id()`.



## Omission de champs

Si vous n'insérez pas tous les champs via votre requête SQL, les valeurs manquantes prendront la valeur par défaut.

## Utilisation d'un champ auto-incrémenté

Quand vous insérez un enregistrement dans une table contenant un champ auto-incrémenté, il ne faut rien spécifier comme valeur dans la commande INSERT. MySQL remplira seul ce champ en fonction des valeurs déjà existantes dans la table. Dans l'exemple suivant, c'est le champ `post_id` qui est la clé primaire auto-incrémentée (voir figure 17-9).

```
INSERT INTO article (post_id, poster_id, post_text )
VALUES ('', 11, 'Tout d\'abord merci pour ... ')
```

Figure 17-9

Utilisation d'un champ auto-incrémenté

← T →		post_id	poster_id	post_text
Edit	Delete	134	2	Salut,  
Edit	Delete	161	2	Yo desolé du delai mais j'ai essayé de retrouver l...
Edit	Delete	162	2	arghh !!  Merci de l'info je vais voir ca <IMG...
Edit	Delete	163	-1	Tout d'abord, merci pour ta réponse.   Alors voi...

## Modifier des données (UPDATE)

```
UPDATE nom_de_table
SET nom_colonne1=expr1 [, nom_colonne2=expr2, ...]
[WHERE where_definition]
```

UPDATE met à jour des enregistrements dans une table avec de nouvelles valeurs et renvoie le nombre d'enregistrements modifiés.

La clause SET indique les colonnes à modifier et les nouvelles valeurs à leur attribuer.

```
mysql> UPDATE table1 SET nom='PIERRE de GEYER'
```

La clause optionnelle WHERE, si elle est spécifiée, précise les enregistrements à mettre à jour. En absence de celle-ci, tous les enregistrements sont mis à jour.

```
mysql> UPDATE table1 SET nom='DASPET', prenom='Eric' WHERE id=5
```

### Remarque

Si vous changez la valeur d'une colonne en lui spécifiant sa valeur actuelle, MySQL s'en aperçoit et optimise son traitement en ne faisant pas la mise à jour.

Vous pouvez spécifier le mot-clé IGNORE afin qu'une mise à jour ne s'interrompe pas, même si durant l'opération on rencontre des problèmes d'unicité. Les enregistrements posant problème ne seront pas mis à jour.

Si vous accédez à une colonne d'une table dans une expression, UPDATE utilisera la valeur courante de la colonne. Par exemple, la requête suivante ajoute une année à l'âge actuel de tout le monde :

```
mysql> UPDATE table2 SET age=age+1;
```

Les requêtes UPDATE sont évaluées de gauche à droite. Par exemple, la requête suivante double la valeur de la colonne age, puis l'incrémente :

```
mysql> UPDATE table2 SET age=age*2, age=age+1;
```

Depuis la version 4 de MySQL, il est possible d'effectuer un UPDATE qui se base sur plusieurs tables :

```
mysql> UPDATE a,b SET a.prix=b.prix WHERE a.id=b.id;
```

## Effacer des données (DELETE)

```
DELETE [LOW_PRIORITY] FROM nom_de_table
    [WHERE clause_where]
    [ORDER BY ...]
    [LIMIT lignes]
ou
DELETE [LOW_PRIORITY] nom_de_table[*] [,nom_de_table[*] ...]
    FROM table-references
    [WHERE clause_where]
ou
DELETE [LOW_PRIORITY] FROM nom_de_table[*], [nom_de_table[*] ...]
    USING table-references
    [WHERE clause_where]
```

DELETE efface de la table `nom_de_table` les enregistrements qui satisfont la condition donnée par `clause_where`, et retourne le nombre d'enregistrements effacés. Si vous exécutez un DELETE sans clause WHERE, tous les enregistrements sont effacés. Si vous spécifiez le mot-clé `LOW_PRIORITY`, l'exécution de DELETE sera repoussée jusqu'à ce que plus aucun client ne lise la table.

### Note

On peut compiler MySQL avec l'option "I AM A DUMMY" pour obliger la clause WHERE dans un DELETE\*.

L'idée est que seules les lignes concordantes dans les tables énumérées sont effacées. Le but est de pouvoir effacer des lignes de plusieurs tables en même temps tout en utilisant d'autres tables pour les recherches.

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
ou
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

Dans les cas précédents, nous n'avons supprimé les lignes correspondantes que dans les tables t1 et t2. Si une clause `ORDER BY` est utilisée, les enregistrements seront effacés dans l'ordre spécifié par ce critère de tri.

```
DELETE FROM table1 WHERE user = 'daspet' ORDER BY temps LIMIT 1
```

Cela efface une entrée satisfaisant la clause `WHERE`. Ici, parmi toutes les entrées ayant 'daspet' comme valeur pour le champ `user`, celle avec la plus petite valeur pour le champ `temps` sera supprimée.

**Note**

Si vous ne définissez pas la clause `ORDER BY` dans votre requête de suppression, l'entrée effacée ne sera pas forcément la plus vieille (surtout après un `optimize`).

L'option `LIMIT` lignes, spécifique à MySQL pour `DELETE`, donne au serveur le nombre maximal de lignes à effacer avant que le contrôle ne revienne au client.

## Remplacer des données (REPLACE)

```
REPLACE
  [INTO] nom_de_table [(nom_de_colonne,...)]
  VALUES (expression,...),(...),...
ou
REPLACE
  [INTO] nom_de_table [(nom_de_colonne,...)]
  SELECT ...
Ou
REPLACE
  [INTO] nom_de_table
  SET col_name=expression, nom_de_colonne=expression,...
```

`REPLACE` fonctionne comme `INSERT`, si ce n'est qu'elle donne la priorité aux nouvelles données. Si un ancien enregistrement a la même valeur qu'un nouveau pour un index `UNIQUE` ou une clé primaire, l'ancien enregistrement sera effacé avant que le nouveau ne soit inséré.

**Remarque**

Pour utiliser `REPLACE`, vous devez avoir les privilèges `INSERT` et `DELETE` sur la table.

Quand vous utilisez une commande `REPLACE`, `mysql_affected_rows()` retourne la valeur 2 si une nouvelle ligne en remplace une existante (il y a eu une insertion puis une suppression, donc deux opérations). Dans la pratique, cela vous permet de savoir si `REPLACE` a ajouté ou a remplacé une ligne : si le nombre de lignes affectées est égal à 1, il s'agit d'un ajout, s'il est égal à 2, il s'agit d'un remplacement.

## Filterer avec la clause *WHERE*

Il est généralement utile de définir une condition dans une requête SQL. On utilise pour cela le mot-clé *WHERE* pour indiquer la ou les conditions s'appliquant à la requête. La clause *WHERE* s'applique à la majorité des commandes SQL. Il s'agit d'un paramètre supplémentaire permettant de limiter le champ d'action de la requête.

Pour cela, plusieurs manières d'utiliser le mot-clé *WHERE* sont à votre disposition.

La forme d'utilisation la plus simple est la formulation d'une condition d'égalité sur un champ de l'enregistrement. Dans l'exemple qui suit, on obtient toutes les informations disponibles des adhérents dont l'âge est 35 ans.

```
mysql> SELECT * FROM adherent
      WHERE age=35
```

L'égalité n'est pas la seule comparaison possible. Il est possible de raisonner en termes de :

- plus grand que : > ;
- supérieur ou égal à : >= ;
- plus petit que : < ;
- inférieur ou égal à : <= ;
- différent de : <> (ou !=).

De plus, rien n'oblige à se limiter à une condition sur un unique champ :

```
mysql> SELECT nom FROM adherent
      WHERE catetid=7 AND prenom='Eric'
```

Dans l'exemple précédent, on obtient de la liste des adhérents le nom de ceux de catégorie 7 et dont le prénom est Éric.

```
mysql> SELECT nom FROM adherent
      WHERE prenom='Cyril' OR prenom='Eric'
```

Dans ce deuxième exemple, on obtient de la liste des adhérents le nom de ceux prénommés Cyril ou Éric.

L'utilisation d'une énumération avec le mot-clé *IN* permet de regrouper plusieurs comparaisons utilisant l'option *OR*:

```
mysql> SELECT nom FROM adherent
      WHERE prenom IN('Laurent','Henri','Eric','John')
```

Avec *NOT IN*, on définit une liste de valeurs à exclure :

```
mysql> SELECT nom FROM adherent
      WHERE prenom NOT IN('Laurent','Henri','Roland','Nadine')
```

Avec *BETWEEN*, on recherche une ou plusieurs valeurs comprises entre deux limites :

```
mysql> SELECT nom FROM adherent
      WHERE naissance BETWEEN '1970/05/12' AND '1980/06/04'
```

### Texte ressemblant (LIKE)

L'instruction `LIKE` s'emploie avec `WHERE` pour rechercher des ressemblances de chaînes de caractères. Elle utilise le signe `%` comme caractère joker, qui peut remplacer zéro ou plusieurs caractère(s) quelconque(s) (dans les commandes *shell* pour les manipulations de fichiers, on a l'habitude d'utiliser le caractère joker `*` qui a le même rôle). Il est aussi possible de remplacer un et un seul caractère quelconque via le trait de soulignement (`_`).

L'exemple suivant permet d'extraire de la table des adhérents toutes les entrées où les noms commencent par `br` :

```
mysql> SELECT * FROM adherent
WHERE nom LIKE 'br%'
```

À la place de `br%`, on peut mettre une condition de la forme suivante, qui permet d'obtenir de la table `table1` toutes les informations de la liste des noms commençant par `br` avec un `t` comme autre lettre, ou d'avoir la liste des noms commençant par `br` et ayant un `u` en quatrième position :

```
mysql> SELECT * FROM table1 WHERE nom LIKE 'br?t%'
mysql> SELECT * FROM table1 WHERE nom LIKE 'br_u%'
```

### Sélectionner des données (SELECT)

Pour sélectionner des données dans une base SQL, on utilise la commande `SELECT`. Elle sert à obtenir des enregistrements venant d'une ou plusieurs tables.

La structure simplifiée de cette commande est la suivante :

```
SELECT champs
[FROM table_1]
[WHERE condition]
```

Le paramètre `champs` indique les données que vous souhaitez obtenir. Généralement, on utilise directement le nom des champs que l'on souhaite connaître :

```
SELECT nom, prenom, age FROM utilisateur
```

Pour sélectionner tous les champs d'une même table, on utilisera un astérisque (`*`).

```
SELECT * FROM utilisateur
```

Le paramètre `table` indique la (ou les) table(s) où se trouvent les champs recherchés.

Le paramètre `condition` indique les restrictions appliquées à la recherche.

```
SELECT pk_visiteur, age FROM visiteurs WHERE age > 35
```

### Nommer les champs lors d'une sélection

Lors d'une sélection, les champs prennent automatiquement le nom de la colonne dont ils proviennent. Les résultats d'opérations (`count(*)`, `SUM()`, etc.) n'ont, eux, pas de noms associés. Vous pouvez préciser explicitement le nom à associer à un champ en utilisant le mot-clé `AS` suivi du nom à donner. Ce nom peut être réutilisé dans le reste de la requête

(clauses WHERE, GROUP BY, ORDER BY, etc.) ou via PHP lors de la récupération des résultats pour accéder à la valeur.

```
SELECT MIN(nom_champ) AS minimum FROM nom_table  
SELECT count(*) AS nb_result FROM nom_table WHERE id=5
```

### Trier les éléments

Il est courant de permettre à vos utilisateurs de classer des résultats, de les classer vous-même par défaut, etc. Dans ce cas, on utilise la clause ORDER BY.

Vous pouvez faire référence aux champs sélectionnés en sortie dans des clauses ORDER BY et GROUP BY en utilisant les noms des champs ou leurs alias :

```
SELECT * FROM tournament ORDER BY resultat;
```

Pour trier dans l'ordre inverse, ajoutez le mot-clé DESC (descendant) au nom du champ dans la clause ORDER BY. Par défaut, l'ordre ascendant est utilisé ; cela peut être indiqué de façon explicite en utilisant le mot-clé ASC.

```
SELECT * FROM tournament ORDER BY resultat DESC;
```

Vous pouvez trier sur plusieurs critères en les séparant par une virgule.

```
SELECT * FROM tournament ORDER BY resultat DESC, goalaverage DESC;
```

#### Attention

Si vous ne triez pas vos résultats avec ORDER, l'ordre est aléatoire, il s'agira généralement, mais pas toujours, de l'ordre d'insertion des données.

### Limiter le nombre de résultats

Dans le cadre d'un environnement web, lorsqu'une requête renvoie un nombre important de résultats, il n'est pas forcément judicieux de tout afficher sur une page. On utilise généralement la clause LIMIT pour demander au serveur SQL de ne retourner que les premiers résultats ; LIMIT 10 affichera donc les dix premiers résultats.

```
SELECT * FROM table LIMIT 5;  
# Retourne les 5 premiers enregistrements
```

### Sélection page à page

On peut aussi utiliser une navigation dite page à page et sauter les premiers résultats. C'est le type d'affichage fait par exemple dans les formulaires de recherche : on sélectionne les 30 premiers résultats à la première page, puis les trente suivants, et ainsi de suite.

Avec MySQL, il vous faudra préciser un deuxième terme dans la clause LIMIT. Ainsi, LIMIT 10,20 sélectionnera vingt résultats à partir du dixième. La position de départ est notée à partir de zéro.

```
SELECT * FROM table LIMIT 5,10;  
# Retourne les enregistrements 6 à 15 avec MySQL
```

Avec PostgreSQL, la position de départ est donnée avec le paramètre `OFFSET`. La sélection suivante est équivalente à la variante MySQL précédente :

```
SELECT * FROM table LIMIT 10 OFFSET 5 ;  
# Retourne les enregistrements 6 à 15 avec PostgreSQL
```

### Connaître le nombre d'enregistrements

Il n'est pas rare de voir, sur des applications permettant une consultation via une arborescence, le nombre d'articles contenus dans une catégorie.

Une technique, malheureusement répandue, pour calculer le nombre d'enregistrements dans une table consiste à récupérer l'ensemble de la table et à calculer le nombre de lignes. Cette méthode est à proscrire absolument, car elle implique une forte charge sans aucune nécessité.

```
SELECT * FROM produit WHERE prix > 15 ;
```

On utilisera alors la fonction `count(*)` pour éviter cette surcharge.

```
SELECT count(*) FROM produit WHERE prix > 15 ;
```

On utilisera de préférence un alias (`AS compteur`) pour faciliter la manipulation ultérieure du résultat. Ainsi, le nombre de lignes renvoyé par `count(*)` sera le contenu du champ `compteur`.

```
SELECT count(*) AS compteur FROM produit WHERE prix > 15 ;
```

### Résultat minimal ou maximal

```
MIN ([DISTINCT | ALL ] (nom_colonne))  
MAX ([DISTINCT] nom_colonne)
```

Cette fonctionnalité de MySQL sert essentiellement à récupérer un seul résultat dans le cas où l'on cherche à récupérer le plus grand ou le plus petit enregistrement. `MIN()` et `MAX()` s'utilisent pour retourner respectivement la valeur minimale et maximale d'une colonne ; les valeurs `NULL` sont ignorées.

`MIN()` et `MAX()` s'utilisent avec l'instruction `SELECT` sur une colonne de type numérique.

Elles sont généralement associées à la commande `DISTINCT`, qui spécifie que seules les valeurs différentes seront prises en compte. `ALL` (valeur par défaut) spécifie que toutes les valeurs seront prises en compte.

### Gérer les doublons (DISTINCT)

Lorsque le moteur construit la réponse, il renvoie toutes les lignes correspondantes, même si ces dernières sont en double. Il est donc souvent nécessaire d'utiliser le mot-clé `DISTINCT` pour éliminer les doublons dans la réponse.

```
SELECT DISTINCT prenom FROM adherent WHERE age > 25
```

## Gérer les jointures

Lorsque vos informations se trouvent dans plusieurs tables, il est possible de croiser les données lors de la sélection. On parle alors de jointure.

```
SELECT adherent.nom, msg.texte FROM adherent, msg
```

Dans notre exemple, nous faisons appel à deux tables (*adherent* et *msg*) dans la clause *FROM* et, lorsque nous utilisons un champ, nous préfixons son nom par celui de la table auquel il appartient, suivi d'un point. Chaque enregistrement de la table *adherent* sera donc croisé avec chaque enregistrement de la table *msg*. Si chacune contient 500 lignes, il y aura 250 000 lignes résultats. Il est donc important de restreindre les croisements en donnant un critère qui permette de lier ensemble les bons enregistrements.

Dans l'exemple suivant, on récupère des tables *adherent* et *msg* tous les messages précédés du nom de leur auteur :

```
SELECT adherent.nom, msg.texte FROM adherent, msg
WHERE adherent.id_utilisateur = msg.id_auteur
```

Devoir écrire le nom de la table en entier devient vite lourd si l'on a des requêtes longues, aussi conseille-t-on de renommer les tables pour simplifier l'appel :

```
SELECT ad.nom, mg.texte
FROM adherent ad, msg mg
WHERE ad.nom='daspet'
```

Les jointures peuvent facilement faire s'écrouler les performances de votre serveur. Il est impératif de penser à mettre des index sur les champs qui serviront de critères lors de la jointure, afin de faciliter le travail du SGBD.

## Gérer les transactions

Une transaction correspond à un bloc d'instructions que l'on souhaite unies. L'ensemble des requêtes doit être exécuté, sans quoi tout est annulé ; on parle d'atomicité. Ce doit être le cas par exemple dans les transactions bancaires ; on n'imagine pas que votre compte soit débité sans que votre achat soit payé au vendeur, les deux requêtes sont dépendantes l'une de l'autre.

En SQL classique, une transaction commence par l'instruction *BEGIN*. Elle se termine lorsqu'elle est explicitement arrêtée par l'utilisateur. L'instruction *COMMIT* valide tout le bloc SQL et applique les éventuels changements. L'instruction *ROLLBACK* permet d'annuler le bloc SQL et d'annuler les changements qui auraient pu avoir lieu.

### Attention

Une transaction ne peut avoir lieu que sur une table utilisant un moteur transactionnel tel que InnoDB.





# 18

## Utiliser une base de données avec PHP

---

Il existe plusieurs possibilités pour utiliser une base de données avec PHP. Vous pouvez, pour chaque type de SGBD, utiliser une extension native dédiée (mysqli pour MySQL, oci8 pour Oracle...). Bien que ces extensions aient des similitudes entre elles, vous aurez alors à manipuler des fonctions spécifiques différentes selon votre SGBD. L'autre solution que nous vous proposons avec PHP 5 est d'utiliser PDO. Il s'agit d'une extension qui vous permet de travailler de manière unifiée quel que soit votre SGBD. Dans ce livre, nous avons décidé de parler principalement de PDO, car il constitue une méthode résolument tournée vers l'avenir, offrant beaucoup de souplesse et de puissance.

Pour des raisons de compatibilité avec d'anciennes versions de PHP, vous pourriez avoir besoin d'utiliser des fonctions natives spécifiques : c'est pour cela que nous allons introduire brièvement le fonctionnement de l'extension « mysql », largement répandue par PHP 4.

### Approche classique PHP 4

Il est possible que vous n'ayez pas le choix des armes et qu'il vous soit imposé de reprendre une application n'utilisant pas PDO. C'est le cas pour les versions 4 de PHP. Afin de vous permettre de comprendre les différents mécanismes de connexion à une base de données MySQL en PHP 4, nous allons faire un rapide point.

Il faut noter qu'en utilisant l'extension mysql classique vous :

- n'aurez pas accès aux fonctionnalités de MySQL 5 ;
- n'aurez pas accès à la version objet ;

- n'aurez pas accès aux requêtes préparées.

```
<?php
$hote = 'localhost';
$user = 'cyril';
$pass = 'motdepasse';
$base = 'publication';

// Étape 1 : connexion
$link = mysql_connect($hote, $user, $pass);
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
// On choisit la base 'publication'
mysql_select_db($base);

// Étape 2 : création et exécution de la requête SQL
$pseudo = 'Cyril';
// On échappe notre variable :
$pseudo = mysql_real_escape_string($pseudo);

$query = "SELECT * FROM rmq WHERE pseudo = '$pseudo'";

// On exécute la requête SQL
$result = mysql_query($query);

// Étape 3 : traitement du résultat

while ($row = mysql_fetch_assoc($result)) {
    echo $row['pseudo'];
    echo $row['titre'];
    echo $row['texte'];
}

// Étape 4 : libération des ressources et fermeture de la connexion
mysql_free_result($result);
mysql_close($link);

?>
```

Cette approche est liée à l'extension « mysql » de PHP 4. Pour MySQL, il est possible également d'employer l'extension « mysqli » (i pour *improve*) si l'on utilise PHP 5. Celle-ci permet de profiter des nouvelles fonctionnalités de MySQL 5 et offre une approche objet. L'approche « mysqli » est relativement semblable à l'approche « mysql », aussi nous vous invitons à consulter le site de PHP pour plus d'informations.

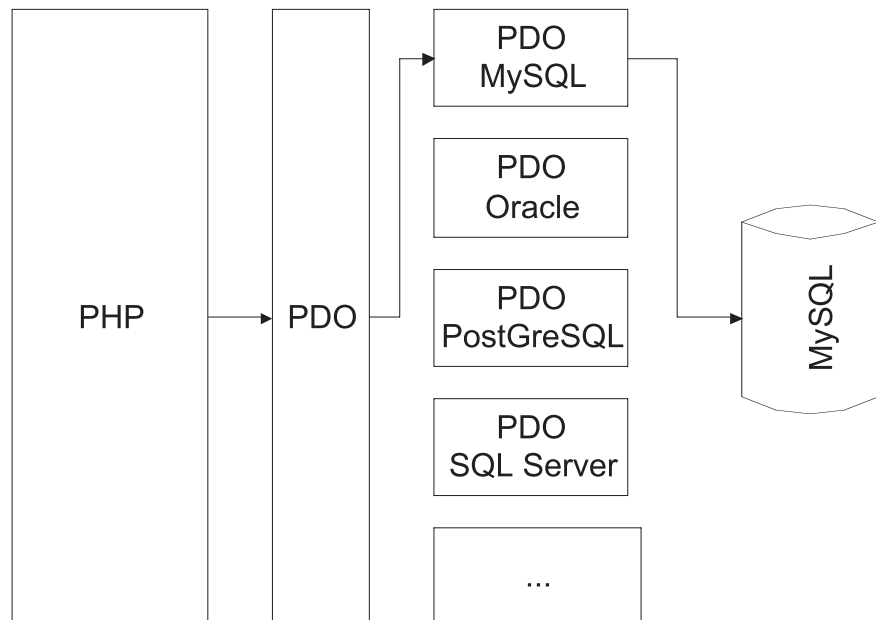
## PDO, PHP Data Object

PDO (PHP Data Object) est la principale nouveauté de PHP 5.1. Cette extension vous apportera un confort d'utilisation et une abstraction plus importants que les anciennes fonctions natives propres à chaque SGBD. L'approche objet de PDO vous permettra de plus d'étendre les fonctions d'accès à votre base facilement et de manière transparente.

En interne, PDO permet à l'équipe de développement de PHP de développer beaucoup plus rapidement de nouveaux connecteurs vers de nouvelles bases de données. Au lieu de tout réécrire à partir du début comme auparavant, ils peuvent se baser sur une architecture complète et ne rajouter que ce qui est spécifique.

PDO est un socle commun pour les connecteurs vers les SGBD. Il s'occupe d'offrir des fonctions de base ainsi que d'unifier les interfaces utilisateur. Il ne s'agit pas à proprement parler d'un système d'abstraction aux bases de données, bien qu'il puisse servir en ce sens.

**Figure 18-1**  
*Architecture des drivers PDO*



### Particularités

#### Performances

Écrit en langage C, PDO est beaucoup plus rapide qu'un système d'abstraction développé en PHP (tel qu'AdoDB, PEAR DB...) et fournit des performances similaires aux anciens pilotes natifs. Les requêtes préparées offrent de plus des possibilités d'optimisation qui n'étaient pas présentes en PHP 4 avec l'ancienne extension MySQL.

## Aptitudes

PDO permet d'exécuter tous les types de requête classiques (INSERT, UPDATE, DELETE, SELECT ou exécution de procédures stockées si votre SGBD le permet). Les données reçues pourront être extraites via plusieurs types de sorties (tableau, objet, variables liées par références...). Les transactions et les modes d'autovalidation (autocommit) sont bien entendu disponibles.

En plus de ces fonctionnalités, PDO permet d'employer des requêtes paramétrées et de normaliser les accès (gestion de la casse des noms de colonnes ou de la syntaxe des paramètres par exemple). PDO émule certaines de ces fonctionnalités si jamais votre SGBD ne les supporte pas (simulation par exemple de l'utilisation de requêtes préparées). Par conséquent, vous n'aurez à porter attention qu'au code SQL lui-même et à ses différences entre les SGBD. Utiliser au maximum du code SQL standard vous permet de réduire fortement la dépendance à votre SGBD.

## Bases de données supportées

PDO inclut une compatibilité avec les principales bases de données avec lesquelles PHP peut communiquer. Dans le cas où la compatibilité native n'est pas supportée, vous pouvez utiliser un pont ODBC. Vous retrouverez entre autres :

- MySQL 3, 4 et 5 (pdo\_mysql) ;
- PostgreSQL (pdo\_pgsql) ;
- SQLite 2 et 3 (pdo\_sqlite) ;
- Oracle (pdo\_oci) ;
- ODBC (pdo\_odbc).

### SQLite3 et PDO

À ce jour, PDO est le seul moyen de se connecter à SQLite3 via PHP.

## Installation

Pour installer PDO, reportez-vous au chapitre 2 concernant l'installation de PHP. Notez qu'il faut activer le noyau PDO et la composante PDO spécifique à votre base de données.

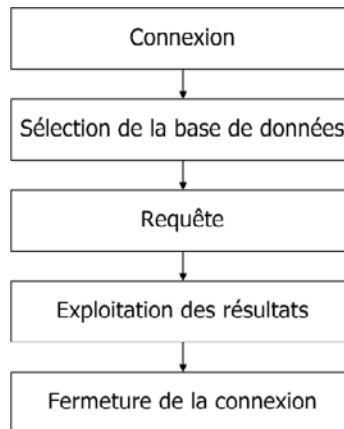
## Utiliser votre base de données

L'utilisation de votre base de données avec PHP s'effectue en cinq étapes, comme le montre la figure 18-2 :

Notons que la dernière action n'est pas obligatoire. La connexion est automatiquement fermée à la fin de l'exécution du script par le moteur PHP. Garder ouverte une connexion si on ne s'en sert plus peut toutefois occuper inutilement des ressources.

Figure 18-2

Utilisation de PHP pour accéder à une base de données



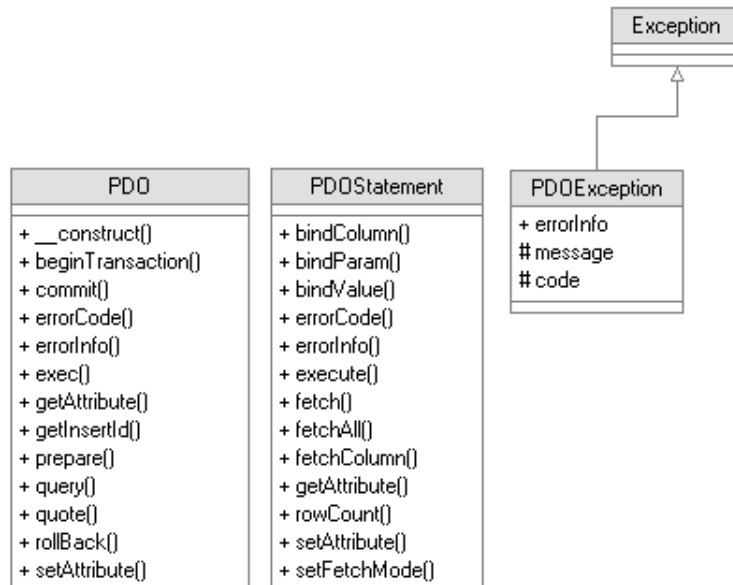
Dans nos exemples suivants, nous allons privilégier l'utilisation du SGBD MySQL. Il se peut que certaines requêtes SQL soient à adapter pour votre SGBD, mais l'utilisation de l'extension PDO reste la même.

## Structure des classes de PDO

Il existe trois classes principales liées à l'utilisation de PDO : la classe PDO qui correspond à votre lien à la base de données ; la classe PDOStatement qui correspond aux requêtes que vous pourriez faire, ainsi qu'à leur résultat ; et enfin la classe PDOException qui permet de traiter les erreurs.

Figure 18-3

Modèle des classes de PDO



## Prise en main rapide

Afin de permettre aux plus pressés de commencer rapidement à manipuler, vous pouvez consulter l'exemple suivant, qui montre comment insérer et lire des données.

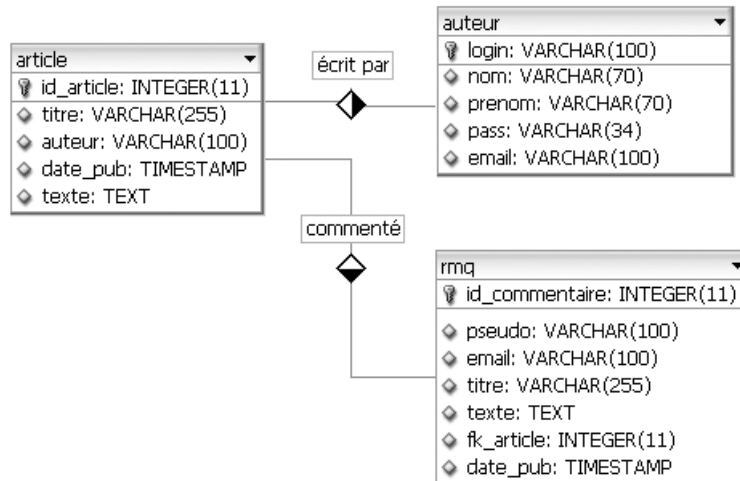
### Prérequis pour les exemples

Pour faire fonctionner ces exemples et les suivants, il vous suffit de créer une base nommée « publication » qui corresponde au modèle de données de la figure 18-4.

Il vous faudra aussi donner les droits d'accès à l'utilisateur « cyril » dont le mot de passe est « motdepasse ».

Figure 18-4

Modèle Physique de  
Données de la base  
d'exemple  
« publication »



```

<?php
// Définition des variables de connexion
$user = 'cyril';
$pass = 'motdepasse';
$dsn = 'mysql:host=localhost;dbname=publication';

// Connexion à la base de données
try {
    $dbh = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
    die( "Erreur ! : " . $e->getMessage() );
}
  
```

```
// Insertion d'un enregistrement
$sql = "INSERT INTO auteur (login) VALUES ('roms')";
$dbh->exec($sql);

// Lecture d'enregistrements
$sql = "SELECT login FROM auteur";
$resultat = $dbh->query($sql);
while ($row = $resultat->fetch()) {
    print_r($row);
}

// Fermeture de la connexion
$dbh = NULL;

?>
```

Si vous obtenez le message suivant :

```
« Erreur ! : could not find driver »
```

cela signifie que vous n'avez pas activé le PDO et/ou son module spécifique à la base de données.

```
« Erreur ! : SQLSTATE[28000] [1045] Access denied for user 'cyril'@'localhost'
  ➔(using password: YES)»
```

Quant à lui, le précédent message indique que l'utilisateur 'cyril' n'a pas le droit de se connecter.

### Erreurs PDO

Les erreurs générées par PDO sont envoyées par défaut sous forme d'exception. Vous pouvez vous reporter au chapitre suivant pour plus de renseignements à ce sujet.

Dans notre exemple, nous utilisons deux méthodes pour exécuter des requêtes SQL : `exec()` et `query()`. La première sert pour les requêtes ne renvoyant pas de résultat (INSERT, UPDATE et DELETE), la seconde renvoie une instance de la classe `PDOStatement` contenant le jeu de résultat correspondant à la requête (utile notamment pour les requêtes SELECT).

Avec ces quelques lignes d'exemple vous serez capable de réaliser la majorité des actions, mais vous pouvez améliorer vos accès à la base de données avec les nombreuses options proposées par PDO. Consultez notamment les requêtes préparées pour protéger vos requêtes des injections SQL (cf. chapitre 27 sur la sécurité).

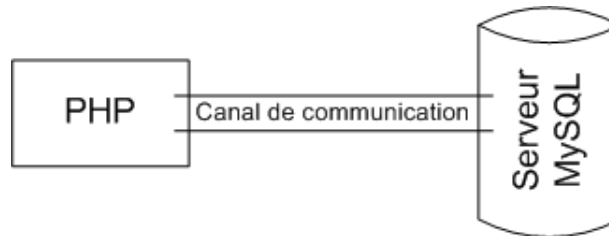


## Connexion au serveur de données

Avant de travailler avec un serveur de gestion de base de données comme MySQL, il faut ouvrir une connexion. Cette connexion sera le canal par lequel PHP et MySQL communiqueront l'un avec l'autre.

Figure 18-5

Ouverture d'une connexion



La première chose à faire est de créer une instance de la classe PDO. Quelle que soit la base de données à laquelle vous souhaitez vous connecter, vous utiliserez la même classe PDO.

Le premier paramètre du constructeur de classe est le DSN (*Data Source Name*), le second le nom d'utilisateur, et le troisième le mot de passe.

### Structure du DSN

Un DSN permet de décrire la base de données à laquelle vous souhaitez accéder.

La convention PDO pour écrire le DSN est d'avoir en premier paramètre le nom du pilote que vous allez utiliser (oci, mysql...). Les paramètres suivants dépendent du SGBD que vous souhaitez utiliser.

```
<?php
$user = 'cyril';
$pass = 'motdepasse';
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

### DSN pour MySQL

Le Data Source Name (DSN) de PDO\_MYSQL est composé des éléments suivants :

- `host` : adresse du serveur distant (nom ou adresse IP, « localhost » pour un serveur local) ;
- `dbname` : nom de la base de données à utiliser ;
- `port` : donnée facultative indiquant le port TCP/IP utilisé pour la connexion ;
- `unix_socket` : donnée facultative indiquant l'adresse de la socket unix pour la connexion locale.

Exemple :

```
mysql:host=Hôte;port=Port;dbname=NomBase  mysql:unix_socket=/tmp/
➔mysql.sock;dbname=NomBase
```

```
<?php
$hôte = 'localhost';
$base = 'test';
$port = '3307';

$socket = '/tmp/mysql.sock';

$dsn = "mysql:host=$hôte;port=$port;dbname=$base";
$dsn2 = "mysql:unix_socket=$socket;dbname=$base";

?>
```

### DSN pour PostgreSQL

Le driver PostgreSQL s'appelle « pgsqldb ». Son DSN est composé de plusieurs paramètres séparés par des espaces :

- host : adresse de la machine ;
- port : numéro du port TCP utilisé ;
- dbname : nom de la base de données ;
- user : identifiant utilisateur ;
- password : mot de passe.

```
pgsql:host=localhost port=5432 dbname=name user=bruce password=pass
```

### DSN pour Oracle

Le driver Oracle s'appelle OCI. Si vous utilisez le tnsname.ora (ce qui est probable avec Oracle), vous n'aurez qu'à ajouter l'identifiant de la base :

```
oci:mabase
```

Toutefois, il est possible aussi d'utiliser l'interface Oracle Instant Client en précisant l'adresse de la machine (nom et port) et le nom de la base. On emploie alors la syntaxe suivante :

```
oci://machine:port/base
```

## Utiliser des connexions persistantes

Le temps nécessaire pour ouvrir une connexion au SGBD représente une partie non négligeable de l'utilisation d'une base de données dans un contexte web. Lors d'une ouverture classique, PHP se voit imposer cette opération au début de chaque exécution.

Lors d'une connexion persistante, PHP ne ferme pas la connexion en fin de script, et la laisse ouverte en mémoire. Lorsque le script suivant cherchera à ouvrir une connexion avec les mêmes paramètres, PHP récupérera l'ancienne, économisant ainsi le coût d'une reconnexion. Cette procédure peut, avec certaines bases de données, permettre un gain important en terme de performances s'il est juste fait quelques calculs SQL simples dans chaque script.

```
<?php
$user = 'cyril';
$pass = 'motdepasse';
$dsn = 'mysql:host=localhost;dbname=publication';

$dbh = new PDO($dsn, $user, $pass, array(
    PDO::ATTR_PERSISTENT => true));

?>
```

Les connexions persistantes sont sauvegardées en mémoire par chaque processus. Il est donc inutile de les utiliser avec PHP en ligne de commande ou en CGI, car le processus se termine avec l'exécution (la connexion serait fermée à ce moment-là).

Sur un serveur de type Apache 1.3, chaque thread (processus Apache) contient sa propre réserve de connexions, qu'il ne partage pas avec les autres. Vous aurez donc a priori un nombre de connexions ouvertes correspondant à votre nombre de processus Apache multiplié par le nombre de couples login/base utilisés pour vous connecter. Sur un serveur mutualisé, ce nombre peut être très important et faire écrouler le serveur. Sur ce type d'architecture (un nombre important d'utilisateurs SGBD différents), utiliser des connexions classiques consommera moins de ressources.

#### Les connexions persistantes et ODBC

Si vous utilisez le driver PDO ODBC et que votre bibliothèque ODBC supporte le pool de connexion ODBC, alors il est recommandé de ne pas utiliser les connexions persistantes PDO mais de laisser le pool de connexion ODBC mettre en cache les connexions.

Le pool de connexion ODBC est partagé avec les autres modules dans le processus ; si PDO met en cache la connexion, alors celle-ci ne sera jamais retournée par le pool de connexion ODBC, faisant que plusieurs connexions seront créées pour les autres modules.

## Gérer les erreurs de connexion

Dans le cas où le pilote que vous avez spécifié ne peut être chargé ou que la connexion ne peut s'effectuer, une exception PDOException est lancée. Ainsi, vous pouvez décider de la meilleure façon de gérer l'erreur (ou de ne pas la gérer, ce qui arrêtera l'exécution).

```
<?php
$user = 'cyril';
$pass = 'motdepasse';
$dsn = 'mysql:host=localhost;dbname=publication';
```

```
try {
    $dbh = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Vous trouverez plus loin dans ce chapitre des informations sur la gestion des erreurs.

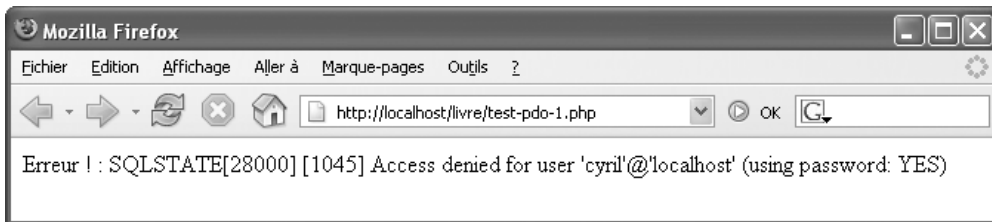


Figure 18-6

Exemple d'erreur renvoyée par PDO

### Sécurité, la gestion des traces

Si votre application n'intercepte pas les exceptions lancées depuis le constructeur PDO, l'action par défaut du moteur PHP est de terminer le script et d'afficher une trace. Cette trace pourrait révéler des détails complets sur la connexion à la base de données, incluant le nom d'utilisateur et le mot de passe. Il en est donc de votre responsabilité de gérer cette exception, soit explicitement (via l'instruction `catch`) ou implicitement via la fonction `set_exception_handler()`.

## Fermer une connexion

Lorsque la connexion à la base de données a réussi, une instance de la classe PDO est retournée à votre script. La connexion est active tant que l'objet PDO l'est. Pour clore la connexion, vous devez détruire l'objet en vous assurant que toutes ses références sont effacées. Vous pouvez faire cela en assignant NULL à la variable gérant l'objet. Si vous ne le faites pas explicitement, PHP fermera automatiquement la connexion lorsque le script arrivera à la fin.

```
<?php
$user = 'cyril';
$pass = 'motdepasse';
$dsn = 'mysql:host=localhost;dbname=publication';
try {
    $dbh = new PDO($dsn, $user, $pass);
```

```
// Utilisation de la connexion
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

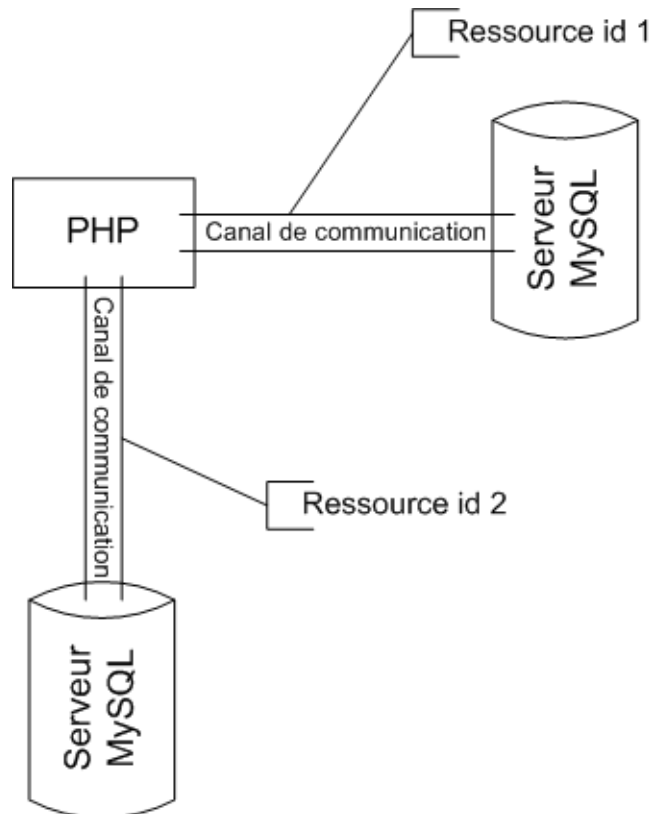
// Actions

if ($dbh) {
    $dbh = NULL ; // Fermeture de la connexion
}
?>
```

### Se connecter à plusieurs bases de données

Si vous ouvrez plusieurs connexions sur des bases différentes, vous aurez besoin de créer plusieurs instances de la classe PDO. Dans ce cas, il vous sera nécessaire de savoir sur quelle base appliquer vos prochaines requêtes.

**Figure 18-7**  
*Connexion à plusieurs bases de données*



Exemple de connexion à plusieurs bases de données :

```
<?php
$user1 = 'cyril';
$pass1 = 'motdepasse';
$dsn1 = 'mysql:host=localhost;dbname=publication';
try {
    $dbh1 = new PDO($dsn1, $user1, $pass1);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

$user2 = 'eric';
$pass2 = 'sonmotdepasse';
$dsn2 = 'mysql:host=localhost;dbname=formation';
try {
    $dbh2 = new PDO($dsn2, $user2, $pass2);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

?>
```

### Créer un fichier de configuration

Vous allez utiliser des connexions aux bases de données en plusieurs points de votre application. La solution impliquant de définir sur chaque page le login, le mot de passe et le serveur de votre base de données est mauvaise, car il est fréquent de devoir changer ces valeurs.

Vous aurez tout intérêt à mettre ces différentes valeurs dans une fonction dédiée ou dans un fichier de configuration. Une méthode simple, si vous n'avez pas de gestion de configuration, est de définir un fichier contenant les paramètres de connexion et l'instanciation de la classe PDO. Il suffira de l'inclure au début de chaque script utilisant MySQL :

```
<?php
define('USER1', 'cyril');
define('PASS1', 'motdepasse');
define('DSN1', 'mysql:host=localhost;dbname=publication');
try {
    $dbh = new PDO(DSN1, USER1, PASS1);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

?>
```

## Effectuer une requête

Une fois votre connexion ouverte, vous allez pouvoir utiliser pleinement votre base de données, pour lire, modifier ou encore supprimer des données. Pour cela, il n'existe pas de fonctions de lecture/écriture comme pour la gestion des fichiers. On utilise directement le langage SQL dont les principes généraux ont été décrits au chapitre précédent.

Pour envoyer une requête au serveur, on peut utiliser deux méthodes de l'objet PDO : `exec()` et `query()`.

### Requêtes préparées

Il est possible également d'utiliser les requêtes préparées qui offrent plus de sécurité mais qui sont légèrement plus lentes dans le cas de requêtes unitaires (par opposition à des requêtes identiques effectuées plusieurs fois avec des paramètres différents). Nous aborderons ce point plus loin dans ce chapitre.

Quand vous exécutez une requête avec les méthodes `query()` ou `exec()`, vous ne faites qu'envoyer votre ordre à votre base de données. Il faut ensuite traiter le résultat. Pour cela, nous distinguerons deux cas :

- après une requête de sélection qui renvoie des résultats ;
- après une requête d'insertion/modification.

Pour une requête ne renvoyant pas de résultats à proprement parler (UPDATE, INSERT...), il faut utiliser la méthode `exec()` qui retourne le nombre de lignes concernées par la requête.

La méthode `exec()` permet d'exécuter une requête mais ne renvoie que le nombre de lignes modifiées : on s'en servira généralement pour faire des insertions, des modifications ou des suppressions.

```
■ Nbe de lignes affectées B exec (requête_sql)
```

Pour une requête renvoyant des résultats (SELECT, DESC, SHOW ou EXPLAIN), il faudra utiliser la méthode `query()` qui retourne une instance de l'objet `PDOStatement` contenant les résultats que vous pourrez réutiliser par la suite pour les lire.

```
■ Instance de la classe PDOStatement B query (requête_sql)
```

La méthode `query()` permet de récupérer des données. Elle renvoie une instance de la classe `PDOStatement`.

```
■ $resultat = $dbh->exec( $sql );
```

Tableau 18-1 La méthode PDO appropriée pour chaque instruction SQL

Requête SQL	Méthode PDO à utiliser
INSERT	exec()
UPDATE	exec()
DELETE	exec()
SELECT	query()
EXPLAIN	query()
SHOW	query()
DESC	query()

## Requêtes invalides

Dans le cas où la requête ne fonctionne pas, les méthodes `query()` et `exec()` renvoient `FALSE`. Cela peut arriver quand elle est mal formée ou quand l'utilisateur ne dispose pas des droits suffisants pour l'effectuer.

```
if($dbh->exec($sql) === FALSE){
    echo 'Il y a une erreur dans votre requête sql :';
    echo $sql ;
    exit() ;
}

if($dbh->query($sql) === FALSE){
    echo 'Il y a une erreur dans votre requête sql :';
    echo $sql ;
    exit();
}
```

### Attention

Notez que nous utilisons « `===` » et `FALSE` plutôt que le comparateur de valeur (et non de type) « `==` ». Étant donné qu'une requête ne renvoyant pas de résultat retournera 0, il faut y faire attention.

## Requête de sélection

Pour toutes les requêtes renvoyant des données autres que le nombre d'enregistrements concernés, il faut utiliser la méthode `query()` de l'objet PDO correspondant.

Après l'exécution d'une requête de sélection, les données ne sont pas affichées, elles sont simplement mises en mémoire. Il faut donc aller les chercher et les afficher.

La méthode `query()` vous renvoie une instance de la classe `PDOStatement`. Cette dernière dispose de deux méthodes qui permettront de manipuler les données renvoyées :



- La méthode `fetchAll()` retourne l'ensemble des données sous forme d'un tableau PHP et libère le SGBD. Vous accéderez alors directement à toutes les données et pourrez exécuter des requêtes tierces pendant l'analyse du résultat. Le contre-coup de cette facilité d'utilisation est une charge importante au niveau du serveur. La totalité des données seront en effet localisées en mémoire.
- La méthode `fetch()` permet une lecture séquentielle du résultat. À un instant  $t$ , vous ne lisez qu'un seul résultat et la mémoire du système n'est pas occupée avec les autres entrées. Cette méthode est très utile pour le traitement de gros résultats. Son désavantage est que vous ne pourrez pas faire d'autres requêtes sur la même connexion PDO pendant le traitement de ce résultat. Vous n'aurez pas non plus accès aux informations comme le nombre de lignes résultat avant d'avoir parcouru l'intégralité dudit résultat.

```

PDOStatement::fetchAll ( [fetch_style] )

PDOStatement::fetch ( [fetch_style
                    [,cursor_orientation [,cursor_offset]]] )

```

### Choisir le format des résultats

Le paramètre `fetch_style` détermine la façon dont PDO retourne les résultats. Il permet de définir de quel type sera le retour : tableau associatif, tableau numériquement indexé, objet.

**Tableau 18-2 Les valeurs de l'attribut `fetch_style`**

Valeur	Action
<b>PDO::FETCH_ASSOC</b>	Retourne un tableau associatif indexé par le nom de la colonne, comme retourné dans le jeu de résultats.
<b>PDO::FETCH_BOTH (par défaut)</b>	Retourne un tableau indexé par les noms de colonnes mais aussi par les numéros de colonnes (commençant à l'indice 0), comme retournés dans le jeu de résultats.
<b>PDO::FETCH_OBJ</b>	Retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats.

```

<?php
// Définition des variables de connexion
$user = 'cyril';
$pass = 'motdepasse';
$dsn = 'mysql:host=localhost;dbname=publication';

// Connexion à la base de données
try {
    $dbh = new PDO($dsn, $user, $pass);
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

```

```
// Lecture d'enregistrements
$sql = "SELECT login, nom FROM auteur LIMIT 0,1";
$stmt = $dbh->query($sql);

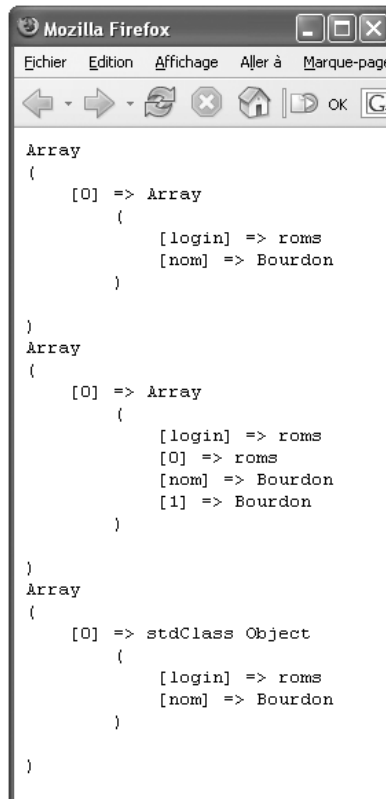
$result = $sth->fetchAll(PDO::FETCH_ASSOC);
print_r($result);

$stmt = $dbh->query($sql);
$result = $sth->fetchAll(PDO::FETCH_BOTH);
print_r($result);

$stmt = $dbh->query($sql);
$result = $sth->fetchAll(PDO::FETCH_OBJ);
print_r($result);
?>
```

**Figure 18-8**

*Les principaux styles de présentation des résultats*



```
Array
(
    [0] => Array
        (
            [login] => roms
            [nom] => Bourdon
        )
)
Array
(
    [0] => Array
        (
            [login] => roms
            [0] => roms
            [nom] => Bourdon
            [1] => Bourdon
        )
)
Array
(
    [0] => stdClass Object
        (
            [login] => roms
            [nom] => Bourdon
        )
)
```

PDO::FETCH\_ASSOC

PDO::FETCH\_BOTH  
(défaut)

PDO::FETCH\_OBJ

## Lire tous les enregistrements

Tous les enregistrements sont renvoyés dans un tableau par la méthode `fetchAll()`. Il suffit donc de parcourir le tableau en affichant son contenu.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

// Lecture d'enregistrements
$sql = "SELECT login, nom, prenom FROM auteur";
$stmt = $dbh->query($sql);
$result = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($result as $row){
    echo $row['nom']; echo '-';
    echo $row['prenom']; echo '-';
    echo $row['login']; echo '<br/>';
}
// Fermeture de la connexion
$dbh = NULL;

?>
```

**Figure 18-9**

*Affichage des enregistrements retournés*



## Nombre d'enregistrements retournés

Si vous voulez savoir combien d'enregistrements sont concernés par une requête de sélection, vous avez deux possibilités :

- créer une requête spécifique utilisant la fonction `COUNT()` de MySQL ;
- compter le nombre d'éléments contenus dans le tableau renvoyé par la méthode `fetchAll()`.

Le premier cas sera le plus adapté si vous n'avez pas besoin de traiter les données ensuite. Dans le cas contraire, la seconde possibilité sera plus appropriée.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');
```

```
// En utilisant une requête particulière
$sql = "SELECT count(*) as nbe FROM rmq WHERE pseudo='Adam'";
$sth = $dbh->query($sql);
$result = $sth->fetchAll();
$nombre = $result[0]['nbe'];
echo $nombre;

/* En comptant le nombre d'éléments présents dans le tableau de résultats */
$sql = "SELECT pseudo, texte FROM rmq WHERE pseudo='Adam'";
$sth = $dbh->query($sql);
$result = $sth->fetchAll();
$nombre = count($result);
echo $nombre;

?>
```

### Traiter les requêtes renvoyant beaucoup de résultats

La méthode `fetchAll()` permet de récupérer toutes les données résultant d'une requête dans un tableau. C'est la méthode la plus simple à utiliser, mais elle ne convient pas à tous les cas d'application. Notamment si la requête renvoie un grand nombre de résultats. Dans ce cas, on lui préférera la méthode séquentielle `fetch()` qui va aller chercher les enregistrements les uns après les autres.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

// En utilisant une requête particulière
$sql = "SELECT * FROM rmq";
$sth = $dbh->query($sql);

while($row = $sth->fetch(PDO::FETCH_ASSOC)){
    print_r($row);
}

?>
```

### Requête d'insertion / modification

Pour les requêtes d'insertion et de modification, on utilise la méthode `exec()` de PDO.

La méthode `exec()` permet d'exécuter une requête et ne renvoie que le nombre de lignes modifiées : on s'en servira généralement pour faire des insertions, des modifications ou des suppressions.

```
■ Nbe de lignes affectées B exec (requête_sql)
```

```
■ <?php
```

```
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

// Insertion d'un enregistrement
$sql = "INSERT INTO auteur (login, nom, prenom)
VALUES ('Alfredo', 'Bie', 'Alfred')";
$dbh->exec($sql);

?>
```

La méthode `exec()` retourne le nombre de lignes qui ont été modifiées ou effacées par la requête SQL exécutée. Si aucune ligne n'est affectée, la méthode `PDO::exec()` retournera 0.

Si la méthode `exec()` ne peut s'effectuer (à cause d'une mauvaise requête SQL par exemple) la valeur de retour sera `FALSE`.

#### Attention à ne pas confondre une valeur de retour 0 et FALSE

Quand votre requête SQL ne modifie aucune valeur dans la base de données, vous obtenez le nombre 0 en retour. Quand la méthode `exec()` ne fonctionne pas correctement, vous obtenez `FALSE` en retour. Attention donc à utiliser l'opérateur de comparaison « `===` » qui vérifie l'égalité de valeur ET de type. Ainsi, `FALSE` ne sera pas considéré de la même manière que 0.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = "DELETE FROM rmq WHERE pseudo='John'";

// Modification d'enregistrement
$retour = $dbh->exec($sql);

if($retour === FALSE){
    die('Erreur dans la requête') ;
}elseif($retour === 0){
    echo 'Aucune modification effectuée';
}else{
    echo $retour . ' lignes ont été affectées.';
}
?>
```

Il existe toutefois une exception. Lorsque vous exécutez une commande `DELETE` sans clause `WHERE`, tous les enregistrements sont effacés : pour optimiser cette requête, MySQL supprime le fichier et le recrée immédiatement. Vous ne pourrez donc pas connaître le nombre d'enregistrements supprimés.

## Connaître l'identifiant de la dernière ligne insérée ou la valeur d'une séquence

Quand vous utilisez des séquences ou des champs auto-incrémentés (par exemple avec MySQL), il est utile de pouvoir connaître l'identifiant de la dernière ligne insérée. Pour ce faire, on peut utiliser la méthode `lastInsertId()`.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

// Insertion d'un enregistrement
$sql = "INSERT INTO article (titre, auteur)
        VALUES ('PHP 5 avancé - V3','cyruiss')";
$dbh->exec($sql);

echo $dbh->lastInsertId();

?>
```

## Sécurité et échappements

Comme tous les langages, SQL a ses propres caractères spéciaux et délimiteurs. Ainsi, une chaîne contenant une apostrophe peut faire dérailler le SGBD qui l'interprétera comme une fin de chaîne de caractères, et non pas comme une apostrophe à l'intérieur d'une chaîne de caractères. Pour utiliser réellement une apostrophe, il faudra lui appliquer une transformation que l'on nomme l'échappement. Le plus souvent il s'agit de préfixer le caractère à échapper par un caractère antislash, « \ », ou de doubler l'apostrophe (").

### Problématique de l'échappement

La problématique de l'échappement est la même avec PHP. Une chaîne entre apostrophes ne peut contenir une apostrophe que si elle est échappée, avec \. On peut retrouver aussi le problème (mais avec d'autres caractères) avec les langages HTML et XML.

PDO propose la méthode `quote()` de l'objet de connexion pour effectuer cette opération. Elle prend en argument une chaîne de caractères et la formate de façon à pouvoir l'utiliser directement dans une requête SQL : des délimiteurs de chaîne sont insérés autour et les caractères spéciaux sont échappés.

```
$nom = "PIERRE de GEYER d'ORTH";
$nom = $dbh->quote($nom);

// Insertion d'un enregistrement
$sql = "INSERT INTO auteur (login, nom)
        VALUES ('Cyruss6', $nom)";
$dbh->exec($sql);
```

Dans notre exemple précédent, l'apostrophe du nom « PIERRE de GEYER d'ORTH » est protégée pour l'insertion.

Oublier d'échapper des caractères spéciaux peut entraîner de graves problèmes de sécurité (on parle souvent de faille par injection SQL). Vous avez la responsabilité de penser à échapper toutes les chaînes de caractères sans exception avant de les envoyer dans une requête. Consultez le chapitre 27 sur la sécurité pour plus d'informations sur les attaques par injection SQL.

Tôt ou tard, un développeur oubliera de faire les échappements nécessaires pour une requête SQL. Pour éviter le problème, PDO vous propose d'utiliser des requêtes dites « paramétrées ». Les paramètres (nombres, chaînes de caractères) n'ont alors pas besoin d'échappement. Nous vous recommandons très fortement de ne pas entrer directement de données dans vos requêtes SQL et de toujours passer par des requêtes paramétrées. Le confort et la sécurité seront bien plus importants. La description de ce mécanisme est détaillée plus loin dans ce chapitre.

**Pour éviter tout problème, utilisez les requêtes préparées**

Les requêtes préparées vous permettent de savoir exactement quelle est la forme des requêtes qui doivent être exécutées. C'est une solution à envisager autant que possible.

**Note sur magic\_quotes\_gpc**

Dans certaines configurations de PHP, les entrées utilisateur (GET, POST, COOKIES) sont automatiquement filtrées lors de leur interprétation, et ce pour éviter les problèmes d'injection SQL (la notion d'injection SQL est traitée plus en détail au chapitre 27). La fonction `addslashes()` est alors automatiquement appliquée à toute donnée qui vient de l'utilisateur (elle ajoute un antislash devant tous les caractères spéciaux SQL). Ce mécanisme est guidé par la directive `magic_quotes_gpc` du fichier `php.ini`. Vous trouverez plus de renseignements à ce sujet au chapitre 2 concernant l'installation et la configuration de PHP et au chapitre 8 concernant les formulaires.

L'activation de ce mécanisme pose presque autant de problèmes qu'il n'en résout. Il est donc peu utilisé sur les configurations récentes. Nous vous conseillons de le désactiver et considérons dans ce chapitre qu'il est désactivé.

Si ce n'est pas votre cas et que vous ne contrôlez pas votre configuration, vous pouvez en annuler les effets à l'aide des quelques lignes suivantes à placer en tout début de chaque script :

```
if (get_magic_quotes_gpc()) {
    array_walk_recursive($_GET, 'stripslashes');
    array_walk_recursive($_POST, 'stripslashes');
    array_walk_recursive($_COOKIE, 'stripslashes');
    array_walk_recursive($_REQUEST, 'stripslashes');
}
```

Vous pouvez également agir en fonction de cette directive de configuration via la fonction `get_magic_quote_gpc()`. L'exemple suivant vous montre comment échapper ou non une valeur fournie par un utilisateur.

```
if (!get_magic_quotes_gpc()) {
    $lastname = $pdo->quote($_POST['lastname']);
} else {
    $lastname = $_POST['lastname'];
}
```

## Gestion des erreurs

Selon la base de données que vous utilisez, la gestion des erreurs peut être très différente. Certaines bases de données disposent d'un support très riche et d'autres n'affichent presque aucune information en cas d'erreur. PDO utilise un code d'erreur unifié pour vous faciliter un éventuel changement de SGBD. Bien entendu, PDO vous donne aussi accès aux codes et aux messages d'erreurs natifs associés à la base que vous utilisez.

D'autres part, avec PDO, vous pouvez définir le déclencheur d'erreurs. Vous pouvez demander à PDO :

- de ne pas afficher les erreurs (défaut) ;
- d'utiliser le mode d'erreur classique (lance une erreur de niveau `E_WARNING`) ;
- d'utiliser les exceptions.

Par défaut, PDO utilise le mode silencieux ; les erreurs sont cependant stockées et consultables en faisant appel aux méthodes `errorCode()` et `errorInfo()`.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = "";
if (!$dbh->exec($sql)) {
    echo $dbh->errorCode() . "<br>";
    $info = $dbh->errorInfo();
    print_r($info);
    // $info[0] == $dbh->errorCode() Code d'erreur unifié
    // $info[1] code d'erreur spécifique au driver
    // $info[2] message d'erreur spécifique au driver
}
?>
```

Pour changer le gestionnaire d'erreur lié à PDO, on fait appel à la méthode `setAttribute()` de PDO :

```
// Mode silencieux
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);

// Mode erreur classique
```



```
$dbh->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_WARNING);  
  
// Mode exception  
$dbh->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
```

## Utiliser les exceptions

Pour utiliser les exceptions afin de gérer les erreurs avec PDO, il faut donner à l'attribut `PDO::ATTR_ERRMODE` la valeur `PDO::ERRMODE_EXCEPTION` :

```
$dbh->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
```

Il vous suffira ensuite d'utiliser `try{} et catch(){} pour gérer les exceptions lancées.`

```
<?php  
// Inclusion du fichier contenant la connexion à la base  
include_once('connect.inc.php');  
  
// On définit le handler d'erreur  
$dbh->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);  
  
try {  
    $sql = "INSERT INTO produit  
           VALUES (NULL,'CB500', 'Honda', '6000')";  
  
    $dbh->exec($sql);  
    // Si une erreur a eu lieu une exception est lancée.  
}  
catch (PDOException $e){  
    print "Erreur ! : " . $e->getMessage() . "<br/>";  
}  
?>
```

## Gestion des transactions

Une transaction correspond à un bloc d'instructions que l'on souhaite unies. L'ensemble des requêtes doit être exécuté, sans quoi tout est annulé : on parle d'atomicité. Ce doit être le cas par exemple dans les transactions bancaires : on n'imagine pas que votre compte soit débité sans que votre achat soit crédité au vendeur, les deux requêtes sont dépendantes l'une de l'autre.

Par défaut, les transactions sont désactivées ou, plus précisément, le mode `auto-commit` est activé. Ainsi, toutes les requêtes SQL sont exécutées lors de l'appel aux méthodes `query()` et `exec()`.

### Les transactions et MySQL

Pour utiliser les transactions avec MySQL, il est nécessaire d'utiliser un moteur de stockage de type InnoDB.

Pour utiliser les transactions, il convient de respecter le schéma suivant :

1. indiquer à PDO que l'on souhaite commencer une transaction avec la méthode `beginTransaction()` ;
2. valider la transaction avec la méthode `commit()` ou l'annuler avec la méthode `rollback()`.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

// On définit le gestionnaire d'erreur en mode 'exception'
$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Démarre une transaction, désactivation de l'auto-commit
$dbh->beginTransaction();
try {
    // Ajout d'un enregistrement
    $sql = "INSERT INTO auteur (login, nom, prenom)
          VALUES ('Max','HaveIard', 'isGood')";
    $dbh->exec($sql);

    // Ajout d'un second enregistrement dépendant du premier
    // syntaxe incorrecte dans cet exemple : titre
    $sql2 = "INSERT INTO article (titre, texte, auteur)
            VALUES ('PHP 5 V3 !','il est sorti','Max')";
    $dbh->exec($sql2);

    // Si les requêtes se sont bien passées, on valide
    // Sinon une exception a été lancée
    $dbh->commit();

} catch (Exception $e){
    // S'il y a eu une erreur, on annule les modifications
    $dbh->rollback();
    echo "Échec: " . $e->getMessage();
}

/* La connexion à la base de données est maintenant de retour en mode auto-commit */
?>
```

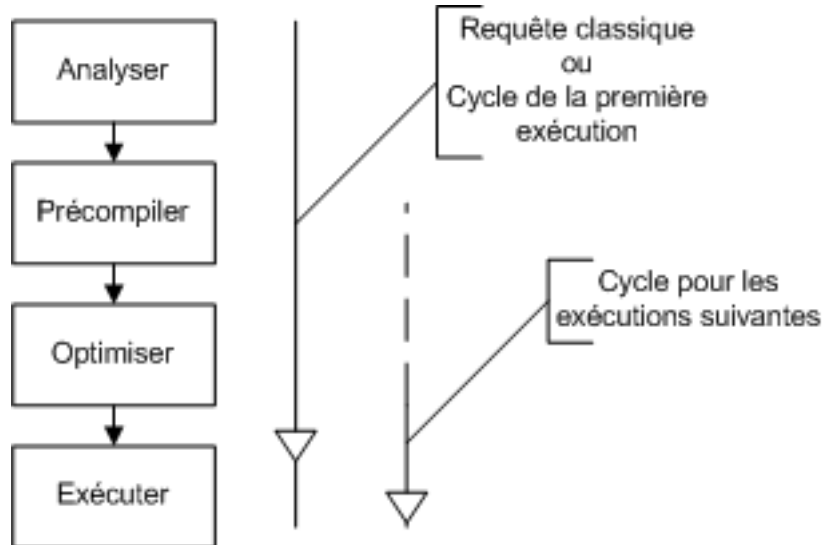
Dans notre exemple, nous avons utilisé les exceptions comme mode de gestion d'erreur de PDO. Si vous le souhaitez, vous pouvez travailler avec un mode d'erreur classique, pour cela il vous suffit de l'indiquer avec la méthode `setAttribute()`. Consultez la partie gestion des erreurs pour plus de détails.

## Les requêtes préparées

Le principe des requêtes préparées est de créer un modèle de requête et de l'enregistrer sur le SGBD, le temps de l'exécution du script (par opposition aux procédures stockées qui le sont de manière permanente). Quand vous aurez besoin de faire une requête, vous ferez appel à ce modèle. Le serveur exécutera alors votre requête, en utilisant les données que vous lui aurez fournies en paramètres pour construire la requête SQL réelle.

Figure 18-10

*Comparaison  
requête classique vs  
requête préparée*



Les requêtes préparées sont recommandées pour :

- Les requêtes multiples : la requête n'est interprétée qu'une seule fois mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents. Quand la requête est exécutée, la base de données va l'analyser, la compiler et l'optimiser. Pour des requêtes complexes, ces étapes peuvent prendre du temps et ralentir votre application si vous devez les répéter. En utilisant des requêtes préparées, vous éviterez de répéter le cycle d'analyse / compilation / optimisation.
- Protéger vos requêtes : les paramètres des requêtes préparées n'ont pas besoin d'être protégés, le pilote de votre base de données le fait tout seul. Vous vous protégez donc des attaques par injection SQL.

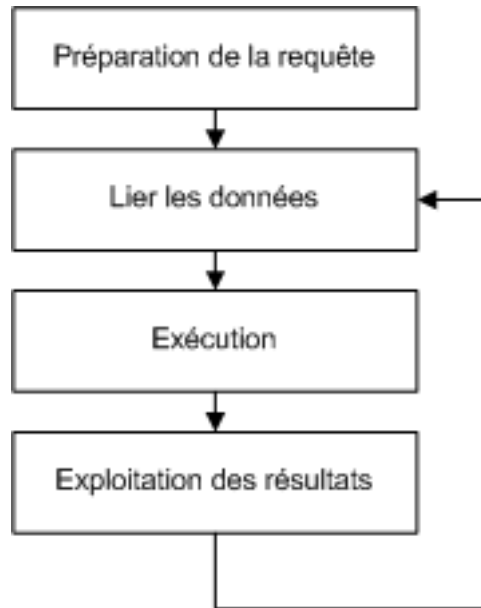
### **PDO émule les requêtes préparées**

PDO émule les requêtes préparées si votre base de données ne dispose pas de cette fonctionnalité. Ainsi, vous êtes assuré de pouvoir travailler de la même façon quel que soit le SGBD.

**Attention !** Il n'est pas possible de préparer deux requêtes de manière parallèle. Vous devrez fermer la requête en cours pour en utiliser une nouvelle. Si vous souhaitez pouvoir exploiter deux résultats simultanément, vous devrez enregistrer les premiers résultats dans un tableau, puis exécuter ensuite la seconde requête.

D'un point de vue pratique, cela va correspondre aux étapes suivantes :

**Figure 18-11**  
*Principe des requêtes préparées*



Désavantage des requêtes préparées :

- Si vous n'utilisez qu'une seule fois votre requête, son temps d'exécution sera très légèrement plus long. Par contre, l'avantage que vous en retirerez sera la sécurité du code.

## Construction de la requête

Pour construire un modèle de requête, il suffit de remplacer chaque paramètre par un point d'interrogation ou par un paramètre nommé (par exemple « :nom »). Vous fournirez les paramètres à substituer quand vous exécuterez réellement la requête.

```
// Requête normale
$sql = "INSERT INTO article (titre, auteur)
      VALUES ('Titre super','auteur sympa)";

// Modèle de requête avec des paramètres nommés
$sql2 = 'INSERT INTO article (titre, auteur)
        VALUES ( :titre , :auteur)';
```

```
// Modèle de requête avec des points d'interrogations
$sql3 = 'INSERT INTO article (titre, auteur)
        VALUES ( ?, ?)';
```

#### Il faut choisir

Il n'est pas possible d'utiliser à la fois des noms de paramètres (:nom) et des points d'interrogation. Vous ne pouvez pas non plus utiliser le même nom de paramètre plusieurs fois.

Un des avantages notables de cette méthode est que vous êtes protégé des attaques dites par injection SQL (voir le chapitre sur la sécurité). Le SGBD sait ce qu'il s'attend à recevoir ; il vérifiera que les données transmises sont correctes et fera les échappements nécessaires.

### Préparer une requête

Une fois le modèle de requête construit, vous devrez le fournir à votre SGBD pour qu'il l'analyse grâce à la méthode `prepare()` : on parle alors de préparation.

Cette méthode prend en argument le modèle de requête SQL et renvoie une instance de la classe `PDOStatement` qu'il faudra utiliser pour les futures opérations de traitement. En cas d'échec, la méthode renvoie `FALSE`.

Cette préparation permet de n'exécuter qu'une seule fois les analyses si vous utilisez un même modèle à plusieurs reprises dans le même script.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = 'INSERT INTO article (titre, auteur)
        VALUES ( :titre , :auteur)';

$stmt = $dbh->prepare($sql);
?>
```

### Lier des données à des paramètres et exécution

Une fois votre modèle de requête créé, il vous faudra le remplir. La façon la plus simple consiste à passer un tableau contenant les différentes valeurs à la méthode `execute()` de votre objet `PDOStatement`.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = 'INSERT INTO article (titre, auteur)
        VALUES ( :titre , :auteur)';
```

```
$stmt = $dbh->prepare($sql);

$titre = 'Memento PHP MySQL';
$auteur = 'Ponçon' ;

$stmt->execute(array(':titre'=>$titre, ':auteur'=>$auteur));
?>
```

L'autre approche, plus pointue, consiste à associer distinctement chaque paramètre à une variable ou à une valeur :

```
bindParam ( parametre, &variable [, type [, taille ]] )
bindValue ( parametre, variable [, type [, taille ]] )
```

Dans le cas de la méthode `bindParam()`, on lie une variable à un paramètre. Ainsi, entre deux exécutions, il ne sera nécessaire que de changer la valeur de la variable.

Tableau des différents types

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = 'INSERT INTO article (titre, auteur)
      VALUES ( :titre , :auteur)';

$stmt = $dbh->prepare($sql);

$titre = 'Memento PHP MySQL';
$auteur = 'Ponçon' ;

$stmt->bindParam(':auteur',$auteur) ;
$stmt->bindParam(':titre',$titre) ;
$stmt->execute();
// Un premier enregistrement a été inséré

$titre = 'Best practices PHP 5' ;
$stmt->execute();
?>
```

Dans le cas de la méthode `bindValue()`, on associe une valeur à un paramètre. Il est important de noter cette distinction entre une valeur fixée à un moment donné et une référence qui lie un paramètre à une variable.

Dans l'exemple précédent, si nous avons utilisé la méthode `bindValue()` au lieu de `bindParam()`, le second appel à `execute()` aurait eu les mêmes conséquences que le premier. Et donc le changement de valeur de `$titre` n'aurait rien changé.

### Séquences et champs auto-incrémentés

La gestion des séquences et des champs auto-incrémentés fonctionne de la même façon que pour le mode classique. Il suffit d'utiliser la méthode `lastInsertId()`.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$sql = 'INSERT INTO article (titre, auteur)
      VALUES ( :titre , :auteur)';

$stmt = $dbh->prepare($sql);

$titre = 'Memento PHP MySQL';
$auteur = 'Ponçon' ;

$stmt->execute(array(':titre'=>$titre, ':auteur'=>$auteur));

echo $dbh->lastInsertId();

?>
```

### Exploitation d'une requête de sélection

Après l'exécution d'une requête de sélection, les données ne sont pas affichées, elles sont simplement mises en mémoire. Il faut donc aller les chercher et les afficher.

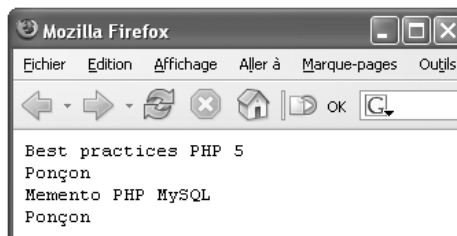
Le fonctionnement est relativement similaire au mode de requêtage classique : vous obtenez un tableau contenant les différentes lignes correspondant à votre résultat.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$auteur = 'Ponçon';
$sql = "SELECT * FROM article where auteur =:auteur";
$stmt = $dbh->prepare($sql);
$stmt->execute(array(':auteur'=>$auteur));
echo '<pre>';
while ($row = $stmt->fetch()) {
    echo $row['titre'],'<br/>';
    echo $row['auteur'],'<br/>';
}
?>
```

Figure 18-12

*Affichage du résultat  
d'une requête de  
sélection*



## Fermeture de la requête préparée

Comme pour la connexion au serveur MySQL, PHP arrête automatiquement la requête préparée à la fin du script. Toutefois, pour libérer des ressources, il est intéressant de pouvoir arrêter l'interprétation de cette requête quand on a fini de l'utiliser. Vous pouvez le faire en donnant la valeur NULL à votre objet PDOStatement.

```
<?php
// Inclusion du fichier contenant la connexion à la base
include_once('connect.inc.php');

$auteur = 'Ponçon';
$sql = "SELECT * FROM article where auteur =:auteur";
$stmt = $dbh->prepare($sql);
$stmt->execute(array(':auteur'=>$auteur));
echo '<pre>';
while ($row = $stmt->fetch()) {
    echo $row['titre'],'<br/>';
    echo $row['auteur'],'<br/>';
}
$stmt = NULL;

?>
```

## Cas d'application

### Gestion de publication

#### Contexte

Votre société dispose d'une petite cellule de recherche et développement (R&D). Jusqu'ici, toutes les publications sont basées sur un support papier, mais il est difficile d'optimiser la circulation des documents et de centraliser les commentaires des lecteurs.

Vous souhaiteriez automatiser la publication de données en profitant des nouvelles technologies en matière de système d'information. Votre souhait est que toutes les personnes présentes sur le réseau de l'entreprise puissent accéder aux documents et les commenter. En revanche, seuls les membres de la cellule recherche et développement auront les droits d'administration sur le système.

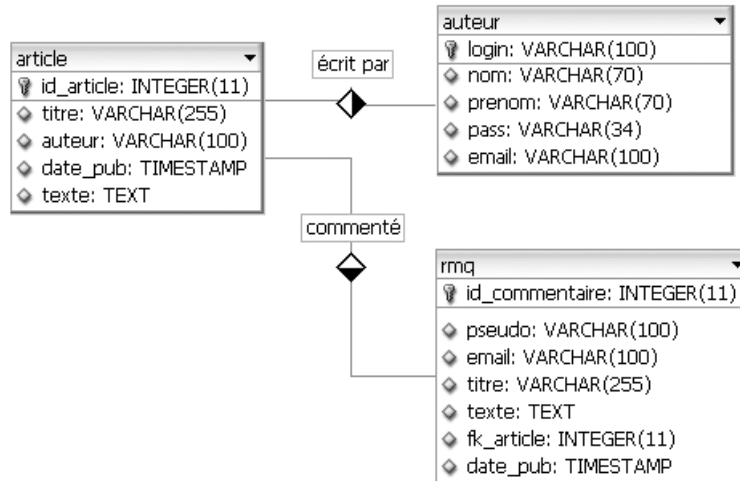
#### Réalisation et solution retenue

Pour créer votre système de publication, vous avez fait le choix d'utiliser un système simple composé des trois tables présentées dans la figure 18-13.



**Figure 18-13**

*MPD (Modèle Physique de Données) du système de publication*

**Remarque**

Ce schéma de base de données est volontairement très simple. Dans le cadre d'un développement réel, il conviendrait d'y ajouter des champs et des tables pour permettre de stocker des fichiers, de gérer des thèmes, de gérer des autorisations de lecture, etc.

Les actions possibles pour les utilisateurs seront :

- affichage de la liste de tous les articles par date, par auteur et par titre ;
- affichage d'un article ;
- sélection et affichage des commentaires relatifs à un article ;
- ajout d'un commentaire.

Les actions disponibles pour les gestionnaires de l'application seront :

- insertion d'un article ;
- modification d'un article ;
- suppression d'un article ;
- suppression d'un commentaire.

**Remarque**

Afin de garder les exemples aussi simples et clairs que possible, nous ne nous soucierons pas de la structure des fichiers HTML.

## Préparation

### Fichier de configuration

Nous allons commencer par créer un fichier de connexion afin de ne pas être obligé de refaire ces manipulations dans tous nos scripts. Ce fichier se nomme `connexion.inc.php` et sera appelé dans les scripts suivants.

```
<?php
define('USER1', 'cyril');
define('PASS1', 'motdepasse');
define('DSN1', 'mysql:host=localhost;dbname=publication');
try {
    $dbh1 = new PDO(DSN1, USER1, PASS1);
} catch (PDOException $e) {
    print "Erreur ! : " . $e->getMessage() . "<br/>";
    die();
}

?>
```

## Actions utilisateur







### Lister tous les articles

La requête SQL permettant de lister tous les articles est simple, mais dépend de ce que nous voulons faire du résultat. Si c'est pour tout afficher, il faut alors effectuer une sélection sur tous les champs de la table `article`. En revanche, si c'est uniquement pour afficher les titres, les auteurs et les dates avec un lien, il suffira de sélectionner ces champs. Nous allons privilégier ce second cas :

```
SELECT id_article, titre, auteur, date_pub FROM article
```

Figure 18-14

Exemple de résultat de la requête

← T →	id_article	titre	auteur	date_pub
<input type="checkbox"/>  	1	Memento PHP MySQL	Ponçon	2006-06-27 14:42:12
<input type="checkbox"/>  	6	Best practices PHP 5	Ponçon	2006-06-27 14:42:47
<input type="checkbox"/>  	9	PHP 5 avancé	cyruss	2006-06-27 14:50:14

Dans notre affichage, nous souhaitons classer les articles par date et limiter le nombre de résultats aux vingt premiers. Notre requête SQL finale serait alors :

```
SELECT id_article, titre, auteur, date_pub
FROM article ORDER BY date_pub DESC LIMIT 0,20
```

Nous allons maintenant afficher ces informations dans une page HTML, via le fichier `listing.php`. Le résultat est visible dans la figure 18-15.

```
<?php
// Connexion à la base de données
include_once('connexion.inc.php');
```

```
// Création de la requête SQL
$sql = 'SELECT id_article, titre, auteur, date_pub
FROM article ORDER BY date_pub DESC LIMIT 0,20';

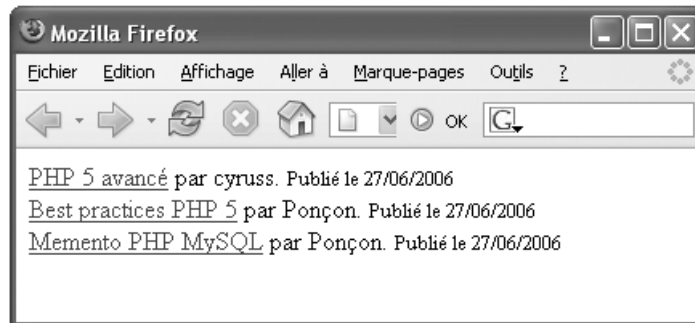
// Exécution de la requête SQL
$stmt = $dbh->query($sql);
$result = $sth->fetchAll();

// On boucle sur l'ensemble des enregistrements :
foreach ($result as $row){
    $titre = $row['titre'];
    $auteur = $row['auteur'];
    $date_pub = $row['date_pub'];
    $id_article = $row['id_article'];

    // On formate la date.
    $jour = substr($date_pub, 8, 2);
    $mois = substr($date_pub, 5, 2);
    $annee = substr($date_pub, 0, 4);
    $date = $jour.'/'.$mois.'/'.$annee;

    // On crée l'affichage
    echo "<a href='detail.php?id_article=$id_article'>$titre</a>";
    echo " par $auteur. <font size=2> Publié le $date</font><br>";
}
?>
```

**Figure 18-15**  
*Liste simple  
des articles*



Comme vous pouvez le remarquer, le lien pointe vers la page `detail.php` et prend en paramètre la clé de l'article sélectionné.

#### Afficher le détail d'un article

Pour afficher toutes les informations d'un article, il va nous falloir interroger deux tables : la table `article` pour en extraire tout son contenu, mais également la table `auteur` pour avoir le profil de l'écrivain. La solution basique consisterait à faire une première requête pour collecter toutes les informations sur l'article en question, à extraire le nom

de l'auteur (clé externe) et à faire une seconde requête sur la base pour obtenir toutes les informations le concernant. Cette méthode est lourde. Nous vous conseillons plutôt d'effectuer une seule requête un peu plus complexe, mais qui vous donnera toutes les informations en une seule passe.

Voici la requête permettant de tout collecter en une seule fois :

```
SELECT ar.id_article, ar.titre AS titre, ar.auteur AS auteur,
       ar.date_pub AS date_pub, ar.texte AS texte, au.login,
       au.nom AS nom, au.prenom AS prenom, au.email AS email
FROM article AS ar, auteur AS au
WHERE ( ( ar.id_article = 1 ) AND ( ar.auteur=au.login) )
```

Voici le code PHP correspondant à l'affichage de la figure 18-16.

```
<?php
// Connexion à la base de données
include_once('connexion.inc.php');

// On récupère l'identifiant passé en paramètre. Pour se protéger, on le force à être un entier.
$id_ar = (int) $_GET['id_article'] ;

// Création de la requête SQL
$sql = "
SELECT ar.id_article, ar.titre AS titre, ar.auteur AS auteur,
       ar.date_pub AS date_pub, ar.texte AS texte, au.login,
       au.nom AS nom, au.prenom AS prenom, au.email AS email
FROM article AS ar, auteur AS au
WHERE ( ( ar.id_article = $id_ar ) AND ( ar.auteur=au.login) )";

// Exécution de la requête SQL
$stmt = $dbh->query($sql);
$result = $stmt->fetchAll();

$row = $result[0];

$titre = $row['titre'];
$a_nom = $row['nom'];
$a_prenom = $row['prenom'];
$a_email = $row['email'];
$date_pub = $row['date_pub'];
$id_article = $row['id_article'];
$texte = nl2br($row['texte']);

// On formate la date
$jour = substr($date_pub, 8, 2);
$mois = substr($date_pub, 5, 2);
$annee = substr($date_pub, 0, 4);
$date = $jour.'/'.$mois.'/'.$annee;
```

```
// On crée l'affichage
echo "<h1>$titre</h1>";
echo "Cet article à été réalisé par $a_prenom $a_nom le $date<br>";
echo 'Vous pouvez le contacter par ' ;
echo "<a href='mailto:$a_email'>e-mail</a>";
echo '<br><b>Texte de l\'article</b><br>', $texte;
?>
```

Notez l'utilisation de la fonction `n12br()`, qui permet de transformer les sauts à la ligne textuels (`\n`) en retours à la ligne HTML (`<br>`).

**Figure 18-16**

*Affichage  
d'un article*



### Les commentaires liés à un article

Pour connaître le nombre de commentaires liés à un article, il suffit d'interroger la table `rmq` en lui indiquant le numéro de l'article. La requête suivante fera l'affaire :

```
SELECT count(*) as nb FROM rmq WHERE id_article = 1
```

Cela pourrait nous permettre de compléter le listing de la figure 18-16 en y ajoutant le nombre de commentaires.

Pour faciliter la consultation des commentaires, on décide de les afficher en bas de la page de l'article. Le résultat visible à la figure 18-17 s'obtient en ajoutant à la page `detail.php` le code suivant :

```
...
$sql2 = "SELECT * FROM rmq WHERE fk_article = $id_ar
ORDER BY date_pub DESC";
```

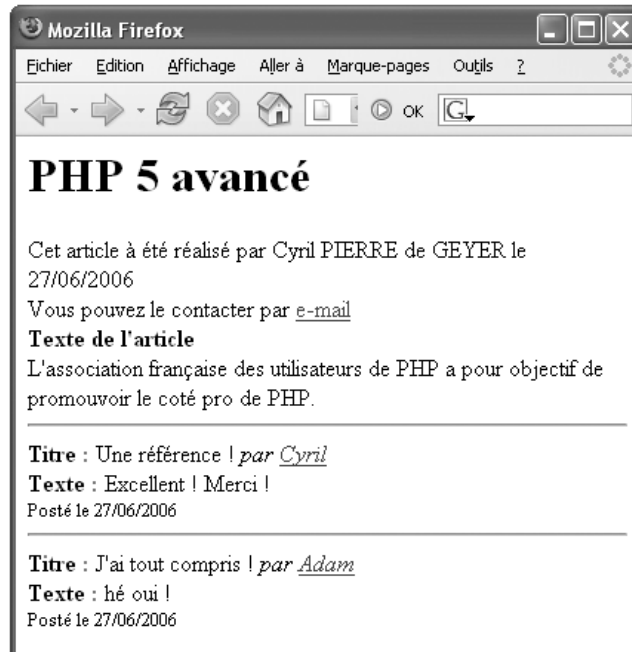
```
$sth = $dbh->query($sql2);
$result = $sth->fetchAll();

// On boucle sur l'ensemble des enregistrements
foreach ($result as $row){
    $pseudo = $row['pseudo'];
    $email = $row['email'];
    $titre = $row['titre'];
    $texte = nl2br($row['texte']);
    $date_pub = $row['date_pub'];

    // On formate la date
    $jour = substr($date_pub, 8, 2);
    $mois = substr($date_pub, 5, 2);
    $annee = substr($date_pub, 0, 4);
    $date = $jour.'/'.$mois.'/'.$annee;

    // On crée l'affichage
    echo "<hr><b>Titre :</b> $titre <i>";
    echo " par <a href='mailto:$email'$pseudo</a></i>";
    echo "<br><b>Texte : </b>$texte <br>";
    echo "<font size=2> Posté le $date</font><br>";
}
?>
```

**Figure 18-17**  
*Affichage des  
commentaires liés à  
l'article courant*



## Action des gestionnaires

Notons tout d'abord qu'il vous incombe de sécuriser l'accès aux pages suivantes. Pour cela, vous pouvez vous reporter au cas d'application du chapitre 11 concernant les sessions.

### Insertion d'un article

Pour insérer un article dans la base de données, il faut recevoir en entrée les informations. Il serait tout à fait possible de traiter des fichiers XML ou des entrées quelconques ; cependant, dans notre cas, ce sont les gestionnaires qui mettront directement l'information dans un formulaire. Les deux seules informations nécessaires à y insérer sont le titre et le texte de l'article. Les autres informations s'obtiennent en fonction du contexte. Plaçons le script suivant dans le fichier `insert_article.php`.

```
<form action='insert_article2.php' method='POST'>
  <input type='text' name='titre' size='49' value=''><br>
  <textarea name='texte' cols='37' rows='6'>
</textarea><br>
  <input type='submit' value='Valider'>
</form>
```

Une fois le formulaire en place et validé, il faut traiter les informations dans le fichier `insert_article2.php`.

Étant donné que les insertions SQL impliquent plusieurs données utilisateurs, nous utiliserons les requêtes préparées qui vont nous permettre de protéger au maximum l'application.

```
<?php
// Connexion à la base de données
include_once('connexion.inc.php');

$sql = 'INSERT INTO article (titre, auteur, texte)
        VALUES ( :titre , :auteur, :texte)';

$stmt = $dbh->prepare($sql);

$titre = $_POST['titre'];
$texte = $_POST['texte'];

$valeurs = array(':titre'=>$titre,
                 ':auteur'=>$auteur,
                 ':texte'=>$texte) ;

$stmt->execute($valeurs);
$stmt->execute();
?>
```

## Modification d'un article

Pour modifier un article, on commence par en récupérer le contenu. Une fois cette étape de sélection franchie, on crée un formulaire contenant comme valeurs par défaut les valeurs de l'article. On va créer le fichier `modif_article.php` et y insérer le code suivant :

```
<?php
// Connexion à la base de données
include_once('connexion.inc.php');

// On récupère l'identifiant passé en paramètre dans l'article
$id_ar = (int) $_GET['id_article'];

// création de la requête SQL
$sql = "SELECT titre,texte FROM article WHERE id_article = $id_ar";

// Exécution de la requête SQL
$sth = $dbh->query($sql);
$result = $sth->fetchAll();

// On boucle sur l'ensemble des enregistrements :
$row = $result[0];

$titre = $row['titre'];
$texte = $row['texte'];

// On formate la date
$jour = substr($date_pub, 8, 2);
$mois = substr($date_pub, 5, 2);
$annee = substr($date_pub, 0, 4);
$date = $jour.'/'.$mois.'/'.$annee;

?>
<form action='modif_article2.php' method='POST'>
<input type='text' name='titre' size='49'
value='<?php echo $titre; ?>'><br>
<textarea name='texte' cols='37' rows='6'>
<?php echo $texte;?>
</textarea><br>
<input type='hidden' name='id_article'
value='<?php echo $id_ar ; ?>'>
<input type='submit' value='Valider'>
</form>
```

Pour accéder à ce script, il convient de créer un espace où vous listez les articles existants, en mettant un lien pour chacun d'entre eux dans lequel vous passez l'identifiant de l'article.

Ce formulaire pointe sur le fichier `modif_article2.php`, qui va modifier effectivement l'information dans la base de données.

```
<?php
// Connexion à la base de données
include_once 'connexion.inc.php';
```



```
// On récupère et traite les données
$titre = $_POST['titre'];
$texte = $_POST['texte'];
$id_article = (int) $_POST['id_article'];

// Création de la requête SQL
$sql = "UPDATE article
      SET titre=:titre, texte=:texte
      WHERE id_article = :id_article";

$stmt = $dbh->prepare($sql);

$valeurs = array(':titre'=>$titre,
                 ':texte'=>$texte,
                 ':id_article'=>$id_article);

$stmt->execute($valeurs);
$stmt->execute();

?>
```

### Suppression d'un article ou d'un commentaire

Pour supprimer un article ou un commentaire, il vous suffit de faire appel au script suivant en lui passant en paramètre l'identifiant de l'article ou du commentaire à supprimer.

```
<?php
// Connexion à la base de données
include_once 'connexion.inc.php';

$sql = "DELETE FROM article WHERE id_article=:article";

$stmt = $dbh->prepare($sql);

$id_article = $_GET['id_article'];

$stmt->execute(array(':article'=>$id_article));

$stmt->execute();?>
```

Cet exemple ne prend pas en compte une éventuelle validation de l'utilisateur quant à la suppression de l'article. Généralement, on utilise une page intermédiaire pour ce type de confirmation.

# 19

## Erreurs et exceptions

---

Une bonne gestion des erreurs est l'un des points qui démarquent un bon développeur d'un simple amateur. Une telle implémentation est un signe de sérieux et un gage de réussite. Effectivement, les erreurs sont le signal d'alarme d'un mauvais fonctionnement ou d'une incohérence dans votre logique. Être averti d'une erreur vous permettra de la traquer, de déceler une faille de sécurité, de repérer un défaut de configuration, voire d'être averti de l'arrêt d'un service. Il convient donc d'y prêter une attention rapportée à la complexité et à l'importance de votre développement. Nous allons voir dans ce chapitre les différentes erreurs PHP, comment les gérer, les intercepter et dans quelle mesure on peut configurer leur gestion. Nous nous attacherons ensuite à définir ce que sont les assertions et en quoi elles peuvent sécuriser une application. Enfin nous étudierons une des grandes nouveautés de PHP 5 : la gestion des exceptions.

### Explications sur les erreurs

#### *Qu'est-ce qu'une erreur ?*

Avant de décrire le fonctionnement des erreurs internes et les détails de gestion, il est utile de préciser ce que l'on regroupe sous le terme « erreur ».

Nous utiliserons donc ce mot de la façon suivante : *une erreur est une différence entre ce que devrait faire l'application et ce qu'elle fait réellement*. On peut aussi parler plus simplement d'un comportement anormal de l'application. Le fait qu'un événement soit qualifiable d'erreur ou non n'est lié ni à sa fréquence, ni à son caractère prévisible, ni à ses conséquences.

## ***Pourquoi gérer les erreurs ?***

La gestion d'erreur intervient principalement dans deux contextes.

### **Pendant le développement**

Pendant la phase de développement, la gestion des erreurs permet de tester et valider que votre application fonctionne bien. Définir une politique de gestion d'erreur stricte vous permet de tester le bon fonctionnement de votre outil et de localiser plus facilement les erreurs. Vous pouvez ainsi vérifier que toutes vos variables sont bien initialisées, que leur valeur est bien comprise entre deux extrémités, que vos fonctions répondent correctement, etc.

Détecter les erreurs vous permet de limiter leur portée en les corrigeant dans la mesure du possible. Par exemple, si la connexion à votre base de données n'est pas active à un moment donné, vous pouvez traiter cette erreur de fonctionnement en proposant un mode dégradé.

### **En production**

Gérer les erreurs en phase de production est une partie très importante car elle vous permet d'être au courant si quelque chose ne fonctionne pas comme prévu. Une erreur peut être due à une condition particulière ponctuelle (votre connexion Internet est tombée), mais peut également venir de problèmes plus importants.

Récupérer les messages d'erreur permet par exemple d'être averti quand la base de données est hors service ou que certaines fonctionnalités ne marchent plus. Plus tôt vous serez averti, plus tôt vous pourrez y remédier.

Les messages d'erreur peuvent aussi vous permettre de vous rendre compte de problèmes de sécurité. Si une erreur survient de manière isolée alors qu'elle n'était jamais apparue auparavant, cela signifie potentiellement que quelqu'un a réussi à trouver une faille dans votre script et à déclencher un comportement anormal.

## ***Que faire avec les erreurs ?***

Il y a deux façons de réceptionner les erreurs : soit les intercepter, soit les enregistrer.

### **Intercepter les erreurs**

Dans le premier cas, cela vous permet de réagir de plusieurs façons :

- afficher une explication ;

- tenter d’y remédier en offrant un service dégradé mais fonctionnel ;
- prévenir automatiquement l’administrateur du site ;
- arrêter l’exécution du script, etc.

Sans gestion d’erreur, votre application au mieux s’arrête et au pire a un comportement incohérent. Imaginez par exemple que le prix de tous vos articles soit nul à cause d’une défaillance de votre SGBD, mais qu’il soit toujours possible de remplir son panier et de commander.

### Enregistrer les erreurs

La seconde méthode consiste à stocker les erreurs dans un journal. Il est ainsi possible à tout moment d’en prendre acte et de réagir en conséquence. La lecture d’un tel journal vous permet de prendre connaissance de chutes éventuelles de votre base de données ou de bogues dans le code, voire de comportements anormaux provoqués par des personnes mal intentionnées. Cette vérification est particulièrement importante lors de la phase de développement.

## Les erreurs PHP

PHP a plusieurs systèmes d’erreurs : les erreurs PHP, les assertions et les exceptions. Le premier, le plus classique, concerne les messages envoyés par le moteur PHP quand il rencontre quelque chose d’anormal.

### *Description d’une erreur PHP*

Pour PHP, un « comportement anormal » regroupe aussi bien une erreur de syntaxe dans un script que la tentative d’ouverture d’un fichier inexistant ou même l’utilisation d’une variable non initialisée.

Il existe bien sûr plusieurs types d’erreurs PHP. Pour les différencier, PHP renseigne plusieurs informations quand il en crée une :

- un niveau d’importance,
- un message explicatif,
- le nom du script en cause,
- le numéro de la ligne en cours.

Toutes ces informations peuvent être récupérées par la fonction qui reçoit l’erreur.

Pour exemple, dans le message d'erreur visible à la figure 19-1, le niveau d'importance est spécifié avec le mot `Warning` (il s'agit d'une erreur non critique), le message explicatif nous indique qu'on a essayé d'ouvrir un fichier non existant ou non accessible (fichier.txt) et que cet essai se situe dans le script `erreur.php` à la ligne 3.

**Figure 19-1**  
*Warning*



## Les bases d'une gestion d'erreur

Vous trouverez dans ce chapitre de nombreux détails sur les différentes problématiques de la gestion des erreurs et les différentes possibilités d'action. Si vous souhaitez ne faire que quelques actions simples, voici un bref résumé des fonctions les plus utilisées.

### Réagir à une erreur

En cas d'échec la plupart des fonctions retournent `NULL` ou `FALSE`. Vous pouvez vous servir de ce comportement pour faire un test :

```
if ( !fonction_a_tester() ){
    // Instructions en cas d'échec
}
```

L'utilisation de conditions peut s'avérer lourde, aussi est-il possible de gérer automatiquement une action en cas d'erreur avec l'opérateur `or`.

```
$result = fonction_a_tester()
    or instruction_en_cas_d_echec();
```

En cas d'échec, on crée généralement une erreur avec la fonction `trigger_error()`. Celle-ci prend en paramètres un texte d'erreur et, optionnellement, un niveau d'erreur.

```
fonction_a_tester()
    or trigger_error('il y a eu un problème', E_USER_ERROR);
```

En choisissant le niveau d'erreur `E_USER_WARNING`, PHP envoie juste un avertissement, mais continue l'exécution du script. La valeur `E_USER_ERROR` correspond à une erreur critique et PHP arrête totalement l'exécution en la recevant.

**Note**

Vous pourriez utiliser la fonction `die()` en cas d'erreur. C'est une mauvaise idée, car vous ne pourrez pas l'intégrer à une gestion centralisée des erreurs plus tard. Préférez `trigger_error()` avec un niveau `E_USER_ERROR`, qui aura le même effet par défaut.

Vous pouvez aussi choisir de silencieusement ignorer les erreurs renvoyées par une fonction ou une instruction en la préfixant par le symbole arobase `@`. Nous vous recommandons toutefois d'essayer d'éviter les erreurs plutôt que de les ignorer.

```
// N'affiche rien en cas d'échec
@mysql_connect();
```

**Configurer sa gestion d'erreur**

Par défaut, PHP affiche le texte de toutes les erreurs reçues. Autant en phase de développement il s'agit d'un comportement important, autant en production on conseille de rien afficher, d'une part pour ne pas effrayer le visiteur, et d'autre part pour éviter de dévoiler des informations sur le script. Il convient donc, pour un serveur en production, de désactiver cet affichage en modifiant la directive `display_errors` dans votre fichier de configuration `php.ini`.

```
display_errors = Off
```

**Attention**

Si vous utilisez le fichier de configuration `php.ini-recommended`, l'affichage des erreurs est désactivé par défaut. Pensez donc à réactiver l'affichage pendant la phase de développement.

Avec ce type de comportement, il vous faut cependant un autre moyen de connaître les éventuels problèmes de l'application. Pour cela, on va stocker les messages d'erreur dans un fichier de *log*.

```
log_errors = On
error_log = /chemin/vers/fichier/phplog.txt
```

Vous pouvez aussi décider de gérer manuellement les erreurs pour pouvoir faire un traitement personnalisé. Des détails seront donnés plus loin dans ce chapitre.

**Configuration de développement**

En phase de développement, il convient de connaître toutes les informations que pourrait nous renvoyer PHP sur d'éventuels comportements anormaux.

```
error_reporting = E_ALL | E_STRICT
display_errors = on
log_errors = on
log_errors_max_len = 1024
track_errors = off
error_log = /var/log/php-error.log
```

## Configuration de production

En phase de production, il ne faut pas dévoiler les éventuels messages d'erreur aux visiteurs, mais il faut tout de même pouvoir être averti. On se base donc sur un système de *log*.

```
error_reporting = E_ALL
display_errors = off
log_errors = on
log_errors_max_len = 1024
track_errors = off
error_log = /var/log/php-error.log
```

## Niveaux d'erreurs et filtres

Il est évident qu'une erreur de syntaxe dans un script n'a pas la même importance qu'une variable non initialisée. Bien qu'il soit important de traiter les deux, votre applicatif ne réagira pas de la même manière.

### Les différents niveaux d'erreur

Pour faire une distinction entre erreurs, PHP attribue un niveau à chacune d'elles. Voici une liste des différents niveaux d'erreur possibles et leur signification. Pour chaque niveau, PHP fournit une constante qui représente un type d'erreur.

#### Erreurs de syntaxe (E\_PARSE)

PHP produit une erreur de type `E_PARSE` quand il a un problème pour interpréter le code PHP. Il s'agit presque essentiellement d'erreurs de syntaxe dans les fichiers (parse error).

#### Erreurs critiques (E\_ERROR)

Le type `E_ERROR` est l'erreur critique la plus courante. Elle intervient quand une fonctionnalité importante ne peut pas s'exécuter (par exemple un `require()`).

#### Avertissements (E\_WARNING)

`E_WARNING` est le message d'avertissement le plus courant. Il intervient quand une fonction a rencontré un comportement anormal, mais que PHP peut continuer son exécution, par exemple quand l'ouverture d'un fichier échoue. En général, les résultats retournés ou les comportements qui suivent risquent d'être erronés.

#### Syntaxe dépréciée (E\_STRICT)

Le niveau d'erreur `E_STRICT` a été ajouté avec PHP 5. Il s'agit de messages d'avertissements qui sont envoyés quand vous utilisez une ancienne syntaxe déconseillée, qui risque de poser des problèmes d'interopérabilité ou de compatibilité ascendante.

### Message informatif (E\_NOTICE)

Une erreur de type E\_NOTICE est de faible importance ; elle n'a qu'une valeur informative. PHP la signale, mais la suite de l'exécution ne sera probablement pas perturbée, les résultats retournés sont probablement ceux souhaités. Essayer d'utiliser en lecture une variable non initialisée provoque un message E\_NOTICE.

### Erreurs internes de PHP (E\_CORE\_XXX)

E\_CORE\_ERROR et E\_CORE\_WARNING sont des erreurs critiques et des avertissements envoyés par le cœur de PHP. Si vous rencontrez un message d'un de ces types, ce n'est probablement pas la faute de votre application, mais celle de PHP lui-même.

### Erreurs de compilation (E\_COMPILE\_XXX)

Les erreurs de type E\_COMPILE\_ERROR et E\_COMPILE\_WARNING sont retournées lors de la compilation en *byte code* de vos scripts. Si vous rencontrez un message d'un de ces types, ce n'est probablement pas la faute de votre application, mais celle de PHP lui-même.

### Erreurs utilisateur (E\_USER\_XXX)

Il vous est possible de créer vous-même explicitement un message d'erreur, comme s'il venait de PHP. Les trois niveaux d'erreur E\_USER\_ERROR, E\_USER\_WARNING et E\_USER\_NOTICE représentent les équivalents de E\_ERROR, E\_WARNING et E\_NOTICE quand ils sont créés par vous et non par PHP.

## Filtrer les erreurs utiles

PHP offre la possibilité de filtrer les erreurs en fonction de leur niveau. Pour effectuer ce filtrage, nous utiliserons soit la fonction `error_reporting()`, soit la directive de configuration du même nom qui se situe dans le fichier `php.ini`.

Chacune des constantes vues précédemment représente un niveau d'erreur qui est une puissance de 2. Il est donc possible de créer un filtre particulier en les ajoutant bit à bit. Par exemple, la valeur `E_ERROR|E_WARNING` représente les erreurs critiques et les messages d'avertissement génériques. Si vous souhaitez voir s'afficher tous les types d'erreurs, vous pouvez utiliser la constante `E_ALL` qui est la somme de tous les types d'erreurs sauf les `E_STRICT` (pour des raisons de compatibilité avec les scripts PHP 4).

En spécifiant un de ces filtres comme valeur pour la directive de configuration, on restreint de fait les types d'erreurs que PHP sera habilité à nous renvoyer.

```
; affiche toutes les erreurs
error_reporting = E_ALL
; affiche toutes les erreurs sauf les notices
error_reporting = E_ALL & ~E_NOTICE
```

Modifier la valeur de la directive dans le fichier de configuration implique que le niveau de gestion d'erreur sera le même sur tous vos scripts. Il est possible de définir un filtre particulier dans un script ou une portion de script, via la fonction `error_reporting()`.

```
<?php
```



```
unset($var) ;

error_reporting(E_ALL) ;
echo 'Le niveau d\'erreur est maximum';

echo $var ;
// Affiche un message d'erreur de type Notice
// car on essaye de lire $var alors qu'elle n'a pas été initialisée

error_reporting(E_ALL & ~E_NOTICE) ;
echo '<br>Le niveau d erreur ne prévient pas des warning';
echo '$var' ;
// N'affiche pas le message d'erreur car il est filtré
// par error_reporting
?>
```

**Figure 19-2**

*Erreur pour  
l'utilisation de  
variables indéfinies*



Généralement, on définit le niveau du filtre à `E_ALL&~E_NOTICE`, afin de ne pas être perturbé par les messages signalant la lecture de variables non initialisées. Dans l'optique d'un développement pointu, il est plutôt conseillé de mettre le niveau d'erreur à `E_ALL` (et donc de toujours initialiser vos variables). Procéder ainsi vous permettra, entre autres, de vous rendre compte des incohérences engendrées par une mauvaise orthographe de vos noms de variables. De telles erreurs donnent généralement des résultats erronés mais sont difficiles à remarquer et à corriger ; laissez PHP vous aider à les repérer.

### Forcer la compatibilité PHP 5

Si vous n'utilisez que du code PHP 5 et pas de code objet venant de la version précédente, vous pouvez même avantageusement mettre une valeur de `E_ALL|E_STRICT` afin de forcer l'utilisation des syntaxes PHP 5, qui vous garantissent une meilleure compatibilité ascendante et une meilleure qualité de programmation. Cette syntaxe est toutefois à éviter si vous utilisez du code qui a été prévu pour PHP 4. Vous auriez alors beaucoup de messages d'avertissement pour des erreurs qui n'en sont pas vraiment (il s'agit juste d'une utilisation tout à fait valide et correcte des anciennes syntaxes).

```
<?php
error_reporting(E_ALL|E_STRICT) ;
```

### Désactiver localement un message d'erreur

Il est possible de désactiver la gestion des erreurs pour une commande particulière, en la préfixant avec l'opérateur arobase @. Cet opérateur fonctionne comme si vous aviez mis un filtre à 0 le temps d'exécuter la commande.

```
<?php
error_reporting(E_ALL) ;
@fopen('/fichier/inexistant', 'r') ;
echo @$variable_inexistante ;
// Ces deux commandes ne généreront aucun message d'erreur
// à cause du @ comme préfixant la partie posant problème
?>
```

### Créer une erreur manuellement

Il existe trois types d'erreurs utilisateur : E\_USER\_ERROR, E\_USER\_WARNING et E\_USER\_NOTICE. Il est possible de provoquer de telles erreurs avec la fonction `trigger_error()`.

En lui fournissant un texte en unique paramètre, PHP crée un message d'erreur en utilisant le type par défaut E\_USER\_NOTICE. Vous pouvez spécifier un autre type d'erreur utilisateur en le fournissant en deuxième paramètre. Par la suite, l'erreur utilisée est traitée de la même façon qu'une erreur provoquée par PHP en interne. La seule différence se situe sur le niveau d'erreur utilisé.

Les figures 19-3 et 19-4 présentent le rendu que vous pouvez avoir sur différents niveaux d'erreur.

```
<?php
$prix = mt_rand(-15,10);
if ( $prix < 0 ) {
    $message = "Il y a un problème avec le prix : $prix" ;
    $niveau = E_USER_ERROR ;
    trigger_error($message, $niveau) ;
}
?>
```

Figure 19-3  
E\_USER\_ERROR

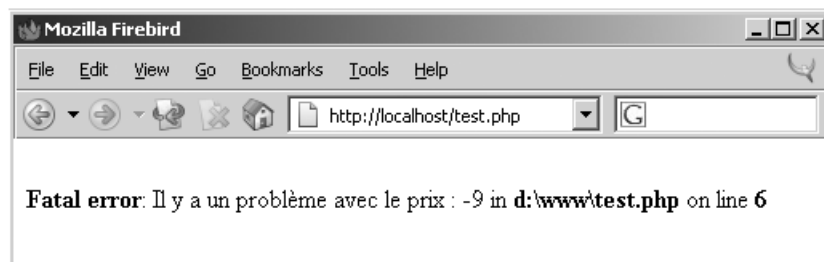
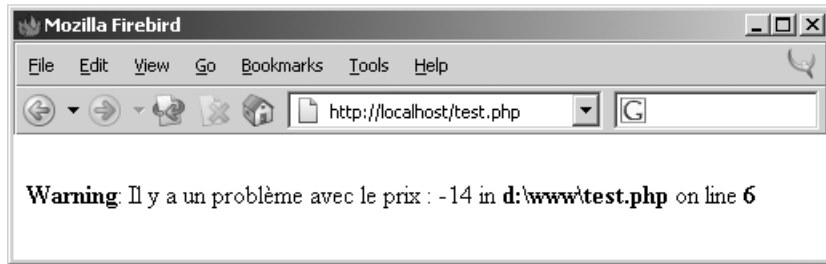


Figure 19-4  
*E\_USER\_WARNING*



## Affichage des erreurs

Le comportement par défaut de PHP est d'afficher les erreurs dans le flux de sortie (généralement dans la page web créée). Ce comportement est très agréable en période de développement car on voit rapidement toutes les erreurs lors des tests.

Inversement, en production, il est peu souhaitable que les utilisateurs voient les messages d'erreur qui peuvent survenir. Effectivement, une telle divulgation d'erreur peut effrayer votre client (s'il voit un message du type *can't connect to /tmp/socks/...*), donner des indications sur votre configuration permettant de faciliter le piratage, ou divulguer des informations confidentielles.

En désactivant la directive de configuration `display_errors` (qui est activée par défaut), vous pouvez désactiver l'affichage des erreurs sur la sortie standard. Les erreurs sont toujours produites et traitées, mais pas affichées par défaut. Nous vous recommandons de désactiver cette directive sur vos serveurs de production et d'utiliser la journalisation à la place.

```
display_errors = Off
```

## Aide au développeur

Depuis les dernières versions 4.3 de PHP, il est possible de créer automatiquement un lien HTML vers la documentation quand un message d'erreur est affiché. Ce lien vous permet de sauter directement vers le chapitre concerné du manuel afin de vous documenter.

Ce comportement est activé par la directive de configuration `html_errors`. Vous aurez à le désactiver si les *tags* HTML des erreurs arrivent à des endroits peu pertinents et cassent toute l'interprétation de la page, rendant complexe la visualisation de l'erreur au lieu de l'améliorer.

```
html_errors = On
```

Par défaut, les liens pointent vers la documentation du site officiel. Il est toutefois possible, lors d'un développement en local ou pour un fonctionnement plus réactif, de le faire pointer sur une documentation locale ou sur un site personnel. PHP fournit seul le nom

de la page de documentation ; ce nom est automatiquement préfixé par le contenu de la directive de configuration `docref_root` et le contenu de `docref_ext` est ajouté en suffixe.

```
; attention à bien inclure le / final
docref_root = "http://example.com/manual/"
; attention à bien inclure le . initial
docref_ext = ".html"
```

### Affichage personnalisé

L'apparence des messages d'erreur affichés par PHP peut être personnalisée. Cette fonctionnalité n'est pas uniquement décorative ; elle peut servir si votre format de sortie n'est pas du HTML, si vous avez besoin de faire fortement ressortir les erreurs (par exemple par une autre couleur) ou si au contraire vous souhaitez les mettre en commentaire HTML pour qu'elles ne se voient que dans le code source HTML.

Le contenu de la directive de configuration `error_prepend_string` est ajouté avant l'erreur affichée, et le contenu de `error_append_string` est affiché après.

```
error_prepend_string = "<font color='red'>"
error_append_string = "</font>"
```

### Journalisation des erreurs (log)

Il est possible d'enregistrer les erreurs dans un fichier de journal (*log*). Ce comportement est le seul moyen prévu par défaut pour enregistrer les erreurs de manière durable dans un historique. Il doit être activé et consulté sur des serveurs en production.

Il peut également être utile d'activer la journalisation aussi sur les serveurs de développement. Les messages d'erreur affichés peuvent passer inaperçus s'ils sont en fin de page et que le développeur porte son attention sur le haut. Ils peuvent même simplement ne pas être affichés si l'erreur se trouve dans un commentaire HTML ou imbriquée dans un *tag* HTML spécifique. Dans ce cas, la journalisation est l'unique comportement qui vous permette de vous assurer qu'il n'y a pas d'erreur pendant l'interprétation. Nous vous conseillons d'activer l'affichage et la journalisation des erreurs pendant les développements.

#### Note

Pensez bien à vérifier régulièrement le contenu de votre fichier de journalisation. Il est aussi utile de mettre en place ce qu'on appelle une rotation des journaux : de temps en temps, recopier ailleurs le fichier de journal sous forme compressée et vider le fichier original, de façon à limiter la place occupée par ces informations.

### Journalisation automatique

En activant la directive de configuration `log_errors` dans le `php.ini`, PHP envoie toutes les erreurs vers le fichier de journalisation du serveur web ou dans son flux d'erreur s'il

s'agit d'une exécution en ligne de commande. Il est très fortement recommandé de s'assurer que cette directive est toujours activée.

Si vous souhaitez utiliser un fichier de journal personnel plutôt que celui de votre serveur web, vous pouvez en spécifier l'adresse dans la directive de configuration `error_log`. Les erreurs sont alors ajoutées à ce fichier quand elles apparaissent. Il vous appartient de vous assurer que votre serveur web a les droits d'écriture sur ce fichier.

```
log_errors = On
error_log = /data/log/phplog.txt
```

### Recoupement des erreurs dans le journal

Il est possible de demander à PHP d'ignorer les erreurs répétées venant d'une même ligne d'un même script. Cela évite de masquer les autres erreurs en surchargeant trop le fichier. Vous pouvez demander à PHP de faire ce tri en activant la directive de configuration `ignore_repeated_errors`.

Vous pouvez aussi demander à PHP d'opérer une agrégation encore plus importante. Si vous activez la directive de configuration `ignore_repeated_source`, PHP ignore les messages d'erreur répétés, même s'ils viennent de fichiers ou de lignes différentes. Il est recommandé de ne pas utiliser cette option, car vous risquez de passer à côté d'une erreur si plusieurs apparaissent à peu d'intervalle, et de n'en corriger qu'une.

### Utiliser le journal système

Vous pouvez aussi, si vous le souhaitez, utiliser la journalisation globale de votre système d'exploitation plutôt qu'un fichier utilisateur. Pour obtenir ce comportement, il vous suffit de spécifier la valeur spéciale `syslog` comme nom de fichier à la directive `error_log`. La gestion centralisée de votre système est alors utilisée ; selon vos logiciels, vous pourrez y filtrer spécifiquement les messages PHP pour les isoler ou les catégoriser.

### Utilisation manuelle du journal

PHP permet de communiquer directement avec le journal sans passer par d'éventuels filtres ou gestionnaires d'erreurs.

La fonction `error_log()` permet d'envoyer le message d'erreur ou une note d'avertissement en paramètre vers le système de journalisation automatique de PHP.

```
error_log('message pour le système de journalisation PHP');
```

### Être prévenu des erreurs par e-mail

En fournissant à la fonction `error_log()` la valeur 1 comme deuxième paramètre, PHP envoie le message d'erreur par e-mail à l'adresse indiquée en troisième. Ce comportement a l'avantage de vous faire parvenir l'information par un système de *push*. Il est très utile dans le cas d'intrusion ou de défaut de sécurité car il externalise les messages

d'avertissement (un pirate pourrait effacer le fichier de journal, mais il ne peut pas toucher à vos messages, qui sont sur un autre serveur).

### En-têtes supplémentaires

Vous pouvez spécifier des en-têtes e-mail supplémentaires via un cinquième argument. Le sujet du message ne peut pas être déterminé par le développeur, même en essayant de le définir via les en-têtes.

```
<?php
$sql = mysql_connect('localhost') ;
if ( !$sql ) {
    // Le serveur SQL est hors service
    // On avertit vite l'administrateur par e-mail
    // car le SGBD est un service critique dont dépend le site
    $message = 'Impossible de se connecter au serveur MySQL ;
    $email = 'administrateur@example.com' ;
    error_log($message, 1, $email) ;
}
?>
```

### Définir le fichier à utiliser

En fournissant la valeur 3 comme deuxième paramètre et une adresse de fichier comme troisième, PHP ajoute le message au fichier de journal utilisateur spécifié.

## Utiliser le journal système

### Initialisation

Avant toute utilisation des fonctions ci-dessous, vous devez faire un appel à la fonction `define_syslog_variables()`, qui sert à initialiser certaines constantes relatives à la journalisation de votre système.

```
<?php
define_syslog_variables();
```

### Ouverture du journal

Les journaux système permettent généralement de filtrer les messages en fonction de paramètres comme un identifiant de programme ou une catégorie de service. Il nous faut d'abord déclarer au journal système nos paramètres avec la fonction `openlog()`.

Le premier argument de cette fonction est une chaîne de caractères. Elle est utilisée comme identifiant pour le journal système. Il s'agit normalement du nom de votre application, de son abréviation ou du nom du service qu'elle procure.

Le deuxième argument est une valeur composée par les constantes suivantes :

- `LOG_CONS` : s'il y a un problème dans l'envoi du message vers le journal, affiche l'erreur sur la console système.

- `LOG_NDELAY` : ouvre la connexion vers le journal système immédiatement et inconditionnellement.
- `LOG_ODELAY` : valeur par défaut, n'ouvre la connexion que lorsque le premier message est envoyé, afin d'économiser les ressources si aucun message n'est transmis.
- `LOG_PERROR` : envoie aussi le message via le flux d'erreur standard.
- `LOG_PID` : le système ajoute automatiquement l'identifiant du processus au message envoyé.

Il est possible de composer ces valeurs pour cumuler les effets. Ainsi, `LOG_ODELAY|LOG_CONS` n'ouvre la connexion que lorsqu'un message d'erreur est transmis et, si la transmission est impossible, alors le message est envoyé vers la console système.

Le troisième et dernier paramètre de la fonction `openlog()` définit le type de service à l'origine du message. Ce dernier est à choisir parmi une liste de constantes. Celles que vous pourriez être amené à utiliser sont les suivantes :

- `LOG_AUTHPRIV` : authentification privée ;
- `LOG_LOCAL0` à `LOG_LOCAL7` : espaces utilisateur ;
- `LOG_USER` : messages utilisateur génériques.

La valeur à utiliser par défaut est `LOG_USER`. Si toutefois vous voulez séparer certaines erreurs des autres en faisant ces catégories, vous pouvez utiliser une des constantes de `LOG_LOCAL0` à `LOG_LOCAL7`. Le service `LOG_USER` est le seul disponible sur les plates-formes Microsoft Windows.

```
define_syslog_variables();
openlog("monApplication", LOG_ODELAY|LOG_CONS, LOG_USER);
```

### Écrire un message dans le journal

Une fois la connexion avec le journal système initialisée, vous pouvez y envoyer des messages avec la fonction `syslog()`. Elle prend en paramètre une priorité d'erreur et une chaîne de caractères comme message. La priorité peut être une valeur définie par une des constantes suivantes :

- `LOG_EMERG` : le système est inutilisable ;
- `LOG_ALERT` : une décision doit être prise immédiatement ;
- `LOG_CRIT` : erreur critique ;
- `LOG_ERR` : message d'erreur générique ;
- `LOG_WARNING` : message d'avertissement fort ;
- `LOG_NOTICE` : message d'avertissement faible ;

- LOG\_INFO : message à caractère informatif ;
- LOG\_DEBUG : information de débogage.

```
<?php
define_syslog_variables();
openlog("monApplication", LOG_ODELAY|LOG_CONS, LOG_USER);
if ( $prix <= 0 ) {
    $message = "un prix de la boutique s'est retrouvé à 0" ;
    $niveau = LOG_ERR ;
    syslog($niveau, $message) ;
}
?>
```

Vous trouverez un exemple d'utilisation du journal système de Microsoft Windows à la figure 19-5.

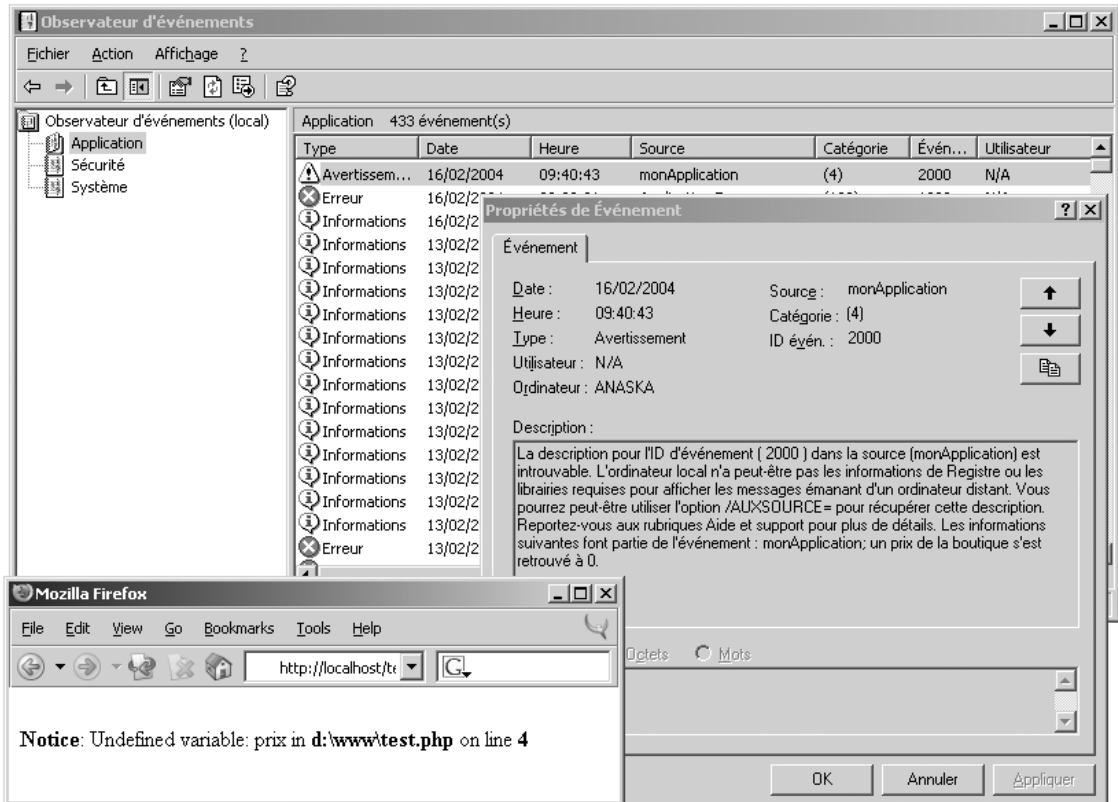


Figure 19-5

Utilisation du journal système



### Clôture de la connexion

Quand vous avez fini d'interagir avec le journal système, il est préférable de fermer la connexion. Cette fermeture est optionnelle (PHP le fait tout seul en fin de script) mais permet de libérer un descripteur de fichier avant la fin du script si votre système en fait un usage important. La clôture se fait via la fonction `closelog()`.

```
<?php
define_syslog_variables();
openlog("monApplication", LOG_ODELAY|LOG_CONS, LOG_USER);
if ( $prix <= 0 ) {
    $message = "un prix de la boutique s'est retrouvé à 0" ;
    $niveau = LOG_ERR ;
    syslog($niveau, $message) ;
}
closelog();
?>
```

### Personnaliser le gestionnaire d'erreurs

Jusqu'à présent, nous avons laissé PHP gérer les erreurs via ses gestionnaires internes. Si vous avez des besoins spécifiques, il est possible de définir un gestionnaire d'erreurs personnalisé qui traitera les erreurs.

Pour créer votre gestionnaire d'erreurs, il faut commencer par créer une fonction qui prend quatre paramètres. Le premier est un niveau d'erreur PHP. Le deuxième est une chaîne de caractères qui contiendra le message d'erreur reçu. Les deux derniers sont le nom du script et la ligne où l'erreur est intervenue, ces paramètres sont automatiquement remplis par PHP.

Le contenu de la fonction est entièrement à votre charge. Vous aurez à afficher les messages sur la sortie, les enregistrer dans des journaux ou tout autre action. PHP ne s'occupera plus de rien, pas même du filtrage dû à `error_reporting` ou à l'opérateur `@`. PHP n'arrêtera pas non plus l'exécution du script après une erreur critique du type de `E_USER_ERROR`.

Il vous revient de faire un appel à la fonction `exit()` avec le code de retour approprié. Si vous souhaitez honorer le filtrage automatique, il vous faudra le faire vous-même avec la valeur de `error_reporting()`.

```
function gestion_erreur( $niveau, $message, $fichier, $ligne) {
    if ( $niveau & error_reporting() ) {
        echo '<b>', htmlentities($message), '</b>' ;
        echo " ligne $ligne du fichier $fichier." ;
        if ($niveau == E_USER_ERROR) exit(255) ;
    }
}
```

Une fois la fonction de gestion définie, il reste à la passer à PHP via la fonction `set_error_handler()`.

```
set_error_handler('gestion_erreur');
```

**Note**

Comme pour toutes les fonctions de rappel, il est possible d'utiliser une méthode en fournissant un tableau à la place d'une chaîne de caractères. Le premier élément doit être l'objet à utiliser (ou le nom de la classe s'il s'agit d'une méthode statique) et le second doit être le nom de la méthode.

Il est possible de restaurer le gestionnaire précédent (qu'il soit défini par l'utilisateur ou par PHP) et de revenir en arrière avec `restore_error_handler()`. Il est ainsi possible d'imbriquer des gestionnaires d'erreurs et d'en activer un spécifique pour une petite partie du code.

```
restore_error_handler();
```

### Erreurs fatales internes à PHP

PHP considère que quand une erreur surgit parmi `E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR` et `E_COMPILE_WARNING`, l'état de l'interpréteur est potentiellement instable ou non sûr. PHP arrête alors directement l'exécution sans faire appel au gestionnaire défini par vous-même. Il est impossible de recueillir côté utilisateur une telle erreur ; il est donc important de définir une gestion d'erreur par défaut correcte, même si vous comptez utiliser un gestionnaire d'erreur personnalisé.

## Les assertions

### *Description d'une assertion*

Une assertion est une affirmation fonctionnelle et logique définie par le développeur. Il est ainsi possible d'affirmer que « le prix est un nombre positif non nul ». Lors de l'exécution, PHP évalue toutes les assertions et retourne une erreur si l'une d'elles n'est pas respectée.

Les assertions sont donc des structures de contrôles. Elles permettent de vérifier que des données sont cohérentes avec ce qu'on devrait avoir, et de signaler un comportement anormal si ce n'est pas le cas.

### *Utilisation d'une assertion*

Les assertions ne doivent être utilisées que pour valider une affirmation qui ne devrait pas pouvoir être fausse. Il s'agit d'un mécanisme de sécurité qui ne doit pas faire partie intégrante de la logique applicative. Logiquement, retirer toutes les assertions ne doit pas changer le comportement normal de votre application ni dégrader sa sécurité.

Si une fonction interne reçoit un prix envoyé par vos soins, vous pouvez mettre une assertion qui vérifie que ce prix est positif et non nul. En revanche, s'il s'agit d'une fonction publique et que le prix reçu provienne d'un utilisateur sans étape intermédiaire de vérification, l'assertion n'est pas adaptée. Dans ce genre de cas, vous devriez faire une condition classique avec `if()`.

Pour faire une assertion, il suffit d'encapsuler votre affirmation dans une chaîne de caractères et de la fournir en paramètre à la fonction `assert()`. La chaîne de caractères est interprétée comme pour la fonction `eval()` et la valeur retournée est analysée comme un booléen. Si ce booléen est faux, alors PHP renvoie une erreur interne.

```
assert( '$prix > 0' );
```

Attention : si vous utilisez des guillemets, PHP interprète vos variables comme d'ordinaire et ne fournit que le résultat à `assert()`. L'affirmation `assert("!empty($prix)")` ne peut par exemple pas fonctionner car `$prix` est interprété pendant la construction de la chaîne de caractères. La fonction `assert()` évalue alors par exemple `!empty(233)`.

## Désactivation des assertions

Comme nous l'avons dit plus haut, les assertions ne devraient pas être nécessaires au bon fonctionnement de votre application. Elles sont là uniquement pour vérifier que tout se passe comme prévu. Il est d'usage de les utiliser pendant le développement et les premiers temps de la mise en production, au cas où un comportement imprévu surviendrait.

Après quelques temps en production sans erreurs, il est possible de désactiver les assertions. En les désactivant, vous demandez à `assert()` de ne plus évaluer ce qui lui est passé en paramètre et de l'ignorer silencieusement. L'avantage d'une assertion par rapport à une condition avec un `if()` classique prend alors tout son sens : il est possible de multiplier les contrôles sans penser aux performances. Quand vous aurez besoin de performances, il vous suffira de désactiver les assertions, le code ne souffrira alors d'aucun ralentissement dû au nombre d'assertions ou à la complexité d'évaluation des affirmations.

Ce comportement fait que nous vous conseillons d'user et d'abuser des assertions, même aux endroits que vous pensez peu propices aux erreurs. Le but est de traquer des comportements anormaux non prévus. N'étant pas prévues par définition, ces erreurs peuvent survenir n'importe où. Plus vous avez d'assertions, plus vous avez de chances de remarquer un comportement anormal. La seule limite que vous devez vous fixer est celle de la lisibilité du code.

## Configuration des assertions

Pour activer ou désactiver les assertions, vous avez deux possibilités :

- opérer ponctuellement via la fonction `assert_options()`,
- travailler sur la configuration générale avec le `php.ini`.

Dans le premier cas, il faut faire appel à la fonction `assert_options()` en lui passant en paramètre la constante `ASSERT_ACTIVE`. Le second paramètre est un booléen. S'il est vrai, les assertions sont activées, sinon elles sont désactivées. Si ce dernier paramètre est absent, `assert_options()` vous retourne la valeur en cours.

```
<?php
if (assert_options(ASSERT_ACTIVE)) {
    echo 'les assertions sont déjà activées';
} else {
    assert_options(ASSERT_ACTIVE, 1) ;
    echo 'les assertions viennent d'être activées';
}
?>
```

La seconde possibilité consiste à modifier le fichier de configuration, le `php.ini`. Cela vous permet de définir le comportement par défaut de votre application. À l'initialisation, PHP utilise les assertions si la directive de configuration `assert.active` est activée, ce qui est le cas par défaut, et les ignore sinon.

```
assert.active On
```

#### Note

Les deux méthodes sont utilisables simultanément. La directive de configuration définit un comportement que vous pouvez modifier par la suite pour un script ou une portion de script avec la fonction PHP.

## Génération d'une erreur interne

Par défaut, quand PHP rencontre une assertion fautive, le moteur retourne un message d'erreur de type `E_WARNING`.

Vous pouvez toutefois décider que vos assertions ont un poids plus important et signaler des erreurs critiques en cas d'échec. Il est alors possible de demander à PHP d'arrêter l'exécution via la directive `ASSERT_BAIL` de `assert_options()`, ou via la directive de configuration `assert.bail` du `php.ini`. Une valeur vraie demande à PHP de stopper l'exécution sur une fautive affirmation, une valeur fautive laisse le comportement par défaut.

```
assert_options(ASSERT_BAIL, 0) ;
```

## Avoir un message explicatif lors de l'erreur

Par défaut, le message d'erreur retourné par PHP lors d'une assertion fautive contient uniquement le code PHP évalué et sa position dans le script. Ce message est le seul que PHP soit capable de faire ; il lui est impossible de savoir pourquoi vous faites cette affirmation et de détailler le problème.

Il serait pourtant utile d'avoir des messages d'erreur plus explicatifs dans les journaux afin de voir d'un coup d'œil quel peut être le problème. Une astuce pour mettre un message d'erreur personnalisé est d'ajouter un commentaire dans le code PHP à évaluer.

Ce commentaire se retrouve dans le message d'erreur et peut servir de description.

```
assert('$prix > 0 ; // le prix reçu est négatif ou nul')
```

Vous devez faire cependant attention à la taille maximale des erreurs dans les journaux (cette taille est habituellement de 1 024 octets). Si vous la dépassez, votre message sera tronqué. Si le code PHP à évaluer est important, vous risquez d'avoir peu de place pour votre message.

### Erreurs lors de l'assertion

Il est possible que la vérification d'une assertion provoque elle-même un message d'erreur. C'est particulièrement vrai si vous faites appel à une fonction utilisateur dans votre affirmation. Étant donné que les assertions ne participent pas à votre application et peuvent théoriquement être ignorées, recevoir ces erreurs d'évaluation peut être considéré comme inutile.

Vous pouvez désactiver les messages d'erreur pendant l'évaluation des assertions à l'aide de la directive `ASSERT_QUIET_EVAL` de `assert_options()` ou la directive de configuration `assert.quiet_eval`. Une valeur vraie désactive les messages d'erreur, la valeur par défaut laisse PHP renvoyer ces erreurs.

```
assert_options(ASSERT_QUIET_EVAL, 0) ;
```

### Personnalisation de la gestion

Les assertions sont faites pour retourner une erreur standard PHP, et pouvoir ainsi être récupérées via les gestionnaires d'erreurs globaux, qu'ils soient internes à PHP ou définis par l'utilisateur. Il est toutefois possible de traiter les assertions séparément, via un gestionnaire spécifique.

Le gestionnaire d'assertions doit accepter trois paramètres : l'adresse du script, le numéro de ligne où l'affirmation a été faite, et le code PHP évalué.

```
function gestion_assert($script, $ligne, $message) {  
    echo 'Une assertion fausse a été faite '  
        . " dans le script $script à la ligne $ligne : $message" ;  
}
```

On enregistre le gestionnaire d'assertions dans PHP à l'aide de la directive `ASSERT_CALLBACK` de la fonction `assert_options()`, en passant le nom de la fonction comme valeur. Toutes les assertions évaluées comme fausses seront envoyées au gestionnaire utilisateur. La valeur `NULL` désactive le gestionnaire personnalisé.

```
assert_options(ASSERT_CALLBACK, 'gestion_assert') ;
```

Il est aussi possible d'initialiser ce paramètre dans la configuration du `php.ini` avec la directive `assert.callback`. Pensez alors bien à toujours inclure la définition de la fonction de gestion avant toute utilisation d'assertion.

## Les exceptions

Le mécanisme d'exception a été ajouté à PHP 5. Cette fonctionnalité existe dans la plupart des langages objets. Il s'agit d'envoyer et de recevoir plus simplement des messages d'erreur évolués.

### Description d'une exception

Une exception est un type d'erreur qui permet des interactions évoluées. Côté PHP, les exceptions sont des objets dérivant de la classe `Exception`. La classe de base permet de stocker dans l'exception un message d'erreur, un code d'erreur numérique, et bien sûr l'adresse du script ainsi que le numéro de la ligne où l'exception a été lancée.

Quatre méthodes permettent d'accéder à ces informations en lecture : `getMessage()`, `getCode()`, `getFile()` et `getLine()`.

Une exception est un objet PHP classique. Il suffit donc d'utiliser le mot-clé `new` pour en créer une. Le constructeur prend en premier paramètre le message d'erreur. Le second paramètre, le code d'erreur, est optionnel. Le fichier et la ligne seront remplis automatiquement par PHP quand l'exception sera exécutée (on parle plutôt de « lancer » (en traduction de *throw*) une exception ; c'est ce vocabulaire que nous utiliserons par la suite.

```
$mon_exception = new Exception('Division par 0') ;  
echo $mon_exception->getMessage() ;
```

Voici un aperçu de l'implémentation de la classe d'exception par PHP telle qu'on pourrait se l'imaginer :

```
class Exception {  
    private $message ;  
    private $code ;  
    private $line ; // Automatiquement rempli par PHP en interne  
    private $file ; // Automatiquement rempli par PHP en interne  
    public function __construct($message, $code = NULL) ;  
        $this->message = $message ;  
        $this->code = $code ;  
    }  
    public function getMessage() {  
        return $this->message ;  
    }  
    public function getCode() {  
        return $this->code ;  
    }  
    public function getFile() {  
        return $this->file ;  
    }  
    public function getLine() {  
        return $this->line ;  
    }  
}
```

## Personnalisation des exceptions

Comme nous venons de le voir, les exceptions sont gérées sous forme de classes. Il est donc possible d'étendre leurs possibilités en y ajoutant des méthodes ou des propriétés pour gérer des cas spécifiques. Si vous avez un type d'erreur bien particulier, vous pouvez personnaliser la classe `Exception` pour en créer une plus spécialisée avec des paramètres supplémentaires.

```
class FichierInexistant extends Exception {
    private $fichierInexistant ;
    private $include_path ;

    public function __construct($message, $fichierInexistant) {
        $this->fichierInexistant = $fichierInexistant ;
        $this->include_path = ini_get('include_path') ;
        parent::__construct($message) ;
    }

    public function getFichierInexistant() {
        return $this->fichierInexistant ;
    }

    public function getIncludePath() {
        return $this->include_path ;
    }
}
```

## Lancement d'une exception

Une fois l'exception créée, il faut l'exécuter, la lancer. Tant qu'il n'est pas lancé, l'objet d'exception n'est rien de plus qu'un objet PHP classique, il n'a aucun comportement particulier. Lancer l'exception permet de dire à PHP qu'une erreur est survenue et de lui fournir en paramètre l'objet d'exception. On lance une exception avec le mot-clé `throw` :

```
if ( ! file_get_contents($fichier, TRUE) ) {
    $message = "Le fichier $fichier n'a pas pu être trouvé" ;
    $except = new FichierInexistant($message, $fichier) ;
    throw $except ;
}
```

## Réception d'une exception

Quand l'exception est lancée, PHP ajoute à l'objet le nom du fichier et le numéro de la ligne où le lancement a eu lieu. PHP arrête alors l'exécution en cours et saute au premier gestionnaire d'exceptions qui sait gérer l'erreur lancée pour continuer l'exécution à partir de là. On déclare les gestionnaires d'exceptions avec les blocs `try` et `catch`. Dans un fonctionnement normal, PHP interprète le contenu du bloc `try` et ignore complètement le bloc `catch`, en continuant à exécuter ce qui suit. Si une exception survient dans un bloc `try`,

PHP arrête l'exécution du bloc, interprète le contenu du catch puis continue avec ce qui suit (si vous ne l'avez pas arrêté entre temps).

```
// Cas générique
try{
    // Code à exécuter tout le temps
} catch(Exception $e){
    // Code de gestion d'erreur
}
```

Sur notre exemple d'exception pour fichier inexistant, on pourrait avoir le code suivant :

```
try {
    $reussite = file_get_contents($fichier, TRUE) ;
    if (!$reussite) {
        $message = "Le fichier $fichier n'a pas pu être trouvé" ;
        $except = new fichierInexistant($message, $fichier) ;
        throw $except ;    // On saute directement au bloc catch
    }
    echo "Ce message n'est pas lu si le fichier n'a pas été trouvé" ;
}
catch(Exception $e) {
    // Cette partie n'est interprétée
    // que si une exception a été lancée
    echo 'Une exception a été reçue : ' ;
    echo $e->getMessage() ;
}
// Ce qui suit est toujours exécuté, qu'il y ait eu exception ou non
```

## Filtrage des exceptions reçues

Dans la déclaration du bloc catch, on peut remarquer qu'on utilise une des syntaxes de la programmation orientée objet. En effet, on vient de spécifier un nom de classe devant l'argument de catch. Le mot-clé catch agit un peu comme une fonction. L'erreur est fournie en paramètre et réceptionnée dans une variable.

Le nom de la classe précédant la variable permet de définir le type d'exception géré. Spécifier ce type permet d'implémenter des blocs catch dédiés à la gestion de certaines exceptions, et d'autres plus génériques pouvant traiter toutes les exceptions. Il est ainsi possible de déclarer plusieurs blocs catch pour chaque bloc try ; PHP utilisera le premier qui correspond.

Nous avons vu au chapitre 12 concernant les objets que le contrôle de type vérifie que l'objet est de la classe spécifiée ou d'une sous-classe. Si vous spécifiez Exception, cela revient à accepter de traiter toutes les exceptions car elles dérivent toutes de cette classe.

Dans l'exemple suivant, on individualise trois types d'exceptions : la classe Exception par défaut, la classe PbOuvertureFichier qui en dérive et traite spécifiquement des erreurs de fichiers, et enfin FichierNonPrésent qui est un cas encore plus spécifique de problème sur l'ouverture des fichiers. Dans la clause try, on lance une erreur générique d'ouverture de



fichier (PbOuvertureFichier). Cette erreur passe alors à travers les différents `catch` jusqu'à en trouver un qui corresponde. Le premier ne correspond pas car il accepte uniquement le cas spécifique des fichiers non présents (classe `FichierNonPresent` et dérivées). Le second accepte toutes les classes dérivant de la classe `Exception` ; comme c'est le cas de notre erreur `PbOuvertureFichier`, son contenu sera donc analysé et le troisième `catch` ne sera jamais utilisé, bien que valide et plus précis.

```
<?php
class PbOuvertureFichier extends Exception { }
class FichierNonPresent extends PbOuvertureFichier { }
try {
    throw new PbOuvertureFichier("Exception fille");
    echo "ceci n'est pas affiché car une exception a eu lieu" ;
}
catch(FichierNonPresent $e) {
    echo "ceci n'est pas affiché car l'exception " ,
        "n'appartient pas à la classe FichierNonPresent " ;
}
catch(Exception $e) {
    echo "ceci est exécuté car l'exception " ,
        "est un objet dérivé de la classe Exception" ;
}
catch(PbOuvertureFichier $e) {
    echo "ceci n'est jamais appelé car l'exception valide d'abord " ,
        "le masque Exception, positionné en premier" ;
}
?>
```

## Propagation des exceptions

### Interception par un bloc parent

Si les filtres des blocs `catch` ne permettent pas de traiter une exception, PHP relance l'exception automatiquement pour vérifier si elle ne peut pas être traitée par un couple de `try/catch` parent.

```
<?php
class ExceptionFille extends Exception { }
try {
    /* Actions quelconques */
    try {
        /* Actions quelconques */
        throw new Exception('...') ;
        echo 'ceci ne sera jamais vu car une exception a eu lieu' ;
    }
    catch(ExceptionFille $e) {
        echo "ceci ne sera jamais vu car l'exception n'est pas" ,
            " de type ExceptionFille" ;
    }
}
```

```
// PHP propage donc l'exception au couple try/catch parent
echo 'ceci ne sera pas vu car on passe directement au catch' ;
try { /* ... */ }
catch(Exception $e) {
    echo "ceci ne sera pas vu car il s'agit d'un bloc frère," ,
        " pas d'un bloc parent" ;
}
}
catch(Exception $e) {
    echo 'ceci sera exécuté car on traite ici toutes les exceptions';
}
?>
```

### Renvoi d'une exception au bloc parent

Si vous remarquez, après avoir intercepté une exception, que finalement vous ne savez pas la gérer, ou qu'elle n'est pas entièrement gérée, rien ne vous empêche de la rejeter avec un `throw`. Elle pourra ainsi être récupérée par une méthode de plus haut niveau. On peut par exemple utiliser cette fonctionnalité pour ne gérer qu'un seul type d'erreur SQL : on vérifie alors si le code d'erreur contenu dans l'exception est bien celui qu'on sait gérer. Si ce n'est pas le cas, alors on renvoie l'exception en espérant qu'un autre `catch()` plus loin saura gérer cette erreur.

```
<?php
try {
    try {
        throw new Exception("message") ;
    }
    catch(Exception $e) {
        echo $e->getMessage() ;
        // Il est important que le catch parent gère aussi l'erreur
        // On la renvoie
        throw $e ;
        echo "ceci n'est pas affiché" ;
    }
    echo "ceci n'est pas affiché" ;
}
catch(Exception $e) {
    echo " - Une exception est survenue - " ;
    echo " - Ceci est affiché car l'exception a été renvoyée - " ;
}
?>
```

### Gestionnaire d'exceptions par défaut

Il peut être utile d'avoir un gestionnaire d'exceptions par défaut. Toutes les exceptions non récupérées passeront par ce gestionnaire et vous permettront d'implémenter une logique d'erreur spécifique pour toutes les exceptions non récupérées, par exemple une journalisation ou un e-mail à l'administrateur.

Vous pouvez avoir un gestionnaire d'exceptions par défaut simplement en mettant tout votre script principal dans un bloc `try` et en implémentant un `catch` qui filtre le type `Exception` (c'est-à-dire toutes les exceptions possibles). Si vous n'avez pas de gestionnaire par défaut et si une exception ne trouve pas de bloc `catch` adéquat, PHP retourne une erreur interne classique à partir du message de l'exception.

Il est aussi possible de définir un gestionnaire d'exceptions par défaut en donnant un nom de fonction à `set_exception_handler()`. La fonction qui implémentera le gestionnaire d'exceptions devra accepter un unique paramètre qui sera l'exception reçue. Cette procédure d'interception est similaire à celle utilisée avec les erreurs PHP, mis à part que c'est une exception qui est passée en paramètre.

```
function gestion_par_defaut($exception) {
    echo $exception->getMessage();
}
set_exception_handler('gestion_par_defaut');
/* Reste du script */
```

## Utilisation des exceptions

L'arrivée des exceptions dans PHP ouvre de grandes possibilités pour la gestion des erreurs dans les codes orientés objet. Une méthode qui perçoit un comportement anormal peut alors lancer une exception spécifique sans se soucier de savoir qui la récupère et comment. Si une classe de plus haut niveau a prévu que cette erreur pouvait survenir, elle aura implémenté un `catch` qui gèrera cette erreur et aucune autre.

Les exceptions ont plusieurs avantages principaux par rapport aux erreurs PHP classiques : elles s'imbriquent facilement, il est possible d'avoir un contexte ou des informations plus importantes qu'un simple message, on peut définir des types d'erreurs pour les catégoriser, on peut filtrer les erreurs qu'on sait gérer ou pas, et pour finir, l'exécution reprend à la fin du bloc qui pose problème en sautant tout ce qui en dépend, alors qu'un gestionnaire d'erreurs classique rend la main juste après l'erreur.

Si vous utilisez la programmation orientée objet, vous avez tout intérêt à vous baser le plus possible sur les exceptions ; elles vous fourniront une souplesse inégalée pour la gestion de vos erreurs. Leur utilisation n'est toutefois pas exclusive et vous pouvez utiliser et les erreurs PHP et les exceptions, même si cela rend la gestion plus complexe et moins claire.

### Utiliser uniquement les exceptions

Si vous comptez utiliser les exceptions, le problème que vous rencontrerez est que les fonctionnalités non objet de PHP utilisent des erreurs classiques et non des exceptions.

Une astuce permet malgré tout de remédier au problème et de récupérer toutes les erreurs PHP sous forme d'exceptions. Il suffit d'utiliser un gestionnaire d'erreurs classique qui a pour comportement de créer et lancer une exception. À chaque fois qu'une erreur intervient, elle lance une exception, avec un comportement tout à fait classique.

```
function Erreur2Exception($no, $mes, $file, $line) {  
    $e = new Exception($mes, $no) ;  
    throw $e ;  
}  
set_error_handler('Erreur2Exception') ;
```

Ce code est toutefois inutilisable tel quel. Le premier problème vient des exceptions non récupérées : PHP les convertit en effet en erreurs classiques. Si de votre côté vous convertissez les erreurs en exceptions, vous risquez d'avoir un comportement en boucle. Une solution est de détecter les erreurs qui viennent d'exceptions non récupérées en lisant le texte, et de ne pas lancer d'exception dans ce cas-là. Une autre solution est de toujours bien penser à définir un gestionnaire d'exceptions par défaut.

Le deuxième problème vient des niveaux d'erreur. Si votre script renvoie un simple message d'avertissement, il est probablement une mauvaise idée de le convertir en exception, qui correspond à une erreur importante, et qui arrêtera l'exécution du script (ou au moins du bloc de code concerné). Il peut aussi être une bonne idée de filtrer les erreurs suivant leur niveau et de ne lancer une exception que pour les erreurs critiques.

## Politiques de gestion d'erreur

Nous avons décrit dans ce chapitre les différentes fonctionnalités de PHP pour traiter les erreurs. Il nous reste à donner quelques pistes sur les comportements à adopter.

### *Le développement*

Pendant le développement, vous avez intérêt à gérer toutes les erreurs de la façon la plus simple et uniforme possible afin de ne pas en oublier. Les recommandations habituelles sont généralement d'afficher ces erreurs sur la sortie pour que le développeur ne puisse passer à côté et de les ajouter simultanément à un fichier de journal dédié à PHP, au cas où une erreur ne serait pas visible ou remarquée dans la sortie. En utilisant le journal du serveur web, vous risquez de ne pas voir les erreurs réelles de votre serveur si elles sont noyées au milieu de tests PHP.

### *Être averti lors d'un problème*

Les messages d'avertissement ou les erreurs sont révélateurs d'un comportement anormal et vous vous devez d'y prêter attention. Généralement, la journalisation suffit à garder trace des erreurs et à comprendre ce qui s'est passé. Pourtant, les journaux ne suffisent pas à être informé et à savoir en temps réel quand une erreur survient.

Une procédure habituelle est d'envoyer un e-mail à l'administrateur quand une erreur survient. Afin de ne pas submerger l'administrateur sous des e-mails répétés, il est recommandé d'inscrire dans un fichier l'heure du dernier e-mail envoyé, pour ne pas en envoyer plus d'un par heure par exemple.

Si vous en avez la possibilité, l'envoi d'un texto ou l'activation d'un *bipeur* peut aussi être une solution si vous n'avez pas de personnel d'astreinte toujours présent pour lire les e-mails.

### ***Toujours agir lors d'une erreur***

Il est fréquent de voir des journaux d'erreurs laissés à l'abandon car personne ne les lit, ou des journaux lus mais qui continuent à avoir les mêmes erreurs de jour en jour. S'il est important de toujours vérifier ce qui a pu se passer, il est aussi important d'agir pour qu'une erreur ne se reproduise plus, même si vous pensez qu'elle ne porte pas à conséquence.

En effet, en la laissant dans les cas que vous connaissez, vous risquez de ne pas remarquer des erreurs différentes (par leur nature ou par leur contexte) qui surviendraient au milieu de la centaine d'erreurs régulières. Imaginez aussi que votre même message d'erreur puisse être compris et accepté aujourd'hui, mais représenter un problème tout à fait différent demain. Pour être sûr de remarquer le problème de demain, il faut corriger celui d'aujourd'hui.

### ***Externaliser les alertes de sécurité***

Si une erreur vient de votre application et si elle correspond à un problème de sécurité potentiel, vous vous devez de la traiter de manière différente des autres. En effet, si quelqu'un arrive à exploiter une faille sur votre applicatif, la première chose qu'il essaiera de faire, c'est effacer les traces de son passage dans les journaux d'erreurs.

Pour mettre en échec son comportement, vous devez avoir un système de journalisation qui n'accepte que l'écriture, pas l'effacement. Votre journal système permet probablement ce type de fonctionnalité : votre application peut y écrire un message sans contraintes mais nécessitera les droits de super-utilisateur pour pouvoir en effacer. Certains administrateurs envoient aussi les messages très importants vers l'imprimante en cas de problèmes de sécurité. Sans aller jusque-là, un autre système simple est d'envoyer un e-mail à une boîte qui est gérée sur un autre serveur. Vous risquez de surcharger votre boîte mail avec des messages d'erreur en cas d'attaque, mais il vaut mieux une boîte trop pleine qu'ignorer ce qui s'est passé.

### ***Gardez des traces sur le contexte***

Quand une erreur nouvelle ou importante intervient, il peut être utile de sauvegarder son contexte : comment elle est arrivée, quelle fonction a appelé le code litigieux, avec quels paramètres, etc. En effet, reproduire un problème n'est pas toujours évident et parfois il est dur de retrouver le cas qui a engendré l'erreur.

La fonction `debug_backtrace()` peut vous être utile. Elle permet de récupérer toute la pile des appels de fonctions avec, à chaque fois, où l'appel a été fait (nom du script et numéro de ligne), le nom de la fonction appelée et les paramètres fournis.

La fonction `get_defined_vars()` permet, elle, de récupérer les valeurs de toutes les variables actuellement définies, qu'elles soient des variables utilisateur, des variables d'environnement ou des variables venant d'une des superglobales.

Une possibilité est d'écrire un nouveau fichier dans un répertoire dédié à chaque fois qu'une nouvelle erreur se produit. Dans ce fichier, on stocke le message d'erreur et ses paramètres, mais aussi l'intégralité de ce que retournent les fonctions `debug_backtrace()` et `get_defined_vars()`. Il vous sera alors par la suite possible de chercher à partir de ces données ce qui a pu causer le problème.



## XML : concepts et SimpleXML

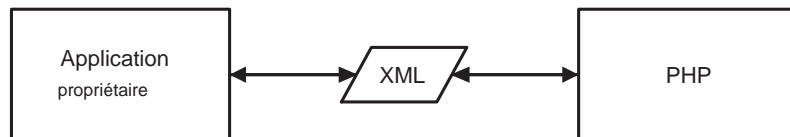
---

Comment vulgariser le XML ? XML est le sigle pour *eXtensible Markup Language*. On pourrait dire que c'est un langage universel permettant d'interfacer des systèmes ne parlant pas la même langue. Il s'agit d'un format standardisé, hiérarchique et manipulable par presque tous les langages de programmation et la majorité des logiciels. Dans ce premier chapitre dédié à XML, nous allons nous intéresser aux concepts et aux possibilités gravitant autour de cette technologie. Bien qu'il existe de nombreux outils permettant de manipuler du XML, nous allons d'abord nous intéresser à celui qui est apparu avec PHP 5 : SimpleXML. Il s'agit d'un module prévu pour gérer simplement des manipulations XML de complexité limitée. Son avantage est de nécessiter très peu de code et de connaissances.

### De l'utilité du XML

Grâce à XML, on peut faire communiquer PHP avec n'importe quelle application ou langage ayant un moteur XML.

**Figure 20-1**  
*XML, un format d'échange de données*



La technologie XML est le standard d'échange principal entre systèmes hétérogènes.



## Gains apportés par XML

Voici quelques exemples de gains que pourrait vous apporter l'utilisation d'XML dans vos applications :

- Simplicité d'utilisation : le traitement XML est relativement simple et les API sont standardisées et similaires dans tous les langages, ce qui facilite l'apprentissage.
- Pérennité de votre modèle : XML est compris par la majorité des applications et des logiciels. Ainsi, quand vous changerez un composant externe de votre application PHP, il y a de fortes chances qu'il puisse aussi comprendre (et donc utiliser) vos flux XML. Notons également que les méthodes de manipulation sont standardisées ; le coût d'une évolution sera donc moindre, car il y aura moins à refaire et à réapprendre.
- Compatibilité et interopérabilité : le XML est un standard du W3C. Utilisant ce format, vous êtes assuré de sa relecture par le destinataire.

## Exemples d'utilisation

Pour entrer dans un domaine plus concret, on peut citer quelques exemples d'applications mettant en œuvre PHP et XML :

- Utilisation des services fournis par des tiers (services Web) : vous pouvez utiliser SOAP ou XML-RPC pour vous connecter par exemple à un site de vente en ligne et exploiter ses outils à distance.
- Échanges basés sur un format standard et connu de tous : vous êtes assuré que vos documents pourront être utilisés par vos clients.
- Agrégation de flux RSS : par exemple, vous pouvez récupérer les dépêches AFP.
- Élaboration de documents à partir d'une seule source vers différents formats : PDF, HTML, XHTML, WML (téléphones Wap), images SVG, etc.

## Présentation et prérequis

Le XML est en vogue depuis quelques années. Ce n'est cependant pas une technologie nouvelle. La réflexion sur ce format a commencé dès 1996 au W3C (*World Wide Web Consortium*, organisme qui édicte les normes du Web comme le HTML et le SVG) et la recommandation a été finalisée deux ans après, en février 1998.

Le XML descend du SGML (*Standard Generalized Markup Language*), défini par le standard ISO-8879 en 1986. Il en reprend la structure générale et les principes afin de le simplifier et de permettre des implémentations rapides et efficaces.

Comme le SGML, il s'agit d'un méta-format ayant pour objectif de structurer les données de manière standardisée, tout en évitant les pièges courants que sont notamment la non-extensibilité ou la dépendance face à une plate-forme. Tous les formats XML utilisent la même structure : des descriptions par balises.

L'application la plus courante du SGML sur le Web est le HTML, qui permet de faire les pages web. Le XML a non seulement réadapté ce format (en créant le XHTML) mais aussi permis l'émergence de nombreux autres formats du domaine web.

## Structure du XML

Décrire complètement la norme XML est hors du cadre de ce livre, mais nous allons vous en proposer un aperçu, ainsi qu'une comparaison avec le HTML. Cela vous donnera les bases pour que vous puissiez manipuler et comprendre les informations de ce chapitre. Pour plus de détails ou d'explications, vous êtes invité à consulter la documentation officielle à l'adresse <http://www.w3.org/XML/>.

### Description par balises

Si vous utilisez HTML, vous êtes probablement familier avec la description par balises. Il s'agit d'entourer le texte par une balise ouvrante (`<balise>`) et une balise fermante (`</balise>`). La balise sert à décrire le contenu et permet d'y accéder facilement.

#### Attention

Contrairement au HTML, la balise de fin est toujours obligatoire en XML.

Dans l'exemple suivant, le contenu « PHP » est décrit par une balise `<langage>` :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<langage>PHP</langage>
```

Une balise peut contenir un attribut ou plusieurs. Contrairement au HTML, tous les attributs doivent avoir une valeur :

```
<description langue="fr">Langage de script</description>
```

Les commentaires s'écrivent comme en HTML :

```
<description langue="fr">Langage de script</description>
<!-- L'attribut de description est langue et sa valeur est fr -->
```

Les balises vides peuvent bénéficier d'une syntaxe réduite : `<balise></balise>` est équivalent à `<balise />`. La balise `<br>` représentant un saut de ligne en HTML s'écrit `<br />` en XHTML.

### Hiérarchie

Il est possible d'imbriquer les balises pour en faire une hiérarchie. L'imbrication est sans limite mais il doit y avoir un unique élément à la racine :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<siteweb>
  <miroir pays="us">http://www.php.net</miroir>
  <miroir pays="fr">http://fr.php.net</miroir>
  <miroir pays="fr">http://fr2.php.net</miroir>
</siteweb>
```

### Imbrication de balises

Attention à bien respecter l'imbrication des balises. Si certains navigateurs web acceptent le chevauchement des balises en HTML, écrire `<b><i>texte</b></i>` est interdit en XML.

Du fait de sa hiérarchie stricte, une des représentations habituelles d'un fichier XML est sous forme d'arbre (voir figure 20-2).

**Figure 20-2**

*Arborescence d'un fichier XML*



### Jeux de caractères

Un fichier XML commence par une déclaration de la version XML et du jeu de caractères utilisés. Il est possible de ne pas préciser le codage des caractères (qui sera alors par défaut un codage ASCII 7 bits) et même de sauter la déclaration entièrement. Comme vous utiliserez probablement les accents français, il est recommandé de déclarer le jeu de caractères correspondant : ISO-8859-1.

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

### Codages caractères

Un codage caractère est ce qui permet à l'ordinateur de représenter un caractère sous forme binaire. Historiquement, les caractères sont stockés sur 7 bits et ne comprennent que l'alphabet américain. Le nom de ce codage est US-ASCII.

Par la suite, les différents pays ont décidé de représentations sur 8 bits, de façon à doubler le nombre de caractères disponibles et à pouvoir utiliser un alphabet différent. Le jeu de caractères ouest-européen, utilisé en France, est nommé ISO-8859-1. Il a les mêmes 128 premiers caractères que l'US-ASCII, mais contient en plus sur les 128 suivants les caractères accentués latins utilisés en France ou en Italie. Un deuxième jeu est apparu récemment, gérant le caractère Euro et quelques autres : le jeu ISO-8859-15.

Le problème de ces multiples jeux de caractères est qu'un même texte peut être interprété de multiples façons selon le jeu utilisé pour la lecture. Pour résoudre ce problème, il existe un codage global contenant tous les caractères de tous les alphabets utilisés. Deux jeux de caractères utilisent ce codage : UTF-8 et UTF-16 (le premier tend à s'imposer sur l'autre du fait de sa compatibilité avec l'US-ASCII). Ils tardent à s'imposer, car ils nécessitent de réécrire les applications et de convertir les fichiers existants.

Dans votre environnement, vous aurez probablement à manipuler trois formats : US-ASCII car c'est le format par défaut du XML, ISO-8859-1 car c'est le format par défaut du HTTP (donc des transferts via le Web) et UTF-8, qui est le jeu international le plus utilisé.

## Entités et caractères spéciaux

Si vous souhaitez utiliser dans vos fichiers XML des caractères significatifs (<, >, " et &) sans qu'ils soient interprétés, il faudra utiliser ce qu'on appelle une entité.

Une entité est un élément commençant par & et finissant par un point-virgule. Les entités correspondant à <, >, " et & sont respectivement &lt;;, &gt;;, &quote; et &amp;.

Il n'y a que quatre entités par défaut en XML (les autres entités existant en HTML ne sont pas définies et sont donc invalides). Si votre jeu de caractères déclaré le permet, vous pouvez directement écrire les caractères tels que les lettres accentuées, sinon il vous faudra utiliser une entité numérique : &x0000; où 0000 représente la valeur hexadécimale du caractère dans le jeu Unicode.

## Espace de noms

Pour éviter des conflits entre plusieurs formats XML qui utiliseraient les mêmes noms de balise, on attribue un identifiant unique à chaque langage XML : l'espace de noms.

Cet espace de noms est identifié par un URI (*Unique Resource Identifier*). En général un URI est aussi une URL (*Uniform Resource Locator*) qui utilise le protocole HTTP, par exemple <http://www.w3.org/1999/xhtml> pour le langage XHTML. Toutefois, un URI peut être n'importe quelle chaîne de caractères unique et même quand il s'agit d'une URL, celle-ci n'est pas forcément utilisable dans un navigateur web.

C'est cet espace de noms qui permet de savoir à quel langage XML appartient une balise. Quand le langage est déterminé et officialisé, on attribue donc l'espace de noms à chaque balise. Cette association est faite avec un attribut spécial nommé `xmlns` qui contient l'URI d'un espace de noms. La présence de cet attribut `xmlns` dans une balise déclare que cette balise et ses descendantes appartiennent à l'espace de noms en question.

Il est aussi possible de définir un préfixe arbitraire qu'on va attribuer à un espace de noms. Cette affectation se fait avec un attribut `xmlns:XXX`, où XXX est le préfixe qu'on souhaite créer et où l'URI de l'espace de noms correspondant est la valeur de l'attribut. Ensuite, on peut ajouter ce préfixe à n'importe quelle balise pour l'affecter à l'espace de noms correspondant. La balise `<html>` deviendrait ainsi `<XXX:html>`.

### Note

La notion d'espace de noms et le fonctionnement des espaces de noms dépassent largement le cadre de ce livre. Nous ne vous donnons que les bases vous permettant de savoir de quoi nous parlons et de décoder les exemples XML. Nous vous recommandons de consulter des ouvrages dédiés à XML pour plus de détails sur ce sujet.

## Exemple de fichier XML

Étant donné le contexte, le meilleur exemple est un fichier XML représentant la documentation d'une fonction PHP. Ce type de fichier est utilisé par les équipes de documentation et de traduction du groupe PHP. Il décrit la fonction `utf8_decode()` dont nous aurons l'occasion de parler plus tard.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- last change to 'utf8-decode' in en/ tree in rev 1.2 -->
<refentry id="function.utf8-decode">
  <refnamediv>
    <refname>utf8_decode</refname>
    <refpurpose>
      Convertit une chaîne UTF-8 en ISO-8859.
    </refpurpose>
  </refnamediv>
  <refsect1>
    <title>Description</title>
    <methodsynopsis>
      <type>string</type>
      <methodname>utf8_decode</methodname>
      <methodparam>
        <type>string</type>
        <parameter>data</parameter>
      </methodparam>
    </methodsynopsis>
    <para>
      <function>utf8_decode</function>
      décode la chaîne
      <parameter>data</parameter>, en supposant
      qu'elle est au format
      <literal>UTF-8</literal>, et la convertit
      au format <literal>ISO-8859-1</literal>.
    </para>
    <para>
      Voir aussi <function>utf8_encode</function>
      pour plus de détails sur le codage
      <literal>UTF-8</literal>.
    </para>
  </refsect1>
</refentry>
```

Ce fichier XML permet, entre autres, d'afficher la documentation en ligne du site `fr.php.net` illustrée à la figure 20-3.

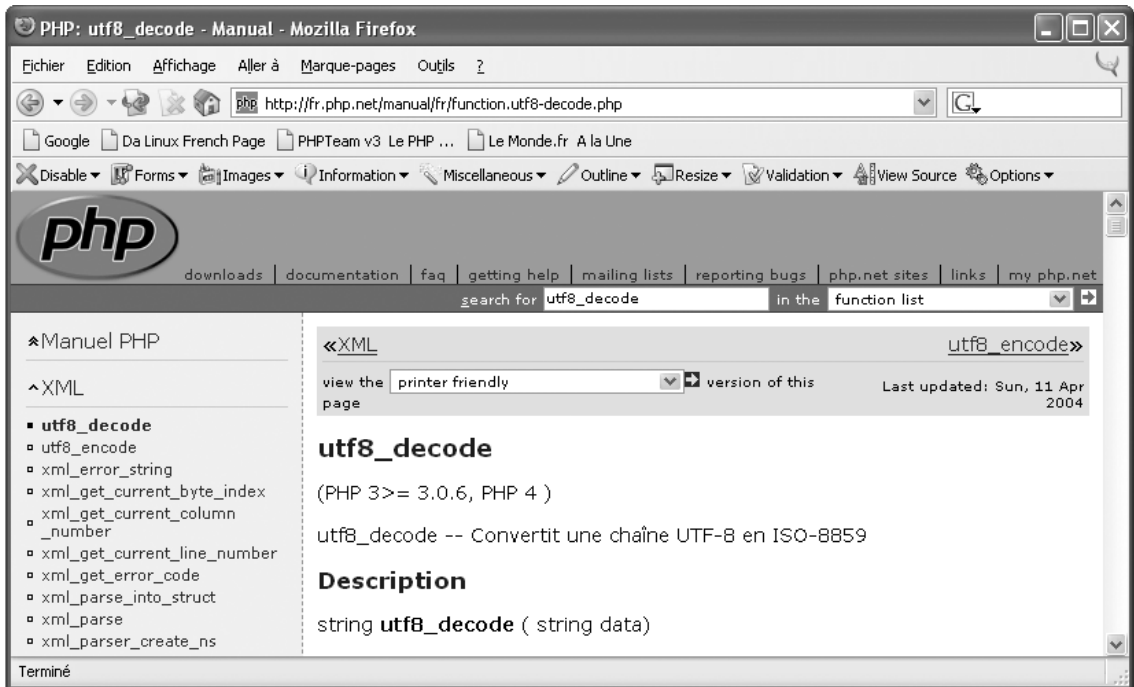


Figure 20-3

Documentation en ligne de `utf8_decode()`

## Principaux formats

Quand on parle de XML, on parle de langage balisé. De nombreux formats basés sur XML ont des balises standardisées permettant ainsi d'échanger des données.

### Syndication avec des fichiers RSS

RSS (*Really Simple Syndication*) est un format conçu par Netscape afin de présenter les titres des articles en publication de manière standardisée. Chacun peut relire ce fichier XML pour connaître les dernières nouvelles et les intégrer à une interface personnelle. Il est alors possible d'être prévenu d'une mise à jour d'un site ou d'un contenu sans avoir à le visiter manuellement régulièrement. On parle de syndication de données.

On trouve plusieurs versions du format RSS ; voici un échantillon de RSS 0.91 :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="0.91">
  <channel>
    <title>PHPTeam.net</title>
    <link>http://www.phpteam.net</link>
    <description>Le PHP professionnel</description>
```

```
<language>fr-fr</language>
<item>
  <title>PHP 5</title>
  <link>http://www.php.net/</link>
  <description>PHP 5 est sorti</description>
</item>
<item>
  <title>PHP 4.3.2</title>
  <link>http://www.php.net</link>
  <description>
    La version 4.3.2 est en ligne. Les utilisateurs d'une
    version &lt; a 4.3.1 sont encourages a mettre a jour.
  </description>
</item>
</channel>
</rss>
```

Comme on le voit, il s'agit d'une description des articles présents sur le site `phpteam.net`. Ce fichier est mis à la disposition de qui le souhaite et permet aux sites distants d'afficher les derniers.

Chaque article est représenté par un couple de balises `<item></item>`. À l'intérieur de celles-ci, on trouve des informations sur l'article : son titre dans la balise `<title>`, sa description dans la balise `<description>` et un lien vers l'article complet dans la balise `<link>`.

### Services Web avec SOAP et XML-RPC

Les services Web représentent une norme de communication entre applications qui se mettent à disposition des services. Il s'agit d'une façon de communiquer entre machines.

Le moteur de recherche web Google et le catalogue de l'éditeur Amazon sont deux exemples de services Web accessibles via XML. En mettant à disposition leurs ressources, ils permettent au client de créer sa propre boutique vendant des produits Amazon ou son propre moteur de recherche utilisant la technologie de Google.

### Images au format SVG

Le SVG (*Scalable Vector Graphics*) est un format vectoriel d'image basé lui aussi sur XML. Une image vectorielle est basée sur des formes simples qui se redimensionnent sans dégradation. On peut alors intégrer une image SVG à une page web sans se préoccuper de la taille d'affichage chez le visiteur.

Puisqu'il s'agit d'un format basé sur la grammaire XML, il est ainsi possible via PHP de modifier ou de créer dynamiquement une image vectorielle SVG. On obtient alors des images d'une qualité parfaite et qui prennent très peu de temps à créer ou manipuler (ce n'est que du texte).

## Fichiers XSL

Le format XSL (*eXtensible Stylesheet Language*) est un format de présentation. Il permet de contrôler comment doit être affiché un fichier XML. Il est composé de deux modules distincts. Le premier module est XSLT, qui permet de modifier les données ou de les déplacer. Le second module est XSL-FO, qui définit des éléments comme les marges, les polices, les positionnements dans la page, etc.

Armés de ces deux outils, les processeurs XSL peuvent transformer un fichier XML en un format lisible par un utilisateur. L'utilisation de XSLT via PHP sera décrite au chapitre suivant.

## Fichiers RDF

Le RDF (*Resource Description Framework*) est une méthode de modélisation des données standardisées. On la retrouve par exemple dans une des versions de RSS. Ce format est fait pour permettre l'utilisation de moteurs informatiques pour traiter les documents et les organiser. PHP peut intervenir à ce niveau, interpréter le RDF et le reformuler dans une forme compréhensible par l'utilisateur (un graphique hiérarchique par exemple).

# Gérer le XML à la main

## Création d'un nouveau fichier

Pour créer un contenu XML simple, la méthode naturelle est de faire comme pour du HTML : utiliser les fonctions `echo` ou `print()`. Il n'y a que deux choses que vous deviez garder à l'esprit : utiliser les entités pour les caractères `<`, `>`, `&` et `"`, et penser à convertir vos chaînes de caractères si vous utilisez un codage autre que US-ASCII ou ISO-8859-1.

## Protéger les caractères spéciaux

Pour les entités, la fonction PHP vous permettant de les protéger est `htmlspecialchars()`. Malgré son nom qui la définit comme une fonction destinée au HTML, elle convient très bien au XML car les entités de base sont les mêmes.

```
htmlspecialchars (chaîne [,quote_style [,charset]])
```

Elle prend en argument la chaîne à transformer. Suivent deux arguments optionnels. Le premier sert à activer ou désactiver la conversion des guillemets (l'entité pour le caractère `"` n'est indispensable que dans un attribut lui-même entre guillemets). La valeur par défaut (`ENT_COMPAT`) remplace les guillemets, la valeur `ENT_NOQUOTES` permet de les laisser tels quels.

Le second paramètre optionnel permet de spécifier un jeu de caractères différent de ISO-8859-1 (voir l'encadré sur les codages caractères, plus haut dans le chapitre).

```
<?php
// Adresse de la page
$lien = 'http://mondomaine.com/page.php?varA=1&varB=2' ;
```



```
// Titre de la page
$titre = 'catalogue de la société "AB&C" ;
// Création des entités
$lien = htmlspecialchars($lien) ;
$contenu = htmlspecialchars($titre, ENT_NOQUOTES) ;
$titre = htmlspecialchars($titre) ;
// Affichage du lien
echo "<a href=\"\$lien\" title=\"\$titre\"> $contenu </a>";
?>
```

Ce script affichera dans le code source de la page résultante. On remarquera que les & ont été transformés en &amp; :

```
<a
  href="http://mondomaine.com/page.php?varA=1&varB=2"
  title="catalogue de la société &quot;AB&C&quot;"
>
  catalogue de la société "AB&C"
</a>
```

## Conversion entre jeux de caractères

### Utiliser le module iconv

Pour la conversion des jeux de caractères, vous devrez vous reporter vers le module PHP nommé `iconv`. Il comprend entre autres une fonction du même nom permettant de convertir une chaîne d'un jeu de caractères à un autre.

La fonction `iconv()` prend trois paramètres : le jeu de caractères local, le jeu destination et la chaîne à convertir. Pour utiliser ce module, la compilation doit avoir été faite avec le paramètre `--with-iconv` sous Linux.

```
<?php
echo iconv("ISO-8859-1","UTF-8","texte à mettre en UTF-8");
?>
```

### Utiliser les fonctions natives

Si vous n'avez pas compilé le module `iconv`, le module `xml` propose deux fonctions permettant les transformations les plus utilisées :

- de ISO-8859-1 vers UTF-8 avec `utf8_encode()` ;
- de UTF-8 vers ISO-8859-1 avec `utf8_decode()`.

Si vous comptez utiliser des caractères internationaux non européens, le standard est l'UTF-8, qui comprend tout le jeu Unicode et donc pourra être utilisé quelle que soit votre localité. Il est peu probable que vous ayez à manipuler un autre jeu que ces deux-là. Les fonctions `utf8_decode()` et `utf8_encode()` prennent comme unique paramètre la chaîne à convertir.

```
<?php
$texte_iso88591 = 'le texte à convertir' ;
```

```
$texte_utf8 = utf8_encode($text_iso88591) ;  
$texte_iso88591 = utf8_decode($texte_utf8) ;  
?>
```

**Attention**

Les chaînes UTF-8 nécessitent plusieurs octets pour stocker certains caractères. Actuellement, si vous n'utilisez pas le module `mbstring`, la plupart des fonctions de chaînes fonctionnent en comptant les octets et non les caractères. Ainsi, `strlen()` et `substr()` vous donneront des résultats potentiellement erronés si vous leur donnez des chaînes UTF-8 en entrée. Vous trouverez plus d'informations sur ces sujets dans le chapitre 5, dédié aux traitements de chaînes de caractères.

## Relecture et manipulations

Relire ou manipuler un fichier XML à la main avec les fonctions usuelles est bien plus complexe que le créer. C'est possible via des expressions régulières ou des fonctions bas niveau, mais difficile et propice aux erreurs. Pour relire ou manipuler vos fichiers XML une fois créés, nous vous conseillons fortement d'utiliser soit SimpleXML, soit les modules que nous présenterons au prochain chapitre concernant la gestion XML avancé.

## Écrire du XML avec XMLWriter

L'extension XMLWriter est basée sur l'API de la bibliothèque libxml XMLWriter. Cette extension permet de créer des documents XML. L'utilisation de cette classe est un bon compromis entre la difficulté de tout gérer à la main et la lourdeur du modèle DOM. Initialement disponible dans PECL, cette extension est intégrée à PHP depuis la version 5.1.2.

Cette extension peut être utilisée dans un style orienté objet ou un style procédural. Le code est relativement verbeux et n'a d'intérêt par rapport à une écriture dite à la main que parce qu'on s'assure que le code final est automatiquement bien formé (sans erreur de syntaxe).

**Figure 20-4**

*Présentation UML  
de la classe  
XMLWriter*

XMLWriter
<code>+openMemory()</code>
<code>+startDocument()</code>
<code>+startElement()</code>
<code>+writeElement()</code>
<code>+endElement()</code>
<code>+flush()</code>
<code>+outputMemory()</code>
<code>+startAttributeNs ()</code>
<code>+startCDATA()</code>
<code>+endCDATA()</code>
<code>+startComment ()</code>
<code>+endComment ()</code>
<code>+writeDTD ()</code>

## Prise en main rapide

Afin de permettre aux plus pressés de commencer rapidement à manipuler XMLWriter, vous pouvez consulter l'exemple suivant qui montre comment créer un fichier XML et lui ajouter des nœuds.

```
<?php
// On instancie la classe XMLWriter
$xml = new XMLWriter();
$xml->openMemory();

// Indiquons que nous souhaitons que le fichier soit indenté
$xml->setIndent(True);

// On indique le type du document XML
$xml->startDocument('1.0', 'ISO-8859-1');

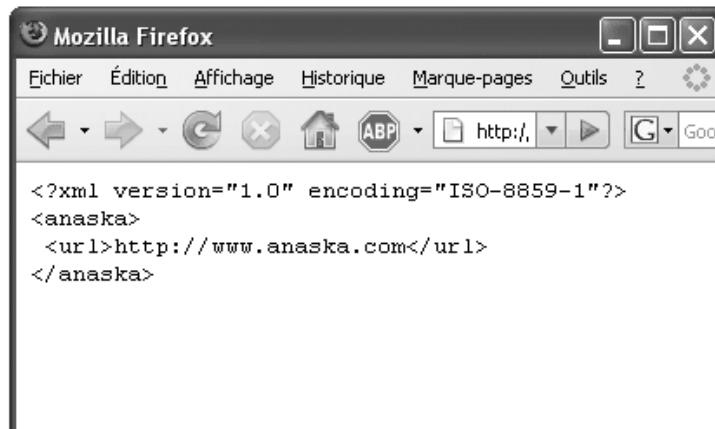
// On ajoute le noeud : <anaska>
$xml->startElement ('anaska');

// Tag : <url>http://www.anaska.com</url>
$xml->writeElement('url' , 'http://www.anaska.com');

// On ferme le noeud
$xml->endElement();

// On affiche le résultat
echo '<pre>';
echo htmlentities($xml->flush());
?>
```

**Figure 20-5**  
*Création XML  
minimaliste*



## Fonctionnalités avancées

### Créer un fichier XML

Si vous souhaitez créer un fichier XML, il existe deux façons de procéder. La première consiste à créer le flux XML et le mettre dans une variable. Pour cela, on utilisera la méthode `openMemory()`. La seconde consiste à mettre le résultat dans un fichier. Pour cela, on utilisera la méthode `openURI()` en lui fournissant en paramètre le nom du fichier XML que l'on souhaite créer.

Dans l'exemple suivant nous allons créer le fichier `test.xml`

```
<?php
$xml = new XMLWriter();
$xml->openURI('test.xml');

$xml->startElement ('anaska');
$xml->writeElement('url' , 'http://www.anaska.com');
$xml->endElement();

$xml->flush();
?>
```

### Insérer des nœuds

Pour insérer des nœuds à votre arbre XML, vous pourrez utiliser deux méthodes. La première, `startElement()` permet d'ouvrir votre nœud ; vous le refermez avec `endElement()`.

Une seconde solution consiste à utiliser la méthode `writeElement()` qui vous permet d'écrire un nœud en une seule passe.

```
class XMLWriter {
    bool writeElement ( string nom, string contenu )
}
```

### Créer un fichier RSS

Afin d'aller plus loin dans l'utilisation des fonctionnalités de `XMLWriter`, nous allons voir comment créer un fichier RSS.

```
<?php
// Tableau contenant des actualités
$actus[] = array(
    'title' => 'Appel a conferencier',
    'url' => 'http://www.afup.org/128',
    'description' => 'L AFUP a le plaisir d annoncer le forum PHP',
    'date' => '2007-10-10'
);

// On continue à remplir le tableau
$actus[] = ...;
```

```
$xml = new XMLWriter();
$xml->openUri('afup.rss');
$xml->startDocument('1.0', 'ISO-8859-1');

$xml->startElement('rss');
$xml->writeAttribute('version', '1.0');
$xml->writeAttribute('xmlns:dc', 'http://purl.org/dc/elements/1');
$xml->startElement('channel');
$xml->writeElement('title', 'RSS par XmlWriter');
$xml->writeElement('link', 'http://www.afup.org');
$xml->writeElement('Description', 'RSS par XmlWriter');

foreach($actus as $v) {
    $xml->startElement('item');
    $xml->writeElement('title', $v['title']);
    $xml->writeElement('link', $v['url']);
    $xml->startElement('description');
    $xml->writeCdata($v['description']);

    $xml->endElement();
    $xml->endElement();
}

$xml->endElement();
$xml->endElement();
$xml->flush();
?>
```

## Utilisation de SimpleXML

Le module `SimpleXML` a été introduit avec l'arrivée de PHP 5. Dans les versions précédentes de PHP, il n'y avait aucun outil permettant de manipuler simplement des fichiers XML. `SimpleXML` propose une interface pour ce type d'applications.

Son utilisation est adaptée pour relire ou modifier facilement des fichiers XML simples. Elle devient en revanche peu pertinente à partir du moment où vos fichiers sont complexes.

Ce module est compilé par défaut avec PHP 5. Vous pouvez donc vous baser dessus sans risquer d'être incompatible avec certaines installations.

### Attention

Dans tout le module `SimpleXML`, les données reçues et envoyées sont codées en UTF-8. Si vous utilisez des caractères spéciaux ou accentués, alors vous devrez passer par les fonctions `utf8_encode()` et `utf8_decode()` vues en début de chapitre.

## Import et export d'un document

### Import du code XML

L'ouverture d'un fichier XML se fait avec la fonction `simplexml_load_file()`. Elle prend en paramètre un chemin de fichier et renvoie un objet de la classe `simpleXMLElement`. Cet objet représente l'élément racine de votre document.

```
■ $racine = simplexml_load_file('monfichier.xml');
```

Il est aussi possible d'initialiser l'objet à partir d'une chaîne de caractères représentant vos données XML plutôt qu'un fichier. Vous pouvez dans ce cas utiliser la fonction `simplexml_load_string()`.

```
■ <?php
$xml = "<document><titre>PHP5 avance</titre></document>";
$racine = simplexml_load_string($xml);
?>
```

#### Note

S'il est spécifié dans le fichier XML, le fichier de grammaire (DTD) est chargé et interprété. Les entités seront donc résolues.

Il est aussi possible de s'interfacer avec le module `DOM` (*Document Object Model*), dont nous détaillerons l'utilisation au chapitre suivant. Pour récupérer un document DOM dans SimpleXML, il faut fournir l'objet document DOM en paramètre à la fonction `simplexml_import_dom()`.

```
■ <?php
$domDocument = new domDocument ;
$domDocument->load('monfichier.xml') ;
$racine = simplexml_import_dom($domDocument) ;
?>
```

#### Attention

Le fichier XML que vous importez doit être codé en UTF-8 ou avoir une déclaration correcte du jeu de caractères utilisé. Dans le cas contraire, SimpleXML affichera un message d'erreur.

### Export et sauvegarde du code XML

À tout moment, vous pouvez afficher ou exporter une partie du document XML (un sous-arbre) grâce à la méthode `asXml()`. Elle renvoie le contenu XML directement dans une chaîne de caractères. Si vous employez ces méthodes sur l'objet représentant l'élément racine, c'est tout le document XML qui sera renvoyé ; si vous l'employez sur un nœud du document, seul le sous-arbre concerné sera transformé.

```
■ <?php
$racine = simplexml_load_file('fichier.xml') ;
echo $racine->asXml() ;
?>
```

## Sauvegarder vos données XML dans un fichier

Vous pouvez envoyer le contenu XML directement vers un fichier en spécifiant l'URI (*Uniform Resource Identifier*) ou le chemin à utiliser en argument à la méthode `asXml()`.

```
<?php
$racine = simplexml_load_file('fichier.xml') ;
$racine->asXml('copie.xml') ;
?>
```

## Manipulation des éléments

Maintenant que notre document est ouvert, il est temps de regarder comment le lire. Pour des fichiers XML simples, vous n'aurez probablement qu'à lire quelques éléments.

Pour nos exemples, on utilisera le code de la page XHTML suivante :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html lang="fr">
  <head><title>PHP 5</title></head>
  <body>
    <h1>PHP 5</h1>
    <p>La version 5 de <acronym>PHP</acronym> vient de sortir</p>
    <p>La dernière version était la 4.3.x</p>
  </body>
</html>
```

### Accéder à un nœud par son nom

#### Un seul nœud

La manière la plus simple d'accéder à un nœud est d'utiliser son nom. Pour utiliser un élément `<body>`, il suffit de lire l'attribut du même nom dans l'objet parent. Cela donne, pour la page XHTML décrite précédemment :

```
<?php
$racine = simplexml_load_file('fichier.xml') ;
$body = $racine->body ;
$element_h1 = $body->h1 ;
echo $element_h1 ;
?>
```

#### Plusieurs nœuds

Si plusieurs éléments ont le même nom, `SimpleXML` renvoie un tableau indexé de ces éléments. On peut donc au choix y accéder avec leur index ou les lire un à un avec `foreach()`.

Le script suivant est illustré par la figure 20-6 :

```
<?php
$racine = simplexml_load_file('fichier.xml') ;
$body = $racine->body ;
```

```
// On récupère le premier paragraphe

```

**Figure 20-6**

*Affichage de plusieurs nœuds*

### Syntaxe unifiée

Pour éviter des résultats ambigus ou de faire trop de conditions, il est possible d'utiliser l'index 0 même s'il n'y a qu'un élément.

```
// on récupère le titre <h1>

```

### Lister les nœuds fils

Si vous souhaitez pouvoir accéder à tous les fils d'un élément, et pas seulement à ceux d'un certain nom, vous pouvez utiliser la méthode `children()`. Elle renverra un objet liste que vous pourrez utiliser de la même façon que ce que vous retournerait une recherche par nom.

```
<?php
// Accès au premier fils de l'élément <body>
$racine = simplexml_load_file('fichier.xml') ;
$body = $racine->body ;
$liste = $body->children() ;
echo $liste[0] ;
// Affiche le titre en <h1>,
// qui était le premier fils de <body>
?>
```

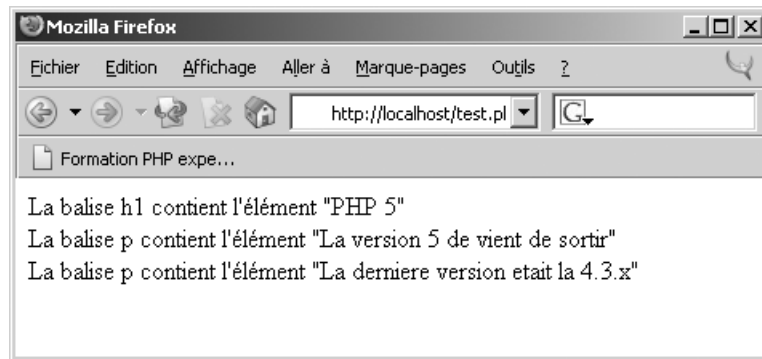


Vous pouvez aussi utiliser la syntaxe `foreach()` pour itérer à travers tous les éléments fils. Dans ce cas, il est possible de récupérer en même temps le nom de tous les éléments fils (voir script suivant et figure 20-7) :

```
<?php
// Réaffiche le contenu de <body>
$racine = simplexml_load_file('fichier.xml') ;
$body = $racine->body ;
foreach( $body->children() as $nom => $element ) {
    echo "La balise $nom contient l'élément \"", $element , "\"<br>";
}
?>
```

**Figure 20-7**

*Connaitre le nom des balises et leur contenu*



En fournissant un URI d'espace de noms à la méthode `children()`, seuls les éléments appartenant à cet espace de noms seront retournés.

### Ajouter un nœud fils

Depuis les dernières versions 5.2 de PHP, il est possible de modifier le document XML via SimpleXML. On ajoute un élément fils à l'aide de la méthode `addChild()`. Cette méthode prend en arguments le nom de la balise, son contenu textuel, et son URI d'espace de noms. Seul le premier paramètre est obligatoire.

```
<?php
// ajoute un paragraphe à la fin d'une page XHTML
// pour simplifier, nous n'utilisons pas d'espace de noms
$racine = simplexml_load_file('fichier.xml') ;
$body = $racine->body ;
$body->addChild("p", "PHP 5 avancé") ;
?>
```

#### Note

Les contenus textuels sont toujours à écrire en UTF-8. Pensez donc à utiliser la fonction `utf8_encode()` si vous n'utilisez pas Unicode habituellement.

## Afficher le contenu textuel d'un nœud

Pour afficher le contenu d'un élément, il suffit de passer l'élément aux fonctions `echo` ou `print`.

```
<?php
// Affichage du titre
$racine = simplexml_load_file('fichier.xml');
$body = $racine->body;
echo $body->h1 ;
?>
```

Attention toutefois, l'affichage du contenu se fait grâce à un artifice : PHP devra décider à chaque opération s'il traite la variable comme un objet représentant le nœud XML (ce qui est le cas par défaut) ou comme une chaîne de caractères représentant le contenu textuel (ce qui est le cas pour un affichage). PHP repère l'utilisation de la variable dans une instruction `echo` ou `print` et utilise à ce moment-là le contenu textuel du nœud au lieu de l'objet.

Pour d'autres fonctions, PHP ne pourra pas ou ne saura pas faire cette distinction. Vous pouvez alors forcer l'utilisation du contenu textuel en utilisant les instructions de transtypage (`string`) ou `strval()`. Pour plus de sûreté, vous devriez toujours faire appel à un transtypage explicite si vous destinez le texte à un traitement et non à un affichage direct.

```
<?php
// Affichage du titre
$racine = simplexml_load_file('monfichier.xml');
$element_body = $racine->body;
echo htmlentities((string) $body->h1 );
?>
```

### Note

Parfois, un élément contient plusieurs morceaux de texte séparés par des balises. Dans ce cas, SimpleXML concatène tous ces morceaux de texte en un seul avant de vous les retourner. Les nœuds textes contenus dans les balises filles ne sont pas retournés.

## Afficher un fichier RSS

Les fichiers RSS ont une structure définie comme nous l'avons expliqué en début de chapitre. Nous pourrions accéder aux différentes informations relayées dans le RSS grâce à une syntaxe simple :

```
<?php
$fichier_rss = 'http://www.afup.org/backend.php3';
$racine = simplexml_load_file($fichier_rss);
foreach($racine->channel->item as $news) {
    echo 'Actu : ', utf8_decode((string) $news->title) , '<br />';
}
?>
```

## Manipulation des attributs

Maintenant que nous savons parcourir l'arbre XML de manière simple, il est temps de nous intéresser aux attributs XML.

### Accéder à un attribut

Les attributs sont utilisables avec la même syntaxe que celles des tableaux associatifs. Ainsi, si `$html` est la balise `<html>` de notre exemple, `$html['lang']` représente son attribut `lang`. La valeur renvoyée sera un tableau des attributs `lang` de l'élément courant. Il peut en effet y avoir plusieurs attributs de même nom s'ils appartiennent à des espaces de noms différents.

```
<?php
// Langue du document XHTML
$html = simplexml_load_file('fichier.xml');
echo $html['lang'];
?>
```

### Lister tous les attributs

Il est possible de lister tous les attributs d'un nœud avec la méthode `attributes()`. PHP renvoie alors un tableau avec, pour chaque ligne, le nom de chaque attribut comme clé et son contenu comme valeur.

```
<?php
// Réécriture de la balise d'ouverture HTML
$html = simplexml_load_file('fichier.xml');
$balise = '<html' ;
foreach( $html->attributes() as $nom => $valeur ) {
    $balise .= " $nom='$valeur' " ;
}
$balise .= ">" ;
echo $balise ;
?>
```

Si vous spécifiez un URI d'espace de noms en argument à la méthode `attributes()`, seuls les attributs de cet espace de noms seront retournés.

### Modifier un attribut

Vous pouvez créer ou modifier un attribut en changeant directement sa valeur. Il est même possible de supprimer un attribut avec `unset()`.

```
<?php
// Passe la déclaration de langue en anglais
$html = simplexml_load_file('fichier.xml');
$html['lang'] = 'en' ;

$html->asXml('copie2.xml') ;
```

```
// Effacement de l'attribut
unset($html['attribut']);
?>
```

### Ajouter un attribut

L'ajout d'un attribut, lui, se fait avec la méthode `addAttribute()` sur l'élément parent. Le premier argument est le nom de l'attribut, le second argument est la valeur de l'attribut. Vous pouvez voir dans la figure 20-8 que l'attribut ajouté a bien été enregistré dans le fichier. Un troisième argument optionnel permet de spécifier l'URI de l'espace de noms.

```
<?php
// crée une déclaration de langue si elle n'existe pas
$html = simplexml_load_file('fichier.xml');
$html['lang'] = 'en';
$html->asXml('copie2.xml');
?>
```

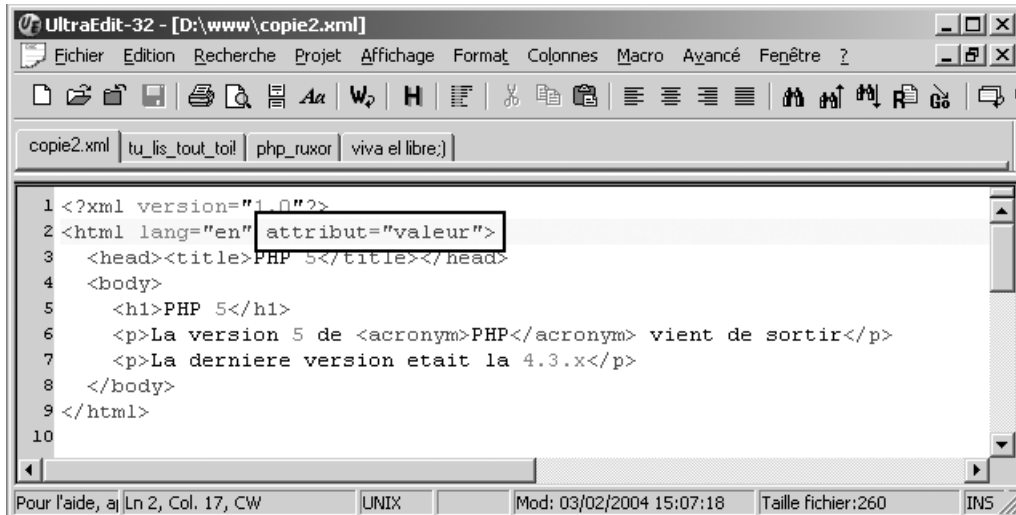


Figure 20-8

*Ajout d'attributs*

## Recherche Xpath

Pour certains, Xpath est au XML ce que SQL est aux bases de données. C'est le nom d'une syntaxe permettant de faire des recherches ou sélections dans un document XML. Il s'agit donc d'un moyen très important d'accéder aux données dans l'univers XML. Vous pourrez trouver les spécifications de sa très riche syntaxe sur le site du W3C à l'adresse <http://www.w3.org/TR/xpath>.

Pour notre exemple, nous utiliserons la recherche `/html/body/p`, qui sélectionne tous les paragraphes directement mis dans une page HTML (par directement, on entend qui ne soient pas imbriqués dans d'autres balises). Le premier `/` représente la racine du document XML, puis on avance en nommant les fils un à un.

Avec SimpleXML, vous pouvez faire une recherche Xpath grâce à la méthode `xpath()`.

```
$objet_simple_xml->xpath( expression_xpath );
```

En lui fournissant une expression Xpath en paramètre, elle retournera une liste d'objets SimpleXML ou la valeur FALSE en cas d'erreur. Le résultat de l'exemple suivant est donné à la figure 20-9 :

```
<?php
$xml = simplexml_load_file('fichier.xml') ;
>xpath = '/html/body/p' ; // Recherche des paragraphes
$paragraphes = $xml->xpath($xpath) ;
foreach( $paragraphes as $p ) {
    echo '<p>' , $p , "</p>\n";
}
?>
```

**Figure 20-9**

*Utiliser la recherche Xpath*



Si vous avez besoin d'utiliser un espace de noms dans une requête Xpath, il faut le déclarer préalablement avec la méthode `registerXPathNamespace()`. Elle prend un préfixe en premier argument et un URI d'espace de noms en second argument.

```
<?php
$xml = simplexml_load_file('fichier.xml') ;
$xml->registerXPathNamespace("h", "http://www.w3.org/1999/xhtml");
>xpath = '/h:html/h:body/h:p' ; // Recherche des paragraphes
$paragraphes = $xml->xpath($xpath) ;
foreach( $paragraphes as $par ) {
    echo '<p>' , $par , "</p>\n";
}
?>
```

## Extension des objets SimpleXML

Les différentes interfaces vues en amont pour SimpleXML utilisent les syntaxes objets de PHP pour fonctionner. Les objets dérivent tous d'une même classe `simpleXMLElement`.

Vous pouvez dériver cette classe avec une classe personnelle et demander à SimpleXML de l'utiliser dans l'interprétation de votre document. Une telle procédure peut être utile pour ajouter vos propres méthodes à l'extension.

Votre classe doit dériver de la classe `simpleXMLElement` et son nom doit être spécifié en second paramètre lors de l'ouverture du document.

```
<?php
class maSimpleXml extends simpleXMLElement {
    function getTitle() {
        $titles = $this->xpath('//title');
        if ($titles[0]) return (string) $titles[0];
        else return FALSE;
    }
}

$xml = '<file><title>mon titre</title></file>';
$file = simplexml_load_string($xml, 'maSimpleXml');
echo $file->getTitle();
// Affiche : mon titre
?>
```

## Cas d'application

### Lecture d'un fichier RSS

#### Contexte

Vous gérez un site régional. Pour fidéliser vos visiteurs, vous avez pris contact avec un site d'information local pour qu'il vous fournisse des brèves d'actualité.

Un accord a été trouvé et votre fournisseur vous met à disposition un fichier RSS 0.91 contenant les 15 dernières actualités de votre région. Votre rôle est de les réafficher dans une page dédiée.

Le fichier RSS fourni ressemble au suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="0.91">
  <channel>
    <title>Actualité 73</title>
    <link>http://fournisseur.contenu.fr/</link>
    <description>
      Toute l'actualité de la Savoie en direct
    </description>
  </channel>
  <item>
```

```
<title>Premier titre d'article</title>
<link>lien vers l'article compler 1</link>
<description>L'article 1 traite de la migration
  des grenouilles croates.
</description>
</item>
<item>
  <title>Second titre</title>
  <link>lien vers l'article compler 2</link>
  <description>résumé de l'article 2</description>
</item>
<item>
  <title>Titre 3</title>
  <link>lien vers l'article compler 3</link>
  <description>résumé de l'article 3</description>
</item>
[...]
```

```
</channel>
</rss>
```

## Réalisation

Dans un premier temps, vous préparez une page dédiée qui affiche les informations. Ces informations sont présentées suivant le modèle de la figure 20-10.

**Figure 20-10**

*Modèle d'affichage  
des informations*



Vous mettez donc en œuvre quelques lignes qui permettront de créer ce fichier :

```
<?php
$xml = simplexml_load_file('http://fournisseur.fr/contenu.rss') ;
echo '<html>';
echo '  <head><title>';
$title = (string) $xml->channel->title ;
$title = htmlentities($title, ENT_QUOTES, 'UTF-8') ;
echo $title ;
echo '  </title></head>';
echo '  <body>';
echo '    <h1>';
echo $title ;
echo '  </h1>';
foreach($xml->channel->item as $actu) {
  echo '<h2>';
  $href = htmlentities((string)$actu->link, ENT_QUOTES, 'UTF-8');
  echo "<a href='$href'>" ;
  echo htmlentities((string)$actu->title, ENT_QUOTES, 'UTF-8');
  echo '</a>';
  echo '</h2>';
  echo '<p>';
  $description = (string) $actu->description ;
  echo htmlentities($description, ENT_QUOTES, 'UTF-8');
  echo '</p>';
}
echo '  </body>';
echo '</html>';
?>
```





# 21

## XML avancé

---

Nous avons vu au chapitre précédent ce qu'était XML et comment on pouvait effectuer des traitements simples sur ce langage. Dans ce chapitre, nous allons passer à la vitesse supérieure et rentrer dans le détail des principales API (*Application Programming Interface*) XML. La première, l'interface SAX, permet d'interpréter finement des flux XML en lecture sans avoir besoin de charger l'ensemble des données en mémoire. La seconde, l'API DOM, permet d'utiliser le XML en lecture, en modification et en écriture. Sa syntaxe est très conséquente, mais elle fournit une méthode standardisée pour tout ce qu'il est possible de faire en XML. Enfin, le XSLT est un système de transformation XML, permettant de convertir un flux XML en un autre format (XML, XHTML, HTML ou PDF par exemple).

Les API DOM et SAX sont basées sur la bibliothèque `libxml2`, qui est maintenant fournie avec PHP. L'équipe de développement a activé ces extensions par défaut ; vous aurez donc l'assurance de les retrouver sur l'essentiel des configurations PHP 5.

### Relecture d'un XML avec SAX

L'API SAX (*Simple API for XML*) est un moteur événementiel dont le rôle est d'analyser un flux de données XML. Le moteur parcourt le flux à la recherche d'événements (rencontre d'une balise ouvrante, d'une balise fermante, d'un nœud de texte, etc.). À chaque fois qu'il rencontre un tel événement, il appelle les fonctions utilisateur que vous y aurez associées.

Le gros avantage de ce fonctionnement est sa faible consommation de ressources. En interprétant le flux au fil de l'eau, PHP n'a jamais besoin de charger le fichier dans son intégralité ; il peut l'utiliser par petits morceaux, provoquer les événements adéquats et

passer au morceau suivant. En revanche, le moteur ne gère rien lui-même : il se contente de lancer des événements et de faire appel à des fonctions définies par l'utilisateur. Il vous appartient donc de stocker les informations contextuelles (nom du nœud en cours, hiérarchie, etc.). Ce comportement rend parfois l'interprétation complexe ou peu naturelle comparée à l'utilisation de `SimpleXML`.

## Fonctionnement des événements

Le moteur SAX est un moteur d'événements. Son unique rôle est de détecter certains comportements et d'exécuter vos fonctions en conséquence. Ainsi, SAX lit le XML et peut nous avertir dès qu'il rencontre :

- des données textuelles ;
- une balise ouvrante ou fermante ;
- une entité externe connue ;
- une entité inconnue ;
- une instruction de traitement (Processing Instruction, PI) ;
- une nouvelle déclaration d'espace de noms.

À chaque fois que SAX rencontre un de ces schémas, il appelle la fonction que vous lui aurez désignée pour ce schéma. Lors de l'appel à la fonction, le moteur passe en paramètres les informations liées à cet événement.

Pour mieux vous faire comprendre, le tableau 21-2 présente dans l'ordre ce que ferait SAX pour le fichier suivant. À chaque ligne, SAX appelle une fonction utilisateur avec des arguments (le nom des fonctions est imaginaire, ces noms sont définis par vous-même lors de la phase préparatoire) :

```
<html lang="fr">
  <head><title>PHP 5</title></head>
  <body>
    <p>La dernière version était la 4.3.x</p>
  </body>
</html>
```

Tableau 21-1 Liste des types de nœud

Texte rencontré	Fonction/événement appelé	Premier paramètre
<html lang="fr">	ouvreElement()	'html' , array('lang'=>'fr')
(espaces)	texte()	(espaces)
<head>	ouvreElement()	'head'
<title>	ouvreElement()	'title'
PHP 5	texte()	'PHP 5'
</title>	fermeElement()	'title'

Tableau 21-1 Liste des types de nœud (*suite*)

Texte rencontré	Fonction/événement appelé	Premier paramètre
</head>	fermeElement()	'head'
(espaces)	texte()	(espaces)
<body>	ouvreElement()	'body'
(espaces)	texte()	(espaces)
<p>	ouvreElement()	'p'
La dernière version était la 4.3.x	texte()	'La dernière version était la 4.3.x '
</p>	fermeElement()	'p'
(espaces)	texte()	(espaces)
</body>	fermeElement()	'body'
(espaces)	texte()	(espaces)
</html>	fermeElement()	'html'

**Note**

Vous remarquerez que les espaces et les sauts à la ligne présents dans le fichier XML provoquent une réaction de la part du *parseur*. Vous pouvez généralement ignorer tout événement qui ne contient que des espaces blancs (espaces et fin de ligne).

**Limitations des événements**

Le moteur SAX ne vous donne accès à aucune information de contexte quand il réagit à un événement. Cela sous-entend que, quand SAX appelle la fonction à l'ouverture d'un élément, il vous donne en paramètre son nom et ses attributs. Vous ne connaissez dans cette fonction ni le contenu de l'élément rencontré (il fera l'objet d'un appel à une fonction propre) ni la position de l'élément dans l'arbre XML (quels parents, frères précédents ou suivants, fils, etc.). De même, quand SAX vous renvoie un nœud de texte, il ne vous dit pas dans quel élément il est. C'est à vous de tenir à jour une variable ou une pile avec le dernier élément ouvert et de la relire quand vous recevez du texte.

De plus, comme le moteur SAX fonctionne sur un flux, il est tout à fait envisageable que ce qu'il envoie ne soit pas complet. Le moteur envoie des données dès qu'il les reçoit. S'il reçoit un texte en deux fois, il est possible qu'il lance deux événements texte au lieu d'un seul, chacun avec la moitié du texte. C'est à vous de vous rendre compte que ces deux nœuds de texte se suivent et éventuellement de les concaténer.

**Initialisation**

L'initialisation du moteur passe par un appel à la fonction `xml_parser_create()`. Il vous sera renvoyé une ressource représentant le moteur SAX ; cette valeur sera utilisée par la suite.

```
$sax = xml_parser_create();
```

## Jeux de caractères

Par défaut, le moteur utilise le jeu de caractères ISO-8859-1 pour le flux. Vous pouvez spécifier un autre jeu de caractères en passant son identifiant en paramètre à `xml_parser_create()`. Les identifiants reconnus sont les suivants : ISO-8859-1, UTF-8 et US-ASCII.

```
■ $sax = xml_parser_create('UTF-8') ;
```

Lorsque SAX rencontre du texte, il le renvoie à vos fonctions dans son codage d'origine, sans conversion ni transformation. Il est possible de demander à PHP d'opérer automatiquement une conversion en faisant appel à `xml_parser_set_option()` avec la constante `XML_OPTION_TARGET_ENCODING` comme premier argument et l'identifiant du jeu de caractères à utiliser comme second argument.

```
■ xml_parser_set_option( XML_OPTION_TARGET_ENCODING, 'ISO-8859-1' );
```

N'oubliez pas que le codage caractère de PHP par défaut est l'ISO-8859-1 et que certaines fonctions ne marcheront correctement qu'avec ce codage si vous n'avez pas activé le module `mbstring` (voir le chapitre 5 sur les traitements de chaînes de caractères). Pour convertir une chaîne d'un codage à l'autre, vous pouvez utiliser `utf8_encode()` et `utf8_decode()`, vues dans le chapitre précédent.

## Utilisation des espaces de noms

Par défaut, le moteur SAX ignore toutes les déclarations d'espaces de noms et ne s'occupe que du nom court des balises. Vous pouvez lui demander de tenir compte des espaces de nom en utilisant la fonction `xml_parser_create_ns()` à la place de `xml_parser_create()`.

Si vous utilisez `xml_parser_create_ns()`, le moteur ajoute automatiquement l'URI de l'espace de noms devant toutes les balises et tous les attributs. Le nom de la balise et son espace de noms sont séparés par défaut par le symbole « : ». Il est toutefois possible de choisir votre propre séparateur en le fournissant comme second argument à la fonction.

## Utiliser la programmation orientée objet

Dans la suite, pour gérer des événements, nous utiliserons des fonctions de rappel. Il faudra fournir au moteur SAX des noms de fonctions à appeler quand il lancera ses événements. Nous verrons par la suite comment décrire ces fonctions et les utiliser.

Pour les *aficionados* de l'objet, il est possible d'associer toutes ces fonctions de rappel à un objet. Pour cela, il faut définir l'objet à utiliser via la fonction `xml_set_object()`. Dès lors, les noms donnés représenteront des méthodes de cet objet et non plus des fonctions globales.

```
■ // Création d'une classe
class gestion_sax { /* . . . */ }
// Instanciation de l'objet
$obj = new gestion_sax();
// Assignation des fonctions de rappel à un objet
xml_set_object( $obj );
```

## Réagir à des événements

SAX fonctionne sur une base événementielle : on associe des actions à chaque événement. Pour les exemples suivants, nous utiliserons un fichier de type RSS. Notre but sera de récupérer ses titres et de les stocker dans un tableau pour les traiter ultérieurement, par exemple pour effectuer une fusion avec un autre fil RSS, ou créer une page HTML contenant les liens vers les articles complets. Voici le fichier utilisé :

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="0.91">
  <channel>
    <title>Fil RSS</title>
    <link>http://filrss.com</link>
    <description>Description du fil RSS</description>
    <language>fr-fr</language>
    <item>
      <title>PHP 5</title>
      <link>http://www.php.net/</link>
      <description>PHP 5 est sorti</description>
    </item>
    <item>
      <title>PHP 4.3.2</title>
      <link>http://www.php.net</link>
      <description>
        La version 4.3.2 est en ligne.
      </description>
    </item>
  </channel>
</rss>
```

### Actions relatives aux textes

La première chose à récupérer dans un fichier XML est probablement le contenu textuel du fichier. Quand il rencontre un tel nœud, SAX appelle la fonction qui a été enregistrée avec `xml_set_character_data_handler()`.

Cette fonction de rappel doit accepter deux paramètres : une référence au moteur (la ressource retournée par `xml_parser_create()`) et une chaîne de caractères.

Dans notre exemple, nous allons parcourir le fichier RSS et réagir à tous les événements textuels. Le résultat du script est visible en figure 21-1.

```
<?php
$fichier_rss = file_get_contents('fichier.rss');
$sax = xml_parser_create('UTF-8') ;

// Affiche directement à l'écran tout le contenu textuel
function noeud_texte( $sax, $texte ) {
    $texte = trim($texte) ;
```

```
// Si le texte ne contient que des espaces, on n'affiche rien
if (empty($texte)) return ;
// Sinon, on affiche son contenu
echo 'Détection : --- <b>';
echo utf8_decode( $texte ) ;
echo '</b>--- : Fin de détection <br>';
}
// On assigne la fonction nœud_texte à tout événement textuel
xml_set_character_data_handler($sax, 'noeud_texte') ;

xml_parse($sax,$fichier_rss);
?>
```

**Figure 21-1**

*Affichage des nœuds  
de texte*



Attention, n'oubliez pas que le moteur SAX fonctionne sur un flux. Il peut très bien, à un moment précis, ne pas encore avoir reçu toutes les données. Il pourrait alors recevoir seulement une partie d'un texte, lancer un événement pour ce texte, et renvoyer ce qui suit par un deuxième événement.

**Note**

Le moteur SAX renvoie tous les nœuds texte rencontrés et garde tous les espaces blancs tels quels. Il envoie un événement texte même pour quelques espaces entre deux balises. C'est à vous qu'il revient de faire le tri entre les espaces significatifs et les autres. Dans notre exemple, nous n'avons affiché que les textes contenant autre chose que des espaces blancs.

## Balises ouvrantes et fermantes

Gérer les nœuds de texte est utile mais pas suffisant. En l'état, notre moteur ne fera que retirer tout le formatage XML et laissera le texte brut, ce qui n'est probablement pas ce que vous souhaitez.

Vous pouvez enregistrer une fonction de rappel qui sera utilisée quand le moteur rencontrera une balise. Une seule fonction SAX sert à enregistrer les événements pour les balises ouvrantes et les balises fermantes : `xml_set_element_handler()`. Elle prend deux paramètres en plus de l'identifiant du moteur SAX : un nom de fonction pour quand SAX rencontre une balise ouvrante et un nom de fonction pour quand il rencontre une balise fermante.

```
xml_set_element_handler(idsax, fctdebut, fctin)
```

La fonction de rappel utilisée pour les balises ouvrantes doit accepter trois arguments : une ressource désignant le moteur SAX (telle que retournée par la fonction `xml_parser_create()`), un nom pour la balise et une liste d'attributs référencés par leur nom.

La fonction de rappel pour les balises fermantes doit accepter deux paramètres : l'identifiant du moteur SAX et le nom de la balise à fermer.

## Casse des noms de balise

Par défaut, les noms des éléments sont transformés en majuscules par le moteur SAX. Pour éviter cette conversion, vous pouvez utiliser la fonction `xml_parser_set_option()` en passant comme paramètres la ressource du moteur puis les constantes `XML_OPTION_CASE_FOLDING` et `FALSE`.

```
xml_parser_set_option($sax, XML_OPTION_CASE_FOLDING, FALSE) ;
```

## Exemple

La liste d'événements suivante sert à afficher le titre du flux RSS donné en exemple plus en amont dans ce chapitre. Vous pouvez remarquer que si la logique permet de traiter des cas complexes, quand vous comptez juste afficher un titre du flux RSS, l'utilisation de SimpleXML est peut-être plus pertinente.

```
<?php
// Affiche le titre du fil RSS
class affiche_titre {
    private $chemin ;

    // Affiche le contenu si la balise est rss/channel/title
    public function noeudTexte( $sax, $texte ) {
        $texte = trim($texte) ;
        // si ce n'est pas un titre, on n'affiche rien
        if ($this->chemin != '/rss/channel/title') return ;
        // sinon, on affiche son contenu
        echo utf8_decode( $texte ) ;
    }
}
```



```
// balise ouvrante
public function baliseOuvrante( $sax, $nom, $attributs ) {
    $this->chemin .= "/"$nom" ;
}
// Balise fermante
public function baliseFermante( $sax, $nom ) {
    $longueur = strlen($nom) + 1 ;
    $this->chemin = substr( $this->chemin, 0, - $longueur ) ;
}
}
$sax = xml_parser_create('UTF-8') ;
$xml = new affiche_titre();
xml_set_object($sax, $xml) ;
xml_parser_set_option($sax, XML_OPTION_CASE_FOLDING, FALSE) ;
xml_set_character_data_handler($sax, 'noeudTexte') ;
xml_set_element_handler($sax, 'baliseOuvrante', 'baliseFermante');

// On charge le fichier rss
$fichier_rss = file_get_contents('fichier.rss');

// On exécute l'ensemble
xml_parse($sax,$fichier_rss, TRUE);
?>
```

**Note**

Nous venons de voir que pour retirer une fonction de rappel il suffit de renvoyer l'événement vers NULL. On peut utiliser cette procédure pour enregistrer et retirer des événements à la volée en fonction de la position dans l'arbre XML.

**Instruction de traitement**

Pour récupérer le contenu des instructions de traitement (*Processing Instructions* en anglais, ce qui est entre `<? et ?>`), vous pouvez enregistrer votre fonction de rappel avec `xml_set_processing_instruction_handler()`. Votre fonction doit accepter trois paramètres : l'identifiant du moteur SAX, la cible de l'instruction de traitement (mot collé juste après le `<?>`) et son contenu.

```
function pi($sax, $cible, $contenu) {
    echo "<?$cible ", htmlentities($contenu), " ?>" ;
}
xml_set_processing_instruction_handler($sax, 'pi') ;
```

**Autres composants et actions par défaut**

Vous pouvez enregistrer des fonctions de rappel pour d'autres événements comme les entités externes ou les déclarations d'espaces de noms. Décrire toutes les possibilités dépasse le cadre de ce livre, car on touche à des fonctionnalités plus poussées de XML.

Vous pouvez consulter la documentation officielle à ce sujet, à l'adresse <http://fr.php.net/manual/fr/ref.xml.php>.

Il est toutefois utile de vous décrire un dernier événement : il est lancé quand rien d'enregistré ne correspond à ce qui est rencontré. Vous pouvez enregistrer une fonction pour cet événement via `xml_set_default_handler()`.

La fonction enregistrée pour cet événement doit accepter deux paramètres : l'identifiant du moteur SAX et une chaîne de caractères représentant la donnée rencontrée. Vous pourrez alors gérer l'élément à la main, quel qu'il soit.

```
function default($sax, $donnee) {
    echo $donnee ;
}
xml_set_default_handler($sax, 'default') ;
```

## Envoi des données et analyse

Une fois tous les éléments décrits, il reste à alimenter le moteur avec le contenu XML. L'envoi du contenu se fait via la fonction `xml_parse()`. Elle prend en premier paramètre la ressource identifiant le moteur, en deuxième une chaîne de caractères représentant du XML, et en dernier un booléen qui décrit si l'envoi du source XML est fini ou pas. Il est ainsi possible d'alimenter le moteur au fur et à mesure de la lecture d'un fichier ou d'une requête réseau.

```
$fp = fopen($fichier , 'r') ;
while ($xml = fread($fp, 1024) {
    xml_parse($sax , $xml, feof($fp)) ;
}
```

Pendant que vous alimentez le moteur, les différentes fonctions sont appelées au fur et à mesure de l'interprétation. Notez bien que, pour un même texte, rien ne garantit qu'il n'y aura qu'un seul appel au gestionnaire de texte : SAX peut très bien faire un appel avec la moitié du texte puis un appel avec la moitié suivante.

Une fois le document entièrement interprété, il est bon de libérer la mémoire utilisée par le moteur à l'aide d'un appel à `xml_parser_free()`. La fonction prend en paramètre la ressource désignant le moteur SAX utilisé.

```
xml_parser_free($sax) ;
```

## Exemple de fonctionnement

Voici la résolution de l'exemple donné avec le moteur SAX. Le visuel du résultat est présenté à la figure 21-2.

```
<?php
class rss {
    var $item = FALSE ;
    var $chem = '' ;
    var $lien ;
```

```
var $titre ;
var $descr ;
var $resume = array() ;

function ouvre($sax, $nom, $attributs) {
    $this->chem .= '/' . $nom ;
    if ($this->chem == '/rss/channel/item') {
        $this->lien = '' ;
        $this->titre = '' ;
        $this->descr = '' ;
        $this->item = TRUE ;
    }
}

function ferme($sax, $nom) {
    if ($this->chem == '/rss/channel/item') {
        $lien = htmlentities($this->lien, ENT_QUOTES, 'UTF-8') ;
        $titre = htmlentities($this->titre, ENT_QUOTES, 'UTF-8') ;
        $descr = htmlentities($this->descr, ENT_QUOTES, 'UTF-8') ;
        echo "<h1><a href='$lien'>$titre</a></h1>" ;
        echo "<p>$descr</p>" ;
    }
    $pos = strrpos($this->chem, '/') ;
    $this->chem = substr($this->chem, 0, $pos) ;
}

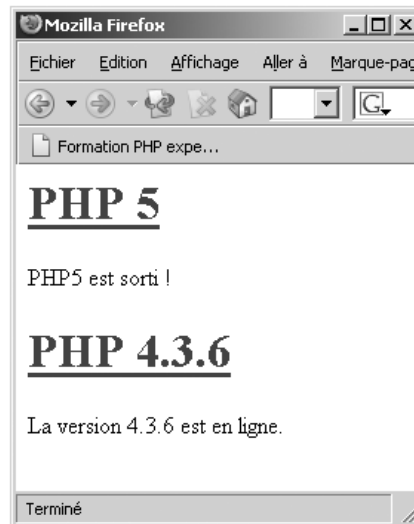
function texte($sax, $texte) {
    if ($this->chem == '/rss/channel/item/link') {
        $this->lien .= $texte ;
    } elseif ($this->chem == '/rss/channel/item/title') {
        $this->titre .= $texte ;
    } elseif ($this->chem == '/rss/channel/item/description') {
        $this->descr .= $texte ;
    }
}

}

$rss = new rss() ;
$sax = xml_parser_create() ;
xml_parser_set_option($sax, XML_OPTION_CASE_FOLDING, FALSE) ;
xml_set_object($sax, $rss) ;
xml_set_element_handler($sax, 'ouvre', 'ferme') ;
xml_set_character_data_handler($sax, 'texte') ;
$fichier = 'fichier.rss' ;
$fp = fopen($fichier, 'r') ;
while ($xml = fread($fp, 1024)) {
    xml_parse($sax, $xml, feof($fp)) ;
}
xml_parser_free($sax) ;
?>
```

Figure 21-2

Lire un fichier RSS



## Manipulation avec DOM

Nous avons vu comment créer un document XML à la main, comment relire un fichier avec SimpleXML et comment interpréter un flux XML avec SAX. Il nous manque une interface pour modifier et manipuler de manière précise un document XML ; c'est là que l'API DOM (*Document Object Model*) intervient.

Le principe de DOM est très différent de celui de SAX. En effet, le moteur va construire un arbre en mémoire représentant le document XML. Toutes les opérations suivantes auront donc lieu sur la représentation en mémoire. Il s'agit d'une API standardisée par le W3C (*World Wide Web Consortium*) permettant, entre autres, de manipuler des documents XML : créer des nœuds, les déplacer, ajouter des attributs ou des fils, détruire des sous-arbres, etc.

### Compatibilité et standardisation

Les méthodes et procédures de manipulation DOM sont exactement les mêmes dans tous les langages. Vous pouvez passer d'un langage à l'autre sans avoir à apprendre plusieurs noms de fonctions et plusieurs API.

L'interface DOM vise à être exhaustive sur les fonctionnalités XML. Tout ce qui est faisable en XML sera faisable via DOM, plus ou moins simplement. Elle n'a que deux défauts :

- Le premier est aussi un de ses avantages, c'est sa verbosité. Le code DOM est clair et facile à relire mais peut se révéler long et trop détaillé par rapport à SimpleXML si vous n'avez pas besoin de fonctionnalités avancées.

- Le second est que, contrairement à SAX, DOM nécessite d'avoir un document complet. Tout le document est interprété en une fois et chargé en mémoire pour construire une structure spécifique. Il est impossible de charger un document au fur et à mesure comme avec SAX et de n'avoir que quelques lignes en mémoire. Sur de gros documents (1 Mo et plus), la charge de votre serveur s'en ressentira.

#### Compatibilité avec PHP 4

Le module DOM de PHP 5 n'a plus grand-chose à voir avec l'ancien DOMXML de PHP 4. Il vise maintenant la conformité totale avec les spécifications, et l'implémentation de l'essentiel des niveaux 2 puis 3 de la norme.

Les fonctionnalités DOM sont trop complexes et trop importantes en quantité pour être détaillées ici complètement. Nous ne décrivons que les méthodes les plus utiles et vous laisserons vous reporter à la documentation officielle de DOM sur le site du W3C, à l'adresse <http://w3.org/DOM>, pour la suite.

## Structure générale

Lorsqu'on interprète un document via le moteur DOM, PHP crée automatiquement une série d'objets et de relations entre eux pour représenter l'arbre XML. Vous avez alors tout loisir de lire ces objets, de les modifier et enfin d'enregistrer dans un fichier le XML produit.

Une fois le XML interprété, tout sera présenté sous forme d'objets : le document sera un objet `DomDocument`, les éléments seront des objets `DomElement`, et ainsi de suite.

#### Gestion des flux PHP

Le moteur DOM a été entièrement intégré à PHP. Toutes les fonctions qui utilisent des fichiers ou URI peuvent utiliser les fonctions d'abstraction de flux PHP, y compris la fonction `Xpath document()`. Vous trouverez plus de détails à ce sujet au chapitre 14 dédié à la gestion des flux.

## Gestion des erreurs

Les spécifications DOM du W3C demandent une gestion des erreurs par exceptions. Toutes les exceptions DOM dérivent de la classe `DomException`. Vous trouverez plus de détails sur la gestion des erreurs et exceptions au chapitre 19.

```
try {  
    // Code DOM avec erreurs potentielles  
} catch(DomException $e) {  
    // Traitement de l'erreur  
}
```

## Codages caractères

La bibliothèque `libxml2` sur laquelle est basé le module `DOM` de PHP utilise le codage UTF-8 en interne, et non l'ISO-8859-1 de PHP par défaut. Attention aux incompatibilités, car toutes les données envoyées et reçues utiliseront ce format. N'oubliez pas d'utiliser les fonctions `utf8_encode()` et `utf8_decode()` en cas de besoin.

## L'objet document

L'objet document représente ce qui contient les données XML. Pour prendre l'analogie avec une feuille de papier classique, le document est la feuille et les données sont le texte écrit dessus.

### Création d'un document

La création d'un document XML est aussi simple que la création de n'importe quel objet dans PHP. Il suffit d'instancier la classe `DomDocument` (définie par défaut par le module PHP).

```
■ $document = new DomDocument();
```

### Chargement des données XML

Une fois le document créé, vous pouvez d'ores et déjà le modifier pour lui ajouter des éléments. Nous commencerons toutefois les explications à partir d'un fichier XML existant.

#### Charger un fichier XML

Pour charger un fichier XML, vous devez appeler la méthode `load()` de l'objet document en fournissant une adresse de fichier.

```
■ <?php
  $document = new DomDocument();
  $document->load('monfichier.xml');
?>
```

#### Charger une chaîne XML

Il est aussi possible de charger directement le XML depuis une chaîne de caractères grâce à la méthode `loadXML()` :

```
■ $xml = "<livre><titre>PHP 5 avancé</titre></livre>";
  $document = new DomDocument();
  $document->loadXML($xml);
```

## Charger un fichier HTML

Le moteur DOM de PHP sait gérer l'extension HTML de l'API DOM. Il est ainsi possible d'ouvrir un fichier HTML en DOM et de le manipuler comme si c'était du XML. Il suffit d'utiliser la méthode `loadHtmlFile()` à la place de `load()`. Les documents XHTML conformes peuvent quant à eux être chargés comme tout document XML valide.

```
$document = new DomDocument() ;
$document->loadHtmlFile('http://www.php.net/') ;
```

## Import depuis SimpleXML

Si vous avez utilisé SimpleXML pour lire rapidement un fichier et que vous souhaitez faire quelques manipulations DOM, il est possible d'importer l'objet SimpleXML pour construire un objet DOM de manière transparente. Il suffit d'utiliser la fonction `dom_import_simplexml()` avec l'objet SimpleXML en argument. La fonction inverse, `simplexml_import_dom()`, est décrite au chapitre précédent traitant de SimpleXML.

```
<?php
$s = simplexml_load_file('fichier.xml');
$dom = dom_import_simplexml($s);
print $dom->ownerDocument->saveXML();
?>
```

## Accéder à l'élément racine

Contrairement à SimpleXML, l'objet document de DOM ne correspond pas à l'élément racine. Vous pouvez accéder à l'élément racine en cherchant l'attribut `documentElement` de l'objet document.

L'élément dit « élément racine » correspond au nœud qui contient le premier élément ; on peut le comparer à l'adresse / sur un système de fichier Unix (le répertoire qui contient tous les autres). Les différentes balises qu'il pourrait contenir descendent de cette racine.

```
<?php
$document = new DomDocument() ;
$document->load('fichier.xml') ;
$racine = $document->documentElement ;
?>
```

## Accéder au document depuis un nœud

La fonction inverse de l'attribut `documentElement` s'appelle `ownerDocument`. Il permet de récupérer l'objet document à partir d'un nœud quelconque.

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

```
$racine = $document->documentElement ;  
$document = $racine->ownerDocument ;  
?>
```

## Sauvegarde des données

Pour sauvegarder vos documents XML à partir de l'objet `document`, il est possible de récupérer le contenu dans une chaîne de texte ou directement dans un fichier.

La méthode `save()` prend en paramètre un chemin de fichier (y compris distant, par FTP par exemple) pour sauvegarder les données XML.

```
<?php  
$document = new DomDocument() ;  
$document->load('fichier.xml') ;  
$document->save('fichier2.xml') ;  
?>
```

La méthode `saveXML()` renvoie, elle, le contenu sous forme d'une chaîne de caractères.

```
<?php  
$document = new DomDocument () ;  
$document->load('fichier.xml') ;  
echo $document->saveXML() ;  
?>
```

## Sauvegarder en HTML

De même qu'il est possible d'ouvrir un document HTML, il est possible de sauvegarder en HTML plutôt qu'en XML, en utilisant la méthode `saveHTML()` à la place de `saveXML()`.

L'affichage d'un document XHTML (donc XML) dans la syntaxe HTML ne nécessite que peu de différences. Les changements les plus importants sont les suivants :

- Le moteur ne ferme pas les balises vides (on note `<br>` et non `<br />`).
- Les attributs XHTML tels que `selected="selected"` sont transformés en attributs vides (simplement `selected`, sans valeur).

## Description d'un nœud

En XML, on désigne sous le terme générique « nœud » chaque partie du document XML. Un élément est un nœud, les attributs et les textes en sont aussi.

### Type de nœud

Les nœuds sont des objets de la classe `DOMNode`. Aucun objet ne devrait appartenir directement à cette classe. Ils sont toujours d'une sous-classe spécialisée (`DomElement`, `DomAttribute`, etc.).



Vous pouvez connaître le type d'un nœud à partir de son attribut propriété `nodeType`. Il retourne un entier, mais plusieurs constantes sont prédéfinies pour vous permettre une utilisation simplifiée (voir tableau 21-2).

**Tableau 21-2 Liste des types de nœuds**

Valeur	Signification	Constante
1	élément	XML_ELEMENT_NODE
2	attribut	XML_ATTRIBUTE_NODE
3	nœud de texte	XML_TEXT_NODE
4	section CDATA	XML_CDATA_SECTION_NODE
5	référence d'entité externe	XML_ENTITY_REF_NODE
6	entité	XML_ENTITY_NODE
7	instruction de traitement	XML_PI_NODE
8	commentaire	XML_COMMENT_NODE
9	document	XML_DOCUMENT_NODE

```
<?php
$xml = "<livre>Alice</livre>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
//On se place au niveau du premier nœud
$livre = $document->documentElement ;
echo $livre->nodeType ; // Affiche 1
$texte = $livre->firstChild ;
echo $texte->nodeType ; // Affiche 3
?>
```

### Nom d'un nœud

Il existe deux attributs de l'objet `DOMNode` qui permettent de récupérer le nom d'un nœud : `nodeName` et `tagName`. Pour un élément, ces attributs retournent tous les deux le nom de l'élément (<livre> donnera « livre »). Pour les autres types de nœuds, `tagName` retourne un nom indéfini tandis que `nodeName` donne généralement le type du nœud (par exemple `#text` pour les nœuds de texte).

```
<?php
$xml = "<livre>Alice</livre>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;
echo $livre->nodeName ; // Affiche livre
echo $livre->firstChild->nodeName ; // Affiche #text
?>
```

## Contenu d'un nœud

Le contenu d'un nœud s'obtient en lisant son attribut `nodeValue`. On peut ainsi obtenir la valeur d'un attribut ou d'un nœud de texte. Il est toutefois important de souligner qu'un élément n'a pas de valeur, ce sont éventuellement ses fils de type nœuds de texte qui en ont.

```
<?php
$xml = "<livre type='conte'>Alice</livre>";
$document = new DomDocument();
$document->loadXML($xml);
$livre = $document->documentElement;
echo $livre->nodeName; // Affiche livre
echo $livre->firstChild->nodeValue; // Affiche Alice
$type = $livre->getAttributeNode('type');
echo $type->nodeValue; // Affiche conte
?>
```

## Navigation dans l'arbre

Jusqu'ici, nous avons manipulé des exemples XML relativement simples et nous n'avons pas navigué dans la structure de ces documents. Les fonctions suivantes nous permettent de faire des recherches ou des sélections dans l'arbre. Elles permettent par exemple de sélectionner les nœuds fils d'un élément, ou le nœud suivant, le nœud parent, etc.

### Liste des nœuds

Dans vos recherches, vous recevrez généralement un objet de type `DOMNodeList`. Il s'agit d'une liste de nœuds.

Cet objet implémente l'interface `Iterator` (voir le chapitre 12 sur la programmation orientée objet pour plus de détails) et vous pouvez donc le parcourir avec `foreach()`.

```
$nodeList; // Ojet de type DOMNodeList
foreach( $nodeList as $node ) {
    print_r( $node );
}
```

Vous pouvez aussi accéder à un item en particulier à l'aide de la méthode `item()` et d'un index numérique.

```
$nodeList->item(0); // Premier nœud de la liste
```

La quantité de nœuds présents dans une liste peut être récupérée avec l'attribut `length` de l'objet `DOMNodeList`.

```
echo "Il y a ", $node->childNodes->length, " nœuds fils";
```

## Nœuds fils

La liste des nœuds fils d'un nœud peut être connue via l'attribut `childNodes` du nœud père. L'objet renvoyé est un objet de type `DOMNodeList`. L'exemple suivant est illustré à la figure 21-3.

```
<?php
$xml = "
  <livre>
    <titre>PHP 5</titre>
    <auteur>E.D</auteur><auteur>C.PdG</auteur>
  </livre>" ;
$document = new DomDocument();
$document->loadXML($xml) ;
$livre = $document->documentElement ;
// Affichage des fils de $parent
foreach( $livre->childNodes as $node ) {
  if ( $node->nodeType == XML_ELEMENT_NODE ) {
    echo 'Balise : <b>' , $node->tagName, '</b><br>' ;
    echo 'Contenu : <b>';
    echo utf8_decode($node->firstChild->nodeValue), '</b><br>' ;
  }
}
?>
```

Figure 21-3

*Lister les éléments  
fils*



### Note

La liste des nœuds fils ne contient pas que les éléments ; les nœuds de texte sont aussi retournés, par exemple.

Il est possible d'accéder directement au premier ou au dernier nœud fils à l'aide des attributs `firstChild` et `lastChild`.

```
<?php
$xml = "<versions>
    <version>3</version>
    <version>4</version>
    <version>5</version>
</versions>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$versions = $document->documentElement ;
// On récupère le premier fils : l'élément version 3
$trois = $versions->firstChild ;
// On récupère le dernier fils : l'élément version 5
$cinq = $versions->lastChild ;
?>
```

Vous pouvez tester la présence de nœuds fils à l'aide de la méthode `hasChildNodes()`.

```
// Affichage des fils de $version
if ($versions->hasChildNodes()) {
    foreach( $versions->childNodes as $node ) {
        if ($node->nodeType == XML_ELEMENT_NODE) {
            echo $node->tagName, ' : ' ;
            echo utf8_decode($node->firstChild->nodeValue), '<br>' ;
        }
    }
}
```

#### Note

Des espaces vides entre deux balises seront vus comme des nœuds fils de type texte, ils ne sont pas ignorés.

## Nœud parent

Le nœud parent d'un objet peut être connu via l'attribut `parentNode` du nœud fils. Il renvoie un objet de type `DOMNode`.

```
$parent ;
$fils = $parent->firstChild ;
$parent = $fils->parentNode ;
```

## Nœuds frères

Pour accéder aux nœuds juste avant ou juste après le nœud courant dans l'arbre, il est possible d'accéder au nœud parent puis d'en lister les fils pour naviguer.

Plus simplement, il est aussi possible d'accéder directement au nœud frère précédent et au nœud frère suivant (s'ils existent) via les attributs `previousSibling` et `nextSibling`.

```
print_r( $node->childNodes->item(0) ) ;
print_r( $node->childNodes->item(1) ) ;
```

```
// Est équivalent à
$node = $node->childNodes->item(0) ;
print_r( $node ) ;
print_r( $node->nextSibling ) ;
```

### Recherche d'un élément par son identifiant

Plutôt que de parcourir l'arbre à la main en descendant avec `childNodes`, on peut rechercher directement un élément par son identifiant. Ce qu'est un identifiant dépend de votre modèle de données ; avec les documents XHTML, l'identifiant est l'attribut `id`.

En passant un identifiant à la méthode `getElementById()` vous récupérez (en cas d'existence) une référence du nœud ayant cet identifiant (donc un objet de type `DOMNode`).

```
$document = new DomDocument();
$document->load('fichier.xml') ;
$sommaire = $document->getElementById('sommaire') ;
```

### Recherche d'un élément par son nom

Vous pouvez aussi rechercher tous les éléments d'un certain nom en le donnant en argument à la méthode `getElementsByTagName()`. Cette méthode vous retourne une liste de nœuds de type `DOMNodeList`. En l'appliquant au document ou à sa racine, vous aurez une liste complète des éléments de ce nom. Vous pouvez aussi restreindre votre recherche en l'appliquant à un objet `domElement`. Le moteur ne cherche alors que parmi les nœuds fils et leurs descendants.

Le script suivant cherche les différents auteurs d'une liste et les affiche (voir figure 21-4) :

```
<?php
$xml = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>
    <auteurs>
        <auteur>
            <nom>Stéphane MARIEL</nom>
            <livre>PHP5</livre>
            <livre>PostgreSQL et PHP</livre>
        </auteur>
        <auteur>
            <nom>Eric DASPET, Cyril PIERRE de GEYER</nom>
            <livre>PHP5 avancé</livre>
        </auteur>
    </auteurs>" ;

$document = new DomDocument() ;
$document->loadXML($xml) ;

// Recherche des différents auteurs
$auteurs = $document->getElementsByTagName('auteur') ;
```

```

foreach($auteurs as $auteur) {
    $livres = array() ;
    $nom = '' ;
    foreach($auteur->childNodes as $child) {
        if ($child->nodeType != XML_ELEMENT_NODE) continue ;
        if ($child->tagName == 'nom') {
            $nom = utf8_decode($child->firstChild->nodeValue) ;
        } elseif($child->tagName == 'livre') {
            $livres[] = utf8_decode($child->firstChild->nodeValue) ;
        }
    }
    echo "<p>$nom : ", implode(', ', $livres), '</p>' ;
}
?>

```

Figure 21-4

Rechercher un  
élément par son nom



Si vous utilisez les espaces de noms, il est possible de spécifier un espace à utiliser pour le nœud recherché grâce à la fonction `getElementsByTagNameNS()`. Elle fonctionne de manière similaire à `getElementsByTagName()`, mais accepte deux arguments : le premier est l'espace de noms (sous forme d'URI, pas de préfixe) et le second est le nom de l'élément recherché.

```

<?php
$xml = "<ey:livre xmlns:ey='mon_uri'>PHP 5 avance</ey:livre>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livres = $document->getElementsByTagNameNS('mon_uri', 'livre') ;
foreach($livres as $livre) {
    echo $livre->firstChild->nodeValue ;
}
?>

```

## Gestion des attributs

### Lecture d'un attribut

Quand vous accédez à un élément (nœud de type `DomElement`), vous pouvez accéder à ses attributs. Pour lire le contenu d'un attribut dont le nom vous est connu, vous pouvez utiliser la méthode `getAttribute()` avec le nom de l'attribut comme argument. Vous trouverez le résultat du script suivant en figure 21-5.

```
<?php
define('DOM_ELEMENT', 1) ;
$xml = "<?xml version='1.0' encoding='iso-8859-1'?>
    <auteurs>
        <auteur>
            <nom>Stéphane MARIEL</nom>
            <livre collection='Les cahiers'>PHP5</livre>
            <livre collection='Les cahiers'>PostgreSQL et PHP</livre>
        </auteur>
        <auteur>
            <nom>Eric DASPET, Cyril PIERRE de GEYER</nom>
            <livre collection='Blanche'>PHP5 avancé</livre>
        </auteur>
    </auteurs>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$auteurs = $document->documentElement ;
foreach($auteurs->getElementsByTagName('auteur') as $auteur) {
    $livres = array() ;
    $nom = '' ;
    foreach($auteur->childNodes as $child) {
        if ($child->nodeType != DOM_ELEMENT) continue ;
        if ($child->tagName == 'nom') {
            $nom = utf8_decode($child->firstChild->nodeValue) ;
        } elseif($child->tagName == 'livre') {
            $collection= utf8_decode($child->getAttribute('collection')) ;
            $titre = utf8_decode($child->firstChild->nodeValue) ;
            $livres[] = "Collection $collection : $titre" ;
        }
    }
    echo "<h1>$nom</h1>" ;
    echo "<ul>" ;
    foreach($livres as $livre) {
        echo "<li>$livre</li>" ;
    }
    echo "</ul>" ;
}
?>
```

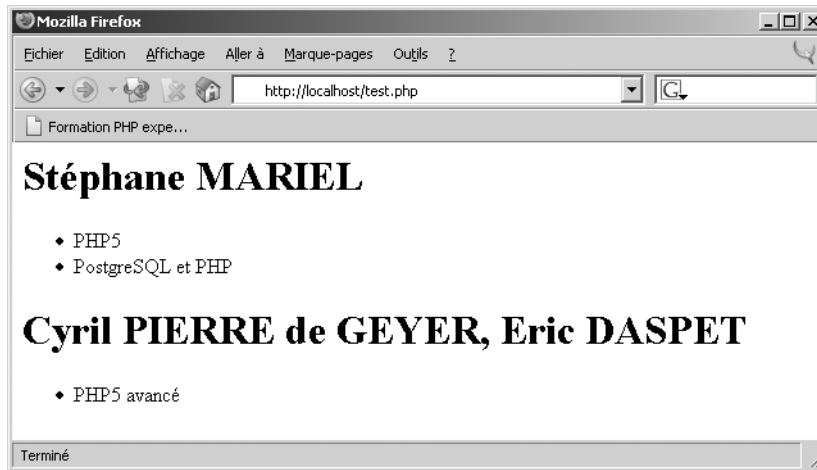


Figure 21-5

*Affichage des attributs*

### Modification ou création d'un attribut

Vous pouvez modifier le contenu d'un attribut par son nom de façon similaire, grâce à la méthode `setAttribute()`. La méthode prend en paramètres le nom de l'attribut et sa nouvelle valeur.

```
<?php
$xml = utf8_encode("<livre>PHP 5 avance</livre>");
$document = new DomDocument();
$document->loadXML($xml);
$livre = $document->documentElement;
$livre->setAttribute('collection', 'blanche');
// Affiche la valeur de l'attribut ajouté : blanche
echo $livre->getAttribute('collection');
// On enregistre le nouveau fichier
$document->save('fichier2.xml');
?>
```

En utilisant `setAttributeNS()` à la place de `setAttribute()`, vous pouvez spécifier un espace de noms à l'attribut. Vous devez alors fournir l'URI de l'espace de noms en premier paramètre, les nom et valeur en second et dernier.

### Effacement d'un attribut

La méthode `removeAttribute()` vous permet de retirer un attribut d'un élément à partir de son nom.

```
<?php
$xml = "<livre collection='blanche'>PHP 5 avancé</livre>";
$xml = utf8_encode($xml);
```



```
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;
$livre->removeAttribute('collection') ;
// On enregistre le nouveau fichier
$document->save('fichier2.xml') ;
?>
```

### Attributs en tant que nœuds

Il est aussi possible de traiter les attributs plus généralement, comme un nœud spécifique. La méthode `getAttributeNode()` fonctionne de manière similaire à `getAttribute()` mais retourne un nœud (objet de classe `DomAttribute`) à la place du contenu. Vous pouvez le manipuler comme un nœud classique, récupérer son nom, le déplacer dans l'arbre, etc.

```
<?php
$xml = "<livre collection='blanche'>PHP 5 avancé</livre>" ;
$xml = utf8_encode($xml) ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;
$collection = $livre->getAttributeNode('collection') ;
echo $collection->nodeName ; // Affiche collection
?>
```

### Liste des attributs

De même que vous pouvez lister tous les fils d'un nœud avec la propriété `childNodes`, vous pouvez lister tous les attributs d'un élément avec l'attribut `attributes`. Le fonctionnement est identique, sauf que la liste de nœuds ne contient que des objets de type `DomAttribute`.

```
<?php
$xml = "<livre collection='blanche'>PHP 5 avancé</livre>" ;
$xml = utf8_encode($xml) ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;

foreach($livre->attributes as $attribut) {
    echo $attribut->nodeName ; // Affiche collection
}
?>
```

### Création de nœuds

Le moteur DOM ne permet pas uniquement d'accéder au document XML en lecture, comme nous l'avons vu jusqu'à présent. Là où il fait la différence avec les API SAX et SimpleXML, c'est qu'il permet de modifier en profondeur le document et de créer de nouveaux nœuds (SimpleXML ne permet que les modifications sur les textes et attributs).

Attention, créer un nœud ne suffit pas à faire en sorte qu'il soit affiché dans le document XML. Il faut encore par la suite le greffer dans l'arbre à l'endroit souhaité.

Pour créer un nouveau nœud, on individualise plusieurs étapes :

- chargement du document DOM sur lequel travailler ;
- création d'un nouvel élément ;
- greffe de l'élément à la bonne position dans l'arbre ;
- sauvegarde.

Nous nous contentons pour l'instant de décrire la deuxième étape, la création des différents types de nœuds. La greffe de ce nœud dans l'arbre XML sera détaillée plus loin dans ce chapitre, avec les autres modifications de l'arbre.

### Création d'un élément

Pour créer un élément, vous devez faire appel à la méthode `createElement()` de l'objet document en lui passant le nom de l'élément en paramètre. L'objet élément retourné sera intimement lié au document dans lequel il est créé. Vous ne pouvez donc pas directement instancier la classe `DomElement`, car vous n'aurez pas cette relation avec le document.

```
<?php
$document = new DomDocument() ;
$livre = $document->createElement('livre') ;
echo $livre->nodeName ; // Affiche livre
?>
```

Il est possible de spécifier un espace de noms pour l'élément en utilisant plutôt la méthode `createElementNS()`, qui prend en arguments l'URI de l'espace de noms et le nom de l'élément.

### Création d'un nœud de texte

De manière similaire à `createElement()`, il est possible de créer un nœud de texte à partir de la méthode `createTextNode()`. Le contenu du nœud est à passer en argument.

```
<?php
$document = new DomDocument() ;
$livre = $document->createTextNode('PHP 5 avancé') ;
echo $livre->nodeValue ; // Affiche PHP 5 avancé
?>
```

### Création d'autres types de nœuds

Il est possible de créer tous les types de nœuds de la même façon que les éléments et les textes.

Les attributs sont créés avec la méthode `createAttribute()`, qui prend en arguments l'élément parent, le nom de l'élément et son contenu.

La méthode `createComment()` permet de créer un commentaire en spécifiant son contenu.

Les sections de texte CDATA utilisent la méthode `createCDATASection()` et ne prennent comme argument qu'une chaîne de caractères qui représente le contenu.

Il est possible de créer une référence vers une entité externe avec la méthode `createEntityReference()` et en spécifiant le nom de l'entité comme argument.

Enfin, pour créer une instruction de traitement (*Processing Instruction*, PI), on utilise la méthode `createProcessingInstruction()` avec deux paramètres : le nom de la cible et le contenu.

### Copie d'un nœud existant

Plutôt que de créer un nouveau nœud, il est aussi possible d'en copier un. La méthode `cloneNode()` d'un nœud permet de copier un nœud et renvoie la copie comme valeur de retour. Cette méthode accepte un argument booléen ; s'il est faux, seul le nœud est copié, s'il est vrai, c'est tout le sous-arbre (le nœud copié, tous ses fils et sa descendance).

```
<?php
$xml = "<livre collection='blanche'>PHP 5 avancé</livre>" ;
$xml = utf8_encode($xml) ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;
$copie = $livre->cloneNode(FALSE) ;
var_dump( $copie->childNodes() ) ; // Renvoie FALSE
// car les fils n'ont pas été copiés
?>
```

## Modification de l'arbre XML

Quand vous accédez à un nœud, il est possible de le déplacer dans l'arbre et de le greffer à un autre endroit. Vous pouvez ainsi déplacer un nœud existant ou insérer un nœud que vous avez préalablement créé.

### Insertion d'un nœud fils

La méthode `appendChild()` permet d'ajouter un fils au nœud actuel. Le fils est ajouté en dernier de la liste si d'autres nœuds fils existaient déjà.

```
<?php
$xml = "<livre></livre>" ;
$doc = new DomDocument() ;
$doc->loadXML($xml) ;
$txt = $doc->createTextNode(utf8_encode('PHP 5 avancé')) ;
$livre = $doc->documentElement ;
$livre->appendChild($texte) ;
// On enregistre le nouveau fichier
$doc->save('fichier2.xml') ;
?>
```

La méthode `insertBefore()` est similaire à `appendChild()`, mais permet d'insérer le nœud juste avant un autre, au lieu d'être inséré en dernier. Elle prend deux paramètres : le nœud à insérer et le nœud référence (l'insertion se fera juste avant ce dernier).

```
<?php
$xml = "<livre>avancé</livre>" ;
$xml = utf8_encode($xml) ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$php = $document->createTextNode('PHP 5 ' ) ;
$livre = $document->documentElement ;
$avance = $livre->firstChild ;
$livre->insertBefore($php, $avance) ;
foreach( $livre->childNodes as $child ) {
    echo $child->nodeValue ;
}
// Affiche PHP 5 avancé
?>
```

### Effacer un nœud

Il est aussi possible d'effacer un nœud, que ce soit un nœud créé ou un nœud déjà présent dans l'arbre. Il suffit de passer le nœud en paramètre à la méthode `removeChild()` de l'objet document.

```
<?php
$xml = "<livre>PHP 5 avancé<a-retirer /></livre>" ;
$xml = utf8_encode($xml) ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$livre = $document->documentElement ;
$livre->removeChild( $livre->lastChild ) ;
?>
```

### Remplacer un fils

Plutôt que d'ajouter un nœud avant un nœud existant, puis de retirer l'ancien, il est possible de remplacer directement un nœud par un autre en une opération grâce à la méthode `replaceChild()`. Le premier argument est le nœud à insérer et le second est le nœud à retirer.

```
<?php
$xml = "<livre>PHP 6 avancé</livre>" ;
$document = new DomDocument() ;
$document->loadXML($xml) ;
$php5 = $document->createTextNode('PHP 5 avancé') ;
$livre = $document->documentElement ;
$php6 = $livre->firstChild ;
$livre->replaceChild($php5, $php6) ;
echo $livre->firstChild->nodeValue ;
// Affiche PHP 5 avancé
?>
```

## Création d'un document complet

En cumulant les fonctions de création et de manipulation d'arbre, il est possible de créer un document complet depuis zéro :

```
<?php

// Création du document
$document = new DomDocument() ;

// On crée l'élément principal <livre>
$livre = $document->createElement('livre') ;
$document->appendChild($livre) ;

// On ajoute un <titre>
$titre = $document->createElement('titre') ;
$livre->appendChild($titre) ;
// et son contenu texte
$txt = utf8_encode('PHP 5 avancé') ;
$txt = $document->createTextNode($txt) ;
$titre->appendChild($txt) ;

// On ajoute maintenant un <auteur> au <livre>
$ericDaspet = $document->createElement('auteur') ;
$livre->appendChild($ericDaspet) ;
// et son contenu texte
$txt = utf8_encode('Eric Daspet') ;
$txt = $document->createTextNode($txt) ;
$ericDaspet->appendChild($txt) ;

// et un deuxième
$cyrilPierreDeGeyer = $document->createElement('auteur') ;
$livre->appendChild($cyrilPierreDeGeyer) ;
$txt = utf8_encode('Cyril Pierre de Geyer') ;
$txt = $document->createTextNode($txt) ;
$cyrilPierreDeGeyer->appendChild($txt) ;

// Affichage du résultat
echo $document->save('fichier.xml') ;
```

Le code précédent donnerait un fichier XML équivalent au suivant :

```
<?xml version="1.0">
<livre>
  <titre>PHP 5 avanc&xE9;</titre>
  <auteur>Eric Daspet</auteur>
  <auteur>Cyril Pierre de Geyer</auteur>
</livre>
```

**Note**

Nous parlons de XML équivalent car on peut écrire un même contenu de plusieurs façons. La version actuelle lors de nos tests met par exemple toutes les balises XML à la suite sans espaces ni sauts de ligne. On peut noter aussi la présence d'une entité dans l'exemple pour symboliser le e accentué. On aurait aussi pu choisir un jeu de caractères arbitraire dans le prologue XML et écrire directement le é dans ce codage au lieu d'utiliser l'entité.

## Recherche Xpath

Xpath est une spécification du W3C qui a pour but de fournir une interface de recherche et de sélection sur des données XML. La syntaxe Xpath ne fait pas partie du cadre de ce livre, vous pouvez cependant trouver des informations à l'adresse <http://www.w3.org/TR/xpath>.

### Initialisation du moteur

L'extension DOM de PHP permet de gérer des requêtes Xpath. Il faut pour cela instancier un objet qui gèrera la requête. Cet objet est de la classe `DomXpath` et attend un document DOM en argument pour son constructeur.

```
$document = new DomDocument() ;  
$document->loadXML($xml) ;  
$xpath = new DomXpath($document) ;
```

### Lancer une requête Xpath

Pour effectuer une recherche, il vous faut faire appel à la méthode `query()` avec la requête Xpath en argument.

```
<?php  
$xml = file_get_contents('fichier.xhtml');  
$xml = utf8_encode($xml) ;  
$document = new DomDocument() ;  
$document->loadXML($xml) ;  
$xpath = new DomXpath($document) ;  
  
// Recherche tous les formulaires à envoyer  
$result = $xpath->query( "/html/body//form[action='post']" ) ;
```

Par défaut, la recherche est faite à partir de l'élément racine. Il est toutefois possible de définir une autre base pour la recherche en spécifiant le nœud DOM référence en second argument.

```
$requete = "form[action='post']" ;  
$reference = $document->documentElement->lastChild ;  
$result = $xpath->query($requete, $reference) ;
```

Le résultat renvoyé est un objet liste de nœuds DOM classique, tel qu'on en a rencontré auparavant.

```
$document = new DomDocument() ;
$document->loadXML($xml) ;
$xpath = new DomXpath($document) ;
// Recherche tous les formulaires à envoyer
// avec la méthode POST
$result = $xpath->query( "/html/body//form[action='post']" ) ;
echo "Il y a ", $result->length, " formulaire(s) en POST" ;
```

### Gérer les espaces de noms

Si vous voulez utiliser les espaces de noms dans vos requêtes Xpath, il faut auparavant enregistrer les correspondances entre vos préfixes et les URI d'espaces de noms.

Vous pouvez définir une telle correspondance avec la méthode `register_ns()`. Elle prend en paramètres un préfixe et un URI d'espace de noms.

```
$document = new DomDocument() ;
$document->loadXML($xml) ;
$xpath = new DomXpath($document) ;
// Recherche tous les formulaires à envoyer
// avec la méthode POST
$xpath->register_ns('html', 'http://www.w3.org/1999/xhtml') ;
$requete = "/html:html/html:body//html:form[action='post']" ;
$result = $xpath->query($requete) ;
echo "Il y a ", $result->length, " formulaire(s) en POST" ;
```

### Extension des classes DOM

Tout le module DOM fonctionne avec des classes prédéfinies comme `DomDocument`. Il est possible d'étendre cette dernière classe pour y ajouter ses propres éléments.

Ainsi, pour une collection de livres, on peut imaginer la classe suivante :

```
<?php
class collection extends domDocument {
    public function ajouterLivre($titre) {
        $livre = $this->createElement('livre') ;
        $titre = $this->createTextNode($titre) ;
        $livre->appendChild($titre) ;
        $this->documentElement->appendChild($livre) ;
    }
}
$xml = "<collection><livre>PHP 4</livre></collection>" ;
$collection = new collection() ;
$collection->loadXML($xml) ;
$collection->ajouterLivre("PHP 5 avancé") ;
echo $collection->saveXML() ;
?>
```

**Important**

Si vous avez un constructeur dans la classe dérivée, vous devez absolument faire appel au constructeur DOM par `parent::__construct()` avant toute manipulation XML du document.

## Utilisation de Xinclude

Xinclude est une extension XML du W3C permettant d'inclure des documents dans d'autres documents avec de simples liens. Vous pouvez trouver plus d'informations à l'adresse <http://www.w3.org/TR/xinclude/>.

Pour effectuer les remplacements Xinclude (c'est-à-dire remplacer les liens par les contenus des documents liés), il vous suffit de faire appel à la méthode `xinclude()` de l'objet document de DOM.

```
$document = new DomDocument() ;  
$document->loadXML($xml) ;  
$document->xinclude() ;
```

## Validation et conformité

Utiliser un gros fichier XML est parfois délicat. Il peut être important de vérifier que sa structure correspond à ce qu'on attend pour éviter d'avoir un comportement erroné de l'application qui l'interprète.

Les DTD (*Document Type Definition*) sont les fichiers de description XML classiques. Ils sont relativement limités dans les contraintes imposées au document, mais sont livrés avec presque tous les formats.

Il est possible de vérifier la conformité d'un document avec un fichier DTD grâce à la méthode `validate()`. Elle prend en unique paramètre une adresse pour le fichier de grammaire à utiliser. Si la fonction renvoie la valeur `TRUE`, le document XML est considéré comme valide. Dans le cas contraire, c'est qu'il contient une erreur ou n'est pas conforme à la structure demandée.

```
$document = new DomDocument() ;  
  
if ( $document->loadXML($xml)  
    && $document->validate('fichier.dtd') ) {  
    echo 'Le document est un XML bien formé et conforme' ;  
} else {  
    echo 'Le document contient une erreur ou n'est pas conforme';  
}
```

Vous pouvez aussi spécifier la DTD sous forme d'une chaîne de caractères au lieu d'un chemin de fichier en utilisant la méthode `validateSource()` plutôt que `validate()`.

D'autres formats de validation plus stricts ou plus complets ont fait leur apparition pour combler les lacunes des DTD. PHP peut ainsi valider votre document en lisant un schéma XML ou un fichier relaxNG.



Pour utiliser un schéma XML, vous devez vous servir de `schemaValidate()` et `schemaValidateSource()`. Pour relaxNG, c'est `relaxNGValidate()` et `relaxNGValidateSource()`. Les comportements sont identiques aux fonctions `validate()` et `validateSource()`.

## Transformation XML par XSLT

La conversion d'un document XML en un autre format (XML ou non) est parfois complexe si on se contente des outils vus précédemment. L'analyse de l'arbre peut nécessiter beaucoup de code de bas niveau, diminuant la lisibilité et la simplicité de votre application. Les performances peuvent aussi être décevantes s'il s'agit de convertir d'un format XML vers un autre : il est peu efficace de recréer complètement le fichier de zéro.

C'est particulièrement vrai avec l'avènement du XHTML ; le contenu des pages web étant lui-même en XML, il est intéressant de pouvoir opérer une transformation directe de la source vers la page XHTML.

DOM contient quelques manipulations simples pour modifier un document, mais ne permettra qu'une correction et pas une réelle transformation ou conversion. Le langage adapté pour effectuer des transformations XML est XSLT. Il s'agit d'une autre norme du W3C définissant un format XML qui permet de convertir un document en un autre. Le format XSLT est trop complexe pour être décrit ici, mais vous pouvez en trouver de nombreuses documentations accessibles sur Internet.

### Utilisation du module XSL

Sous PHP 4, nous avons le choix entre utiliser le moteur de Sablotron ou le moteur de la `libxslt` pour faire les transformations. Le premier était relativement lent et consommateur de ressources, le deuxième était intégré à une implémentation DOM non standard et expérimentale. En raison de ces deux désavantages, il n'était pas fréquent que les moteurs XSLT soient intégrés à PHP.

Avec PHP 5 et son nouveau module DOM, l'intégration de la `libxslt` a été réécrite. C'est celle-ci que nous allons décrire, même si le module Sablotron reste encore accessible pour ceux qui le souhaitent. La `libxslt` est une des bibliothèques C très répandues pour gérer les transformations XSLT. Elle a été créée dans le cadre du projet Gnome et est basée sur la `libxml2`. Elle implémente un moteur XSLT parmi les plus performants et gère les extensions EXSLT.

Contrairement aux modules XML vus précédemment, le module de gestion XSLT n'est pas compilé par défaut avec PHP. Sous Unix, si vous compilez PHP vous-même, il vous faudra passer l'option `--with-xsl` au script de configuration. Sous Microsoft Windows, il vous faudra décommenter l'extension `php-xsl.dll` dans votre `php.ini`.

## Initialisation

Le moteur XSLT s'utilise via un objet propre, un peu comme Xpath. Il vous faut donc commencer par instancier un objet de la classe `xsltProcessor`.

```
$moteurXslt = new xsltProcessor() ;
```

## Chargement de la feuille de style

Une fois le moteur XSLT instancié, il faut lui fournir la feuille de style XSLT qui servira à la transformation. Cette définition se fait via la méthode `importStyleSheet()`, qui accepte un document DOM en argument. Il sera donc nécessaire de charger la feuille XSLT via DOM avant de l'envoyer au moteur de transformations.

```
$moteurXslt = new xsltProcessor() ;  
$style = new domDocument() ;  
$style->load('style.xml') ;  
$moteurXslt->importStylesheet($style) ;
```

Charger la feuille XSLT au préalable et non pas en même temps que le document XML à transformer vous permettra de faire du traitement par lots et d'initialiser une seule fois la feuille de style pour plusieurs transformations.

## Transformation

La transformation du document elle-même se fait via la méthode `transformToXml()`. Elle accepte un document DOM en argument et renvoie le XML produit.

```
$moteurXslt = new xsltProcessor() ;  
$style = new domDocument() ;  
$style->load('style.xml') ;  
$moteurXslt->importStylesheet($style) ;  
$source = new domDocument() ;  
$source->load('source.xml') ;  
echo $moteurXslt->transformToXml($source) ;
```

Il est toutefois possible d'écrire directement le résultat dans un fichier à l'aide de la méthode `transformToUri()` à la place de `transformToXml()`. Le chemin cible est alors à spécifier en second argument et peut utiliser les abstractions de flux de PHP (donc écrire dans un fichier compressé ou sur un serveur FTP).

```
$moteurXslt->transformToUri($source, 'resultat.xml') ;
```

Il existe aussi une méthode `transformToDoc()` transformant le document et renvoyant un objet DOM de type `DomDocument` que vous pourrez modifier ou relire par la suite.

```
$domDocument = $moteurXslt->transformToDoc($source) ;
```

## Sorties HTML

Les méthodes `transformToXml()` et `transformToUri()` savent automatiquement tenir compte de l'instruction `method="html"` ou `indent="yes"` des feuilles de style XSLT (reportez-vous à la documentation sur le format XSLT pour plus d'informations).

## Paramètres de transformation

Il est possible d'envoyer des paramètres au processeur avant la transformation et de les récupérer en XSLT pendant la transformation.

Vous pouvez définir un paramètre en utilisant la méthode `setParameter()`. Le second argument est le nom du paramètre et le troisième est sa valeur. Cette valeur doit être codée en UTF-8 et non en ISO-8859-1. Le premier argument est prévu pour être à terme un URI d'espace de noms, mais n'est pas utilisé pour l'instant ; transmettez donc une chaîne vide.

```
$moteurXslt = new xsltProcessor() ;
$style = new domDocument() ;
$style->load('style.xml') ;
$moteurXslt->importStylesheet($style) ;
$moteurXslt->setParameter('', 'PHP', '5' ) ;
$source = new domDocument() ;
$source->load('source.xml') ;
echo $moteurXslt->transformToXml($source) ;
```

Il est possible de relire ou de supprimer un paramètre défini, respectivement avec `setParameter()` et `removeParameter()`.

```
$moteurXslt->setParameter('', 'PHP', '5' ) ;
echo $moteurXslt->getParameter('', 'PHP') ; // Affiche 5
$moteurXslt->removeParameter('', 'PHP') ;
```

## Extensions et interactions avec PHP

### Extensions EXSLT

La `libxslt`, sur laquelle est basé le module XSL de PHP, sait utiliser les extensions EXSLT. Vous trouverez par exemple parmi ces extensions des fonctions de calcul de date, quelques opérations numériques, des expressions régulières ou des fonctions avancées de traitement de chaînes. Vous en trouverez une description à l'adresse <http://www.exslt.org/>.

### Utilisations de fonctions PHP

Bien que cela ne soit pas à considérer comme une bonne solution du point de vue théorique, il est possible d'utiliser des fonctions PHP dans votre feuille XSLT.

Cette fonctionnalité n'est pas activée par défaut, pour vous permettre d'intégrer des feuilles de transformation externes en toute sécurité (sans que l'auteur puisse y exécuter

du code PHP). Pour l'activer, vous devez faire appel à la méthode `registerPHPFunctions()` de l'objet représentant le moteur XSLT.

```
$moteurXslt = new xsltProcessor() ;  
$moteurXslt->importStylesheet($style) ;  
$moteurXslt->registerPHPFunctions() ;
```

Vous pouvez alors utiliser les fonctions Xpath `php:function()` et `php:functionString()`, où `php` est l'espace de noms <http://php.net/xsl>. Le premier argument est le nom de la fonction à utiliser ; viennent ensuite les paramètres à donner à la fonction PHP. `php:function()` prend en paramètre et retourne des objets DOM. `php:functionString()` utilise uniquement des chaînes de caractères, que ce soit en entrée ou en sortie.

Voici un exemple de code XSLT utilisant la fonction `date()` :

```
<xsl:stylesheet  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  xmlns:php="http://php.net/xsl" version='1.0'>  
<xsl:template match="/">  
  
  <xsl:value-of select="php:function('date', 'r')"/>  
  
</xsl:template>  
</xsl:stylesheet>
```

Une des possibilités pourrait être l'utilisation de `gettext` dans les feuilles XSLT, pour permettre une internationalisation des textes à moindre coût.

Vous pouvez utiliser ainsi des fonctions natives de PHP, mais aussi des fonctions utilisateur. N'oubliez cependant pas de renvoyer vos textes avec le bon codage caractère.

Il est possible de restreindre l'utilisation de cette fonctionnalité à un jeu réduit de fonctions PHP. Dans ce cas on peut transmettre la liste des noms de fonctions autorisées dans un tableau en paramètre à `registerPHPFunctions()`.

### Abstractions de flux

Le module XSL est entièrement intégré à PHP. Tous les chemins de fichiers ou URI donnés passent automatiquement par les fonctions de gestion de flux PHP. Vous pouvez donc utiliser de manière transparente des fichiers compressés ou sur FTP.

La fonction Xpath `document()`, éventuellement présente dans les feuilles XSLT, utilise aussi ces fonctions d'abstraction.



## Les services web

---

L'objectif des services web est de permettre à votre application de faire appel à des fonctions distantes, appartenant à une application tierce, ailleurs sur le réseau. Ainsi, votre application peut faire une recherche sur le moteur Google, demander la liste des nouveautés sur Amazon ou récupérer les cours de la bourse en temps réel. On dit alors que l'on « consomme » un service web.

Inversement, votre propre application peut offrir ses propres services sur le réseau pour permettre à n'importe qui de s'interfacer avec les données que vous souhaitez partager.

La version 5 de PHP arrive avec une nouvelle extension native pour gérer le format le plus courant de ces services web : SOAP.

### Introduction aux services web

#### *Protocoles et technologies*

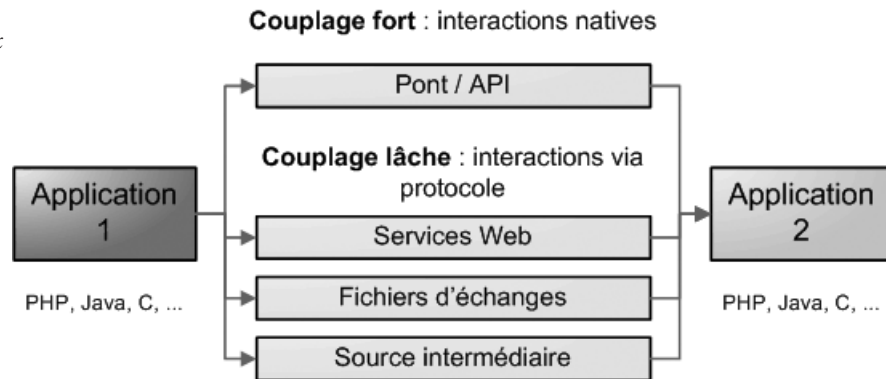
Les services web font entrer en jeu de nombreux protocoles et technologies. Dans un premier temps, nous allons expliquer le concept du service web, en opposition aux méthodes traditionnelles, puis nous aborderons ensuite les différentes approches possibles.

#### **Couplage lâche, couplage fort**

Pour communiquer entre deux applications, il existe deux types de couplage :

- couplage fort : interactions natives ;
- couplage lâche : interactions via des protocoles ou formats d'échanges.

Figure 22-1  
Couplage entre deux applications



Dans le cas du couplage fort, nous pouvons citer la JSR223 qui permet d'instancier des classes Java directement dans PHP. Le couplage fort engendre plus de dépendance, mais implique généralement des temps de réponse plus rapides.

Le couplage lâche consiste à utiliser un moyen (protocole) d'échange des données (en passant par un fichier texte, par un flux XML, par une base de données, par des services web).

### Protocoles d'invocations de services à distance

Les services web ont le vent en poupe, mais le concept d'invocation de services à distance n'est pas nouveau. Nous pouvons par exemple citer RMI (Remote Method Invocation) et Corba (Common Object Request Broker Architecture).

Par rapport à ces méthodes, la différence des services web vient du fait qu'ils se basent sur des formats et des protocoles standardisés et massivement utilisés : HTTP et XML. Pour travailler, nous pouvons utiliser différentes approches :

- XML-RPC (Remote Procedure Call) ;
- SOAP ;
- REST.

Parmi ces méthodes, nous avons d'un côté SOAP et XML-RPC, et de l'autre REST. Les deux premiers sont basés sur des technologies XML qui utilisent généralement le protocole HTTP, mais qui peuvent se servir aussi d'autres transports. REST, à l'inverse, est le modèle natif de HTTP. Nous pouvons utiliser des données XML sur REST, mais nous n'y sommes pas limités.

#### Le protocole HTTP offre une plus grande ouverture

L'utilisation de HTTP permet aussi aux applications d'utiliser les services web sans être bloquées par les pare-feu et proxies des entreprises. C'est une des raisons pour lesquelles on préfère les services web à Corba et à RMI.

## REST

REST (REpresentational State Transfer) permet de construire une application pour les systèmes distribués comme le Web. Ce n'est pas un protocole ou un format, mais une architecture (celle de HTTP). On identifie alors :

- une URI (Uniform Resource Identifier) qui permet d'identifier la ressource à laquelle on tente d'accéder ;
- une méthode HTTP (par exemple GET et POST), pour savoir quelle opération on souhaite effectuer sur la ressource ;
- des en-têtes HTTP, pour gérer les métadonnées et les informations sur le transport ;
- un corps de requête, les données réellement transmises.

Cette architecture est de plus en plus utilisée pour la réalisation de services web destinés à la communication entre machines. Le gros avantage de REST est sa simplicité. Il s'agit en effet uniquement de l'utilisation du protocole HTTP à son plein potentiel.

## XML-RPC

XML-RPC est un protocole RPC (Remote Procedure Call), une spécification simple et un ensemble de codes qui permettent à des processus s'exécutant dans des environnements différents de faire des appels de méthodes à travers un réseau.

Les processus d'invocation à distance utilisent le protocole HTTP pour le transport des données et la norme XML pour leur codage. Il est toutefois possible de faire du XML-RPC sur un autre protocole que HTTP.

XML-RPC est conçu pour permettre à des structures de données complexes d'être transmises, exécutées et renvoyées très facilement.

XML-RPC est l'ancêtre de SOAP.

## SOAP

Simple Object Access Protocol (SOAP) est un protocole de RPC orienté objet bâti sur XML.

Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur une autre machine.

Il s'agit du type de service web le plus courant. Il bénéficie de plus d'une spécification complète et détaillée dans les normes éditées par le W3C.

SOAP est très semblable à XML-RPC. Dans cet ouvrage nous utiliserons SOAP qui dispose avec PHP 5 d'une API native.



## Les différents protocoles en jeu

L'utilisation de SOAP implique plusieurs protocoles en fonction des étapes comme le montre la figure 22-2.

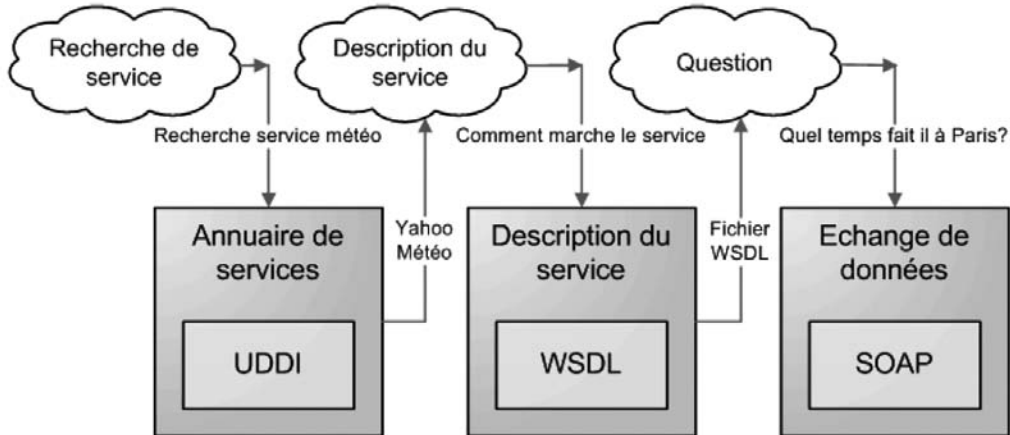


Figure 22-2

*Les différents protocoles en jeu*

### Annuaire de services

Pour trouver un service, on peut utiliser un annuaire reposant sur la norme UDDI (<http://www.uddi.org/>).

UDDI (Universal Description, Discovery and Integration) est un standard de plate-forme interopérable qui permet aux utilisateurs et aux applications de trouver et d'utiliser rapidement, facilement et dynamiquement des services web au travers d'Internet.

Nous ne détaillerons pas cette partie qui est relativement peu utilisée. En général, les informations sur le fournisseur de services web sont publiées dans la documentation, et donc connues.

### Description du service

La description du service se fait au travers d'une description de l'API distante. Les services sont décrits dans un fichier WSDL (Web Services Description Language). Nous reviendrons en détail sur ce protocole.

### Échanges entre applications

Simple Object Access Protocol (SOAP) est un protocole d'appels distants (RPC) orienté objet bâti sur XML. Il permet la transmission de messages entre objets distants, ce qui

veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur une autre machine.

Le transfert se fait le plus souvent à l'aide du protocole HTTP, mais peut également se faire par un autre protocole, comme SMTP.

## ***Principe d'un appel à un service***

Nous allons détailler ici le fonctionnement théorique d'un service web. PHP vous permet de faire abstraction de ces données, donc si vous préférez commencer par une approche plus pratique, passez directement à la partie mise en œuvre.

Les aperçus de XML et WSDL vous permettront de déboguer vos échanges SOAP en cas de problèmes.

### **Principe de fonctionnement**

L'appel d'un service web se fait en employant quatre composantes :

- un protocole de transport (généralement HTTP) ;
- une adresse (généralement l'URL) ;
- un nom de méthode (qui sera exécutée) ;
- une liste d'arguments.

Pour exemple, on pourrait s'adresser à un site d'horloge parlante pour lui demander l'heure exacte en mode 24 heures.

On envoie alors une requête HTTP à l'adresse `http://horloge-parlante/service.soap` en demandant la méthode « `heureExacte` », avec un seul paramètre de nom « `mode` », et de valeur « `24h` ».

### **Le message XML**

Le message destiné à notre horloge parlante est transmis en langage XML. Il comporte une enveloppe, qui elle-même peut contenir des en-têtes et un corps.

#### **PHP gère tout**

PHP nous permettra, dans notre utilisation courante, de faire abstraction des détails de ce message. Ce qui suit est donc là plus à titre d'illustration, et pour vous aider à comprendre les mécanismes en jeu. Vous n'avez pas besoin de comprendre tous les détails des messages XML suivants pour utiliser les services web avec PHP.

Le corps du message est composé d'une balise de même nom que la méthode à appeler et de balises internes pour les paramètres. Chaque paramètre peut être d'un type simple (entier, chaîne de caractères), ou d'un type complexe (lui-même composé avec des types simples et des types complexes), avec à chaque fois un nom et une valeur.

Voici un exemple de message SOAP qui pourrait avoir été envoyé à notre horloge parlante :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol env:mustUnderstand="false"
      xmlns:n="http://example.org/horlogeControl">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:heureExacte xmlns:m="http://example.org/horloge">
      <m:mode>24h</m:mode>
    </m:heureExacte>
  </env:Body>
</env:Envelope>
```

**Note**

Il existe plusieurs formes d'appel. Nous documentons ici un appel de type RPC/literal, le type le plus courant et normalisé. La question de l'interopérabilité et des différents types d'appel est présentée plus loin dans ce chapitre.

L'espace de nom « env » est celui de SOAP (son URI est dans la balise racine). Les balises <Envelope>, <Body> et optionnellement <Header> constituent l'enveloppe du message SOAP, c'est-à-dire sa structure.

À l'intérieur des en-têtes (<Header>), on trouve des paramètres sur le message lui-même (sa priorité, sa date, etc.).

Dans le corps (<Body>), on trouve une balise <heureExacte> qui permet de nommer le service demandé. Généralement, on définit un espace de nom pour cette balise et ses paramètres, afin de leur attribuer un sens et de pouvoir les mixer avec d'autres dialectes XML.

Par défaut, les services ont tendance à ignorer les en-têtes qu'ils ne connaissent pas. On peut remarquer ici la présence d'un paramètre `mustUnderstand` qui appartient à l'espace de nom SOAP. Il impose au service de savoir traiter la balise d'en-tête pour traiter le message lui-même.

Il existe quelques autres attributs, mais nous n'irons pas plus loin dans la description du message XML. PHP va nous permettre de nous abstraire de toute cette complexité et d'oublier les détails d'implémentation.

**Note**

Les messages d'erreur ou de réponse sont faits suivant le même format. Ils comportent une enveloppe avec des en-têtes et un corps, et dans le corps on trouve une balise d'erreur spécifique.

## Description d'un service web avec WSDL

Le point le plus délicat d'un service web est de savoir quelles fonctionnalités il offre. Pour qu'un client SOAP adresse sa requête, il faut qu'il connaisse la structure des méthodes proposées par le service : la liste des paramètres et leur type.

Une description du service doit alors être faite pour permettre au client d'envoyer les bonnes données sous la bonne forme, voire de découvrir automatiquement les différentes méthodes offertes par un service. Cette déclaration est réalisée par celui qui offre le service. Elle est faite dans un fichier XML nommé WSDL.

### Note

Un exemple de fichier WSDL est donné à titre d'illustration pour permettre de comprendre l'organisation interne des déclarations. La compréhension globale de l'exemple ci-dessous n'est pas nécessaire pour utiliser les services web en PHP.

Vous pouvez trouver des exemples de fichiers WSDL sur l'annuaire des services web de xmethods à l'adresse : <http://www.xmethods.net/>. Nous illustrerons certains exemples avec le service web de l'étang de Berre, qui permet de récupérer des informations météo sur certaines localités. Son adresse est :

■ <http://www.severalways.org/WS/BerreWeather/BerreWeather.php?wsdl>

Le WSDL est assez complexe. On y retrouve en particulier un espace de nom spécifique à WSDL, l'espace de nom SOAP, l'espace de nom des messages et l'espace de nom XML Schema (qui permet de spécifier des types normalisés). La balise racine est `<definitions>`. Elle comporte cinq sections importantes :

- La section `<types>` définit le contenu des types complexes. On a ici un type `WeatherReport` qui définit les informations propres à une information météo et un type `ForecastReport`, qui définit les informations pour une prévision.

```
<definitions targetNamespace="urn:BerreWeather">
  <types>
    <xsd:schema targetNamespace="urn:BerreWeather">
      <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
      <xsd:complexType name="WeatherReport">
        <xsd:all>
          <xsd:element name="UTC" type="xsd:string"/>
          <xsd:element name="Local" type="xsd:string"/>
          <xsd:element name="Barometer" type="xsd:string"/>
          <xsd:element name="Temperature" type="xsd:string"/>
          ...
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="ForecastReport">
        <xsd:all>
          <xsd:element name="UTC" type="xsd:string"/>

```

```

    <xsd:element name="Local" type="xsd:string"/>
    <xsd:element name="Valid" type="xsd:string"/>
    <xsd:element name="Wind" type="xsd:string"/>
    <xsd:element name="Direction" type="xsd:string"/>
    ...
  </xsd:all>
</xsd:complexType>
</xsd:schema>
</types>

```

- Les sections de `<message>` définissent les listes de messages qui peuvent être échangés avec le service. On y retrouve les différents types de requêtes ainsi que les différentes réponses. Chaque message est détaillé pour lister ses composants (nom et types des paramètres, avec soit des types XML Schema simples, soit des types qui font référence à une déclaration dans la section `<types>`).

```

<message name="GetWeatherRequest">
  <part name="City" type="xsd:string"/>
</message>

<message name="GetWeatherResponse">
  <part name="Weather" type="tns:WeatherReport"/>
</message>

<message name="GetForecastRequest">
  <part name="City" type="xsd:string"/>
</message>

<message name="GetForecastResponse">
  <part name="Forecast" type="tns:ForecastReport"/>
</message>

```

- La section `<portType>` est la plus importante. C'est elle qui va lister les messages autorisés en entrée et en sortie pour chaque opération accessible sur votre service web.

```

<portType name="BerreWeatherPortType">

  <operation name="GetWeather">
    <input message="tns:GetWeatherRequest"/>
    <output message="tns:GetWeatherResponse"/>
  </operation>

  <operation name="GetForecast">
    <input message="tns:GetForecastRequest"/>
    <output message="tns:GetForecastResponse"/>
  </operation>
</portType>

```

- Enfin, les sections `<binding>` et `<service>` vont gérer les détails d'implémentation (protocoles ou couches transport à utiliser, adresse, port, etc.).

```

<binding name="BerreWeatherBinding" type="tns:BerreWeatherPortType">

```

```
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetWeather">
    <soap:operation soapAction="urn:BerreWeather#GetWeather" style="rpc"/>
    <input>
      <soap:body use="encoded" namespace="urn:BerreWeather"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="urn:BerreWeather"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
  ...
</binding>
<service name="BerreWeather">
  <port name="BerreWeatherPort" binding="tns:BerreWeatherBinding">
    <soap:address location
      =>"http://www.severalways.org/WS/BerreWeather/BerreWeather.php" />
  </port>
</service>
</definitions>
```

Grâce à la déclaration WSDL des différents services, un logiciel peut proposer une interface utilisateur dans laquelle on choisit l'appel à faire dans une liste (avec une aide qui donne la description de chaque appel), puis proposer un formulaire pour saisir les paramètres. Chaque paramètre voit alors sa valeur vérifiée par le logiciel suivant les spécifications du WSDL.

Ce WSDL peut aussi servir à un framework ou à un langage de programmation pour transformer automatiquement les messages SOAP en données natives et vice-versa. C'est justement ce que va nous proposer PHP. Grâce au WDSL, PHP va automatiquement transformer les données SOAP en chaînes de caractères, entiers et tableaux, ou l'inverse.

On peut donc voir la déclaration WSDL comme un contrat entre le client qui consomme le service web et celui qui l'offre. On déclare ce qui est échangé, où, et comment. C'est sur ces spécifications que les deux intervenants vont pouvoir se baser pour interpréter les échanges.

## Utilisation simple (avec WSDL)

Une fois que vous avez accès à un fichier WSDL, PHP pourra le lire et l'interpréter. Il connaîtra alors la liste des méthodes qu'il peut exécuter, et les paramètres à fournir.

Avant PHP 5, utiliser des services SOAP en PHP nécessitait l'utilisation d'une bibliothèque externe (généralement nuSOAP). Depuis PHP 5, une extension native nommée « SOAP » est disponible. Elle permet de bien meilleures performances et profite des toutes dernières fonctionnalités objet de PHP. Il vous faudra la compiler ou l'activer dans la configuration PHP (fichier `php.ini`).

## Créer un client SOAP

Les déclarations WSDL permettent à PHP d'automatiser complètement l'utilisation d'un service web. Vous ne manipulerez donc ni XML ni SOAP directement.

C'est PHP qui, en interne, va faire des échanges SOAP avec le serveur distant à chaque fois que vous exécutez une méthode. La seule étape spécifique est d'initialiser votre objet en lui donnant l'adresse du fichier de déclaration WSDL.

```
<?php
$wsdl =
"http://www.severalways.org/WS/BerreWeather/BerreWeather.php?wsdl" ;
$etang = new SoapClient( $wsdl );
```

En reprenant le service de l'étang de Berre cité plus haut, nous appellerons la méthode « GetWeather » pour récupérer le bulletin météo actuel.

On commence par créer un objet de type SoapClient en donnant l'adresse de la déclaration WSDL. Une fois cet objet créé, l'action « GetWeather » sera visible comme une méthode locale de notre objet :

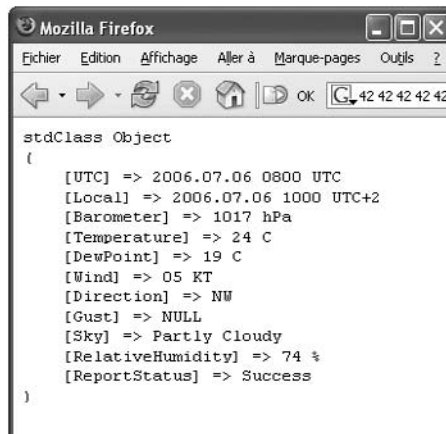
```
<?php
$wsdl =
"http://www.severalways.org/WS/BerreWeather/BerreWeather.php?wsdl"
;
$etang = new SoapClient( $wsdl );
$bulletin_meteo = $etang->GetWeather("Marignane");
echo "La temperature est de ".$bulletin_meteo->Temperature ;
```

La variable `$etang` est un objet qui représente la liaison avec le service web. Chaque méthode appelable sur le service est représentée par une méthode sur l'objet PHP. On manipule cet objet comme si c'était un objet classique en PHP.

Ici, la méthode `GetWeather` prend en paramètre un nom de localité et retourne un objet (type complexe dans la déclaration WSDL) représentant le bulletin météo avec un attribut « Temperature ».

Figure 22-3

*Valeurs retournées  
par la méthode  
GetWeather*



## Utiliser des flux compressés

Les messages SOAP sont des messages XML, donc assez importants en taille mais qui se compressent très facilement.

Quand ces messages sont transmis par HTTP, il est possible d'utiliser un flux compressé si le client et le serveur le supportent. Pour déclencher ce fonctionnement, vous pouvez passer une valeur « true » dans un tableau d'options à l'index « compression » :

```
<?php
$wsdl = "BerreWeather.wsdl";
$params = array('compression' => true);

$etang = new SoapClient( $wsdl, $params);

$bulletin_meteo = $etang->GetWeather("Marignane");
echo "La temperature est de ".$bulletin_meteo->Temperature ;
```

## Gérer les authentifications HTTP et Proxy

De même, les options login et password permettent de gérer une éventuelle authentification HTTP pour utiliser le service web.

```
// NOTE : le service de l'étang de Berre ne demande
// pas d'authentification, c'est juste pour illustration
$wsdl = "BerreWeather.wsdl";
$params = array('login' => 'eric', 'password'=>"secret");

$etang = new SoapClient( $wsdl, $params);
$bulletin_meteo = $etang->GetWeather("Marignane");
echo "La temperature est de ".$bulletin_meteo->Temperature ;
```

Les options proxy\_host, proxy\_port, proxy\_user et proxy\_password permettent d'envoyer la requête SOAP à travers un proxy si le serveur n'a pas un accès direct au réseau de destination (l'utilisateur et le mot de passe sont facultatifs).

## Créer un serveur SOAP

### Créer un fichier WSDL

Pour créer un fichier WSDL, vous pouvez vous référer aux spécifications et l'écrire à la main, ou utiliser un outil permettant de le générer.

La création d'un fichier de déclaration WSDL sort du cadre de ce livre à cause de sa complexité et des nombreuses formes possibles. Toutefois, deux outils se démarquent



pour vous faciliter cette tâche. Ils vous donneront des interfaces pour générer automatiquement une déclaration WSDL standard :

- DIA + UML2PHP5 (<http://uml2php5.zpmag.com>) ;
- ZendStudio (<http://www.zend.com>).

Pour UML2PHP5, décompressez complètement l'archive téléchargée dans le répertoire `xs1t` de Dia. Attention, si vous voulez générer des fichiers WSDL, il faut aussi copier le contenu du sous-répertoire `TOOLS` présent dans l'archive.

Ensuite lancez Dia, faites votre schéma UML (sans oublier de prototyper votre classe avec SOAP).

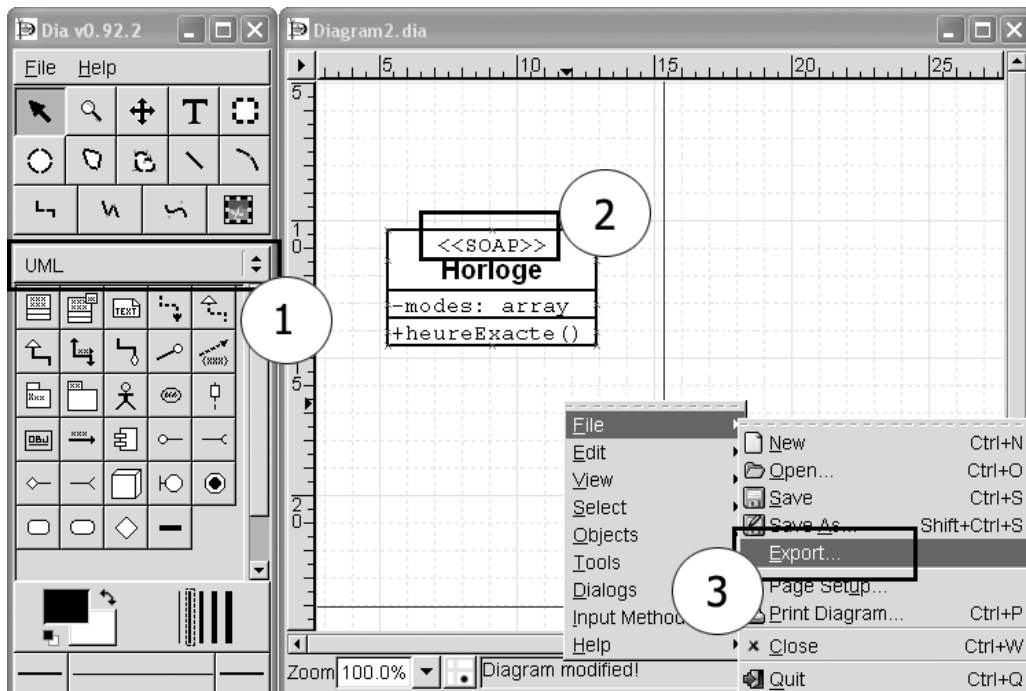


Figure 22-4

Export en WSDL d'un diagramme DIA

Enregistrez le au format `.dia`, exportez en sélectionnant le filtre XSL `*.code`. Sélectionnez « UML-CLASSES-EXTENDED » dans la partie supérieure de la boîte de dialogue qui apparaît et « PHP5/WSDL/SOAP Webservices » dans la partie inférieure. Validez et le tour est joué !

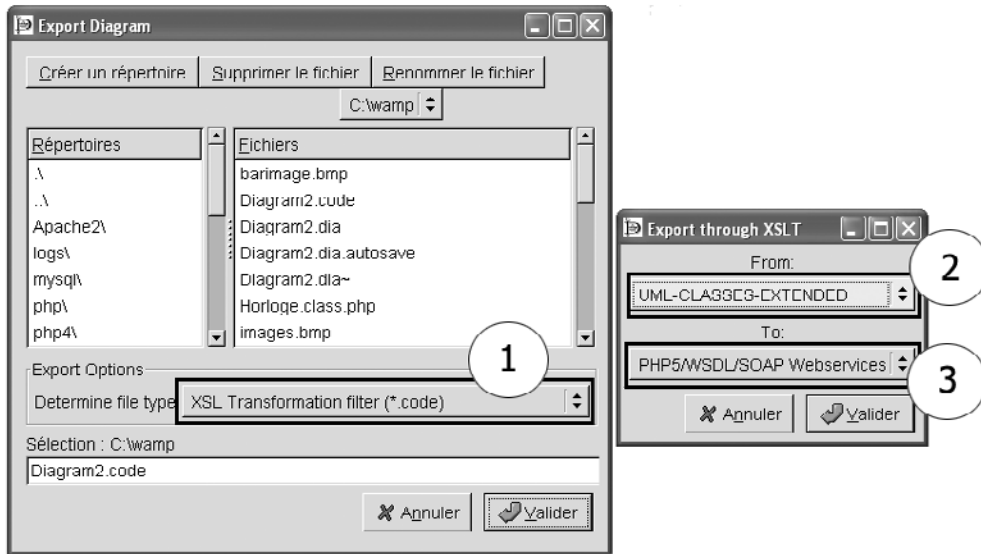


Figure 22-5

Enregistrement du fichier WSDL avec UML2PHP5

### Mettre en place un serveur SOAP

Une fois le fichier WSDL généré, développer un serveur SOAP n'est pas beaucoup plus complexe que d'y faire appel en tant que client. Il s'agit de définir une suite de fonctions tout à fait classiques qui seront « offertes » dans le service web. PHP s'occupe du reste : gestion des messages XML, des erreurs, conversion entre les données PHP et les types XML, etc.

Pour construire un serveur SOAP il nous suffit :

- de déclarer une fonction PHP classique ;
- d'instancier un serveur SOAP ;
- d'ajouter la fonction suivante.

```
<?php
$wsdl = "weather.wsdl" ;
function GetWeather($emplacement) {
    $o = new StdClass ;
    $o->UTC = "2006.07.05 1500 UTC" ;
    $o->Local = "2006.07.05 1700 UTC+2";
    $o->Barometer = "1015 hPa";
    $o->Temperature = "33 C";
    $o->DewPoint = "15 C";
    $o->Wind = "18 KT";
    $o->Direction = "S";
```

```
$o->Gust = "NULL";
$o->Sky = "Sunny";
$o->RelativeHumidity = "34 %";
$o->ReportStatus = "Success";
return $o ;
}

$server = new SoapServer($wsdl);
$server->addFunction("GetWeather");
$server->handle();
```

Pour notre exemple, nous avons téléchargé la déclaration WSDL de l'étang de Berre, et nous y avons modifié l'adresse du service web pour y mettre à la place l'adresse de notre script serveur. L'URL à remplacer se trouve tout à la fin dans la balise <service>. Si notre script serveur SOAP tourne sur le poste local (127.0.0.1), dans le fichier `serveur.php` on obtient la balise XML suivante :

```
<soap:address location="http://127.0.0.1/serveur.php"/>
```

La dernière ligne de notre script serveur fait un appel à `handle()`. PHP prend alors la main, examine la requête SOAP en entrée, appelle votre fonction avec les bons paramètres et retourne la réponse en XML au client. Vous n'avez rien d'autre à faire.

Ici, nous retournons un objet standard (`StdClass`) dont les attributs représentent les propriétés d'un bulletin météo, telles qu'elles sont définies dans le fichier WSDL.

Plutôt que de déclarer plusieurs fonctions une à une dans votre serveur SOAP, vous pouvez aussi créer une classe dédiée et demander au serveur d'utiliser directement tout ce qui est accessible dans votre classe :

```
<?php

$wsdl = "weather.wsdl" ;

class Etang {
function GetWeather($emplacement) {
    $o = new StdClass ;
    $o->UTC = "2006.07.05 1500 UTC" ;
    $o->Local = "2006.07.05 1700 UTC+2";
    $o->Barometer = "1015 hPa";
    $o->Temperature = "33 C";
    $o->DewPoint = "15 C";
    $o->Wind = "18 KT";
    $o->Direction = "S";
    $o->Gust = "NULL";
    $o->Sky = "Sunny";
    $o->RelativeHumidity = "34 %";
    $o->ReportStatus = "Success";
    return $o ;
}
function GetForecast($emplacement) {
    $o = new StdClass ;
```

```
$o->UTC = "2006.07.05 1400 UTC";
$o->Local = "2006.07.05 1600 UTC+2";
$o->Valid = "2006.07.05 1500 UTC TO 2006.07.05 2400 UTC";
$o->Wind = "20 KT";
$o->Direction = "SSE";
$o->Gust = "NULL";
$o->Sky = "Sunny";
$o->ReportStatus = "Success";
return $o ;
}
}

$server = new SoapServer($wsdl);
$server->setClass("Etang");
$server->handle();
```

**Note**

L'utilisation des erreurs et la signification de la classe `SoapFault` seront vus plus loin dans ce chapitre.

## Persistence

Par défaut, comme toujours en HTTP, chaque requête est indépendante des autres. PHP vous permet tout de même de gérer une persistance dans vos services web avec le principe des sessions. Si vous utilisez une classe pour gérer votre service, PHP met alors votre instance dans une session dès qu'il a traité une requête. Si le même client revient rapidement faire une requête, PHP va récupérer l'instance sauvegardée en session pour la restaurer. Vous pouvez alors stocker des informations persistantes dans l'instance de votre classe.

On déclenche le mode persistant avec un appel à la méthode `setPersistence()`. Cet appel doit être fait avant l'appel à `handle()` et sans que la session soit démarrée (donc ni appel à `session_start()` ni utilisation de la directive de `php.ini` `session.auto_start`). La méthode `setPersistence()` prend un paramètre qui sera soit la constante `SOAP_PERSISTENCE_SESSION` pour l'utilisation de la session, soit `SOAP_PERSISTENCE_REQUEST` pour le fonctionnement par défaut.

```
$server = new SoapServer("horloge.wsdl");
$server->setClass("Horloge");
$server->setPersistence(SOAP_PERSISTENCE_SESSION);
$server->handle();
```

**Note**

Pour que la persistance fonctionne, il faut que le client du service web sache gérer les cookies et les renvoie lors de ses futures requêtes. De même, la session n'est utilisée que si elle n'a pas expiré, c'est-à-dire que les différentes requêtes sont suffisamment proches dans le temps.

## Cache WSDL

Les fichiers WSDL sont fréquemment disponibles sur le web par HTTP. Les télécharger à chaque utilisation d'un service web peut se révéler très mauvais pour les performances de votre application. La solution la plus simple est de télécharger manuellement la déclaration WSDL et la stocker sur le disque à côté de vos fichiers PHP pour une utilisation locale. Cette solution a le désavantage de vous figer et vous empêche de profiter des mises à jour du WSDL. Cela risque de poser des problèmes de compatibilité si un changement apparaît dans le WSDL d'origine.

Pour vous éviter cette peine, PHP gère lui-même un cache des fichiers WSDL pour réduire les téléchargements. Ce cache est activable et configurable à l'aide de trois directives dans le fichier de configuration `php.ini`.

La directive `soap.wsdl_cache_enabled` permet d'utiliser le cache quand elle est à 1. Quand cette directive est activée, PHP stocke les fichiers WSDL après les avoir téléchargés dans le répertoire pointé par la directive `soap.wsdl_cache_dir`. Ces fichiers seront ensuite réutilisés en local sans être téléchargés. Ils seront de nouveau téléchargés après un nombre de secondes défini dans la directive `soap.wsdl_cache_ttl`.

```
soap.wsdl_cache_enabled = "1"
soap.wsdl_cache_dir = "/tmp"
soap.wsdl_cache_ttl = 86400
```

### soap

<b>Soap Client</b>	enabled
<b>Soap Server</b>	enabled

Directive	Local Value	Master Value
<b>soap.wsdl_cache_dir</b>	c:\wamp/tmp	c:\wamp/tmp
<b>soap.wsdl_cache_enabled</b>	1	1
<b>soap.wsdl_cache_ttl</b>	86400	86400

Figure 22-6

*Options de configuration pour SOAP*

#### Note pour les développeurs

Le cache WSDL est activé par défaut avec un temps d'expiration réaliste. Vous devriez laisser ces valeurs telles quelles pour un serveur en production. Toutefois, le cache WSDL peut poser de sérieuses difficultés pendant les développements si votre service web n'est pas stabilisé. Vous pourrez alors avoir besoin de désactiver le cache pendant la phase de conception et de test pour que les changements du WSDL soient visibles sans attente.

## Utiliser SOAP sans WSDL

Si aucune déclaration WSDL n'est disponible, PHP ne pourra pas automatiquement gérer les noms des paramètres et les noms des méthodes. Il faut renseigner à la main les noms des paramètres, les noms des méthodes à appeler, les URI d'espace de nom, etc.

Bien que la création d'un fichier WSDL soit hors du cadre de ce livre, nous vous conseillons très fortement de toujours créer une telle déclaration pour accompagner vos services web. Comme vous l'avez vu, l'utilisation d'un service avec WSDL est très simple, que ce soit en tant que client ou en tant que serveur. Le temps passé à créer votre déclaration WSDL sera largement amorti par la simplicité du code qui en résultera. Pensez aussi que l'absence d'une telle déclaration risquera de décourager les gens d'utiliser votre service.

### Créer un client SOAP sans WSDL

La procédure d'instanciation du service est similaire à ce que nous avons avec un fichier WSDL. Au lieu de fournir l'adresse du fichier WSDL en premier paramètre, nous fournissons la valeur NULL et nous indiquons les détails de paramétrage dans un tableau d'options en second paramètre :

```
<?php
$options = array('location' => "http://127.0.0.1/serveur.php",
                'uri' => "urn:BerreWeather",
                "style" => SOAP_RPC ,
                "use" => SOAP_ENCODED
                );
$etang = new SoapClient( NULL, $options );
```

Comme vous pouvez le constater, l'appel est plus complexe. La plupart des paramètres (uri, location, style, use, soapaction) sont en effet normalement déclarés dans le WSDL et il faut ici les spécifier explicitement :

- location est l'adresse web où faire la requête ;
- uri est l'espace de nom XML à utiliser (même si le préfixe peut être http://, l'adresse n'est pas forcément accessible dans un navigateur) ;
- style et use sont des paramètres qui vous sont donnés par celui qui offre le service, nous y reviendrons en fin de chapitre.

L'appel d'une méthode est lui aussi plus lourd. Il nous faut passer par `__soapCall()` et lui donner le nom de ce qu'on veut appeler sur le serveur et un tableau avec tous les paramètres :

```
$methode= "GetWeather" ;
$params = new SoapParam("City", "Marignane") ;
$params = array( $params ) ;
$options = array(
    "soapaction" =>
        "urn:BerreWeather#GetWeather" ,
);
```

```

$bulletin_meteo = $etang->__soapCall($methode,$params,$options);
echo "La temperature est de ".$bulletin_meteo->Temperature ;

```

La classe SoapParam permet à PHP de savoir sous quel nom envoyer notre paramètre. Il s'agit d'encapsuler notre valeur (premier argument du constructeur) et son nom (second argument). En mode WSDL, le nom est automatiquement détecté à partir de la définition du service.

Le paramètre soapaction est nécessaire quand il s'agit d'un service web proposé sur le protocole HTTP. La chaîne de caractères doit représenter « l'intention » du message, généralement on utilise le nom de la méthode appelée. On fournit soapaction dans un tableau d'options, en dernier paramètre de \_\_soapCall().

Il est toutefois possible de simplifier l'appel sans WSDL pour le faire un peu ressembler à l'appel avec WSDL :

```

class SoapClientNoWSDL {
    function __construct($params) {
        parent::__construct(null, $params) ;
    }
    function __call($name, $param) {
        $options = array("soapaction", $name) ;
        $this->__soapCall($name,$param,$options);
    }

$options = array('location' => "http://127.0.0.1/serveur.php",
                'uri'      => "urn:BerreWeather",
                "style" => SOAP_RPC ,
                "use" => SOAP_ENCODED
                ) ;
$etang = new SoapClientNoWSDL($options );
$params = array( $param1 );
$bulletin_meteo = $etang->heureExacte($params);
echo "La temperature est de ".$bulletin_meteo->Temperature ;

```

Il restera tout de même à passer à la main les options au constructeur et il faudra faire en sorte de pouvoir déterminer automatiquement le paramètre soapaction s'il est nécessaire au serveur en face (dans l'exemple ci-dessus, le paramètre soapaction généré n'est pas le bon). Les paramètres, eux, devront toujours être passés à l'aide de SoapParam.

## Serveur SOAP sans WSDL

La procédure pour créer un serveur est similaire. La seule option obligatoire est uri, qui définit l'espace de nom à utiliser dans la réponse :

```

<?php
class Etang {
    // Implémentation similaire à précédemment
}
$options = array(

```

```
'uri' => "urn:BerreWeather"  
);  
$server = new SoapServer(NULL, $options);  
$server->setClass("Etang");  
$server->handle();
```

PHP générera alors une réponse de type `<XxxResponse>` où Xxx est le nom de la méthode appelée. La valeur retournée sera dans une balise `<return>`.

#### Note

Le type de la réponse sera en revanche généré automatiquement. Ce peut donc être un nombre, une chaîne de caractères ou un tableau (exporté comme un type « struct »). Déclarer des types complexes n'a toutefois de sens que dans le cadre d'un échange avec WSDL, où les deux parties connaissent les types spéciaux utilisés.

## Gestion des types et des structures

Les types utilisables avec SOAP sont ceux de la norme XML Schema. On y trouve des entiers, des nombres à virgule flottante, des chaînes de caractères, des booléens, des énumérations et des types dits « complexes » qui sont en fait des compositions ou des listes d'éléments. C'est l'attribut `xsi:type` qui permet dans le message ou dans le WSDL de déclarer un type de donnée. Par exemple, le type entier est noté `xsi:type="xsd:int"`.

### Types spécifiques

Jusqu'à présent, nous avons laissé PHP traduire automatiquement nos données PHP (nombres, booléens, chaînes de caractères et objets) en types SOAP. PHP le fait tout seul de manière basique en traduisant ses propres types simples. PHP peut même utiliser quelques types complexes ou évolués s'il a un WSDL à disposition (pour savoir que le nombre est un `xsi:short` et pas un `xsi:int`).

Toutefois, si vous utilisez des services évolués, et principalement si vous le faites sans l'aide de WSDL, vous aurez aussi probablement à utiliser des types complexes que PHP aura du mal à convertir. Dans ce cas-là, vous devrez utiliser la classe `SoapVar`. Son constructeur prend deux paramètres obligatoires : la valeur PHP et une constante qui détermine le type à utiliser.

Au lieu de retourner ou d'utiliser directement une valeur PHP, vous pouvez retourner une instance de la classe `SoapVar`. Par exemple, pour retourner une date et non un entier :

```
$param = new SoapVar(time(), XSD_DATE);
```

La liste complète des types possibles est trop longue pour figurer dans ces pages, vous la trouverez sur la page principale de la documentation du module SOAP : <http://php.net/soap>. Voici tout de même les types les plus courants :

- `XSD_STRING` : chaîne de caractères ;



- XSD\_BOOLEAN : booléen (vrai ou faux) ;
- XSD\_DECIMAL, XSD\_FLOAT, XSD\_DOUBLE : nombres décimaux (exact, à virgule flottante et à double précision) ;
- XSD\_DATETIME, XSD\_TIME, XSD\_DATE : date complète, heure et jour ;
- XSD\_INTEGER, XSD\_INT : entier standard ;
- SOAP\_ENC\_OBJECT : structure complexe avec plusieurs composants, présentée en PHP sous forme d'objet ;
- SOAP\_ENC\_ARRAY : tableau de données.

Votre service peut toutefois demander des types non standards ou spécifiques. Dans ce cas, PHP peut traduire votre donnée dans un type standard mais déclarer dans le message le type que vous souhaitez. Il suffit de passer le nom du type dans un troisième paramètre et l'URI d'espace de nom du type dans un quatrième :

```
$param = new SoapVar(time(), XSD_DATE, "Dperso", "urn:perso");
```

Au lieu de générer un type `xsi:type="xsd:date"`, PHP convertira bien en date, mais générera un type `xsi:type="p:Dperso"` où `p` est le préfixe de `urn:perso`.

## Structures

XML est un format hiérarchique grâce auquel on peut regrouper des données entre elles, faire des groupes et des sous-groupes. Les services web utilisent fréquemment cette possibilité pour créer des structures : un élément contient plusieurs données référencées par un nom. On peut assimiler ces structures à des tableaux associatifs ou à des objets imbriqués.

C'est ce dernier choix qu'a fait PHP. Un objet avec un attribut « date » et un attribut « heure » sera traduit en un élément avec un sous-élément `<date>` et un sous-élément `<heure>`. Chaque attribut peut lui-même être un objet pour gérer des structures imbriquées complexes.

## Relation entre les classes et les types

Pour gérer plus simplement les types complexes, PHP peut utiliser des objets classiques et faire la conversion entre le nom de classe PHP et le nom de type SOAP. Pour être libre sur les noms de classe et pour qu'un service SOAP ne puisse pas utiliser n'importe quelle classe de votre application sans votre accord, c'est à vous de déclarer quelles classes utiliser via un tableau de correspondance. Ce tableau prend les noms de types WSDL en index et les noms de classe en valeur.

Ainsi, si vous avez un type complexe déclaré dans votre WSDL :

```
<complexType name="HorlogeInWSDL">
  <sequence>
    <element name="Heure" type="xsd:time"/>
    <element name="Date" type="xsd:date"/>
  </sequence>
</complexType>
```

Vous pouvez définir une classe dans votre PHP :

```
class HorlogeEnPHP {  
    public $Heure ;  
    public $Date ;  
}
```

Et la déclarer quand vous instanciez votre client ou votre serveur SOAP. Le dernier paramètre des deux constructeurs est un tableau d'options. Ici, c'est l'entrée `classmap` qui nous intéresse dans ces options :

```
$classmap = array('HorlogeInWSDL' => 'HorlogeEnPHP') ;  
$options = array('classmap'=>$classmap) ;  
$server = new SoapServer("horloge.wsdl", $options);
```

PHP traduira alors tout seul vos objets de classe `HorlogeEnPHP` dans des types complexes SOAP `HorlogeInWSDL`.

## Compatibilité .Net et formats

### *Différents formats de message*

Il existe en réalité plusieurs formes pour les messages SOAP. On peut jouer sur deux paramètres, le `use` et le `style` utilisés précédemment dans le client sans WSDL.

Le `use` peut prendre deux valeurs : `SOAP_LITERAL` et `SOAP_ENCODED`. L'exemple en introduction a été fait en mode littéral (mode par défaut). Avec `SOAP_ENCODED`, chaque élément XML d'une réponse ou d'une requête SOAP a un attribut explicite qui détermine son type (par exemple `xsi:type="xsi:int"` si l'élément contient un nombre entier). L'exemple de départ peut alors se reformuler ainsi :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">  
  <env:Header>  
    <n:alertcontrol env:mustUnderstand="false"  
      xmlns:n="http://example.org/horlogeControl">  
      <n:priority>1</n:priority>  
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>  
    </n:alertcontrol>  
  </env:Header>  
  <env:Body>  
    <m:heureExacte xmlns:m="http://example.org/horloge">  
      <m:mode xsi:type="xsd:string">24h</m:mode>  
    </m:heureExacte>  
  </env:Body>  
</env:Envelope>
```

Le `style` peut lui aussi prendre deux valeurs : `SOAP_DOCUMENT` et `SOAP_RPC`. En mode `RPC` (par défaut), les paramètres de requête ou de réponse sont encapsulés dans un élément unique.

Dans notre cas, c'était `<heureExacte>`. En mode `SOAP_DOCUMENT`, les réponses sont directement mises dans le corps de l'enveloppe SOAP (ici `<env:Body>`) :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol env:mustUnderstand="false"
      xmlns:n="http://example.org/horlogeControl">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:mode>24h</m:mode>
  </env:Body>
</env:Envelope>
```

## Compatibilité avec un service .Net

Le format de message est normalement déclaré dans le fichier WSDL. Si vous n'utilisez pas de déclaration WSDL, il faudra alors vous assurer que vous utilisez le même que celui qui est attendu par vos clients ou votre serveur.

Le format respectant la norme est normalement le `RPC/LITERAL`. On peut toutefois utiliser `DOCUMENT/LITERAL` si le corps de l'enveloppe ne contient qu'un seul élément (cet élément peut toutefois être une structure qui contient d'autres données). Le mode `RPC/ENCODED` est parfois utilisé pour un couplage lâche : les données sont envoyées sans vérification et les types utilisés sont déclarés dans le message. Le mode `DOCUMENT/ENCODED` n'est pas utilisé.

Les serveurs et clients .Net utilisent `DOCUMENT/LITERAL` en enveloppant les paramètres dans un élément unique. Le message ressemble alors à un message `RPC/LITERAL`, seuls le nom de l'élément parent et l'intérieur du WSDL changent. Le nom de l'élément parent sera `XxxRequest` pour une requête et `XxxResponse` pour une réponse, où `Xxx` est le nom de la méthode appelée. Si rien n'est défini spécifiquement et qu'il n'y a qu'une seule donnée en réponse, elle sera dans un sous-élément nommé `XxxResult`, où `Xxx` est toujours le nom de la méthode appelée. Ainsi, pour récupérer le résultat, on pourra faire :

```
$msresponse = $horlogeParlante->HeureExacte("24h");
echo $msresponse->HeureExacteResponse ;
```

## Autres détails et possibilités

### Codage caractères

PHP utilise un codage caractères UTF-8 pour l'émission de ses messages, et pour le traitement de ce qu'il reçoit. Vous devrez donc fournir et réceptionner des chaînes dans ce codage caractère.

Vous pouvez toutefois demander à PHP de faire des conversions pour que les textes de vos scripts soient dans un autre codage, par exemple ISO-8859-1 (ceux des messages envoyés seront toujours en UTF-8, PHP faisant la conversion automatiquement). Il faut alors fournir le nom du codage caractère souhaité dans l'index « encoding » :

```
$options = array('encoding'=>'ISO-8859-1') ;
$server = new SoapServer("horloge.wsdl", $options);
```

## Définir des en-têtes SOAP

Certains paramètres SOAP sont parfois gérés dans les en-têtes de l'enveloppe SOAP. On peut y définir des priorités ou des instructions de routage par exemple. Dans l'objet de client, vous pouvez envoyer un tableau d'en-têtes SOAP. Avant d'exécuter la méthode distante, il est aussi possible d'envoyer des en-têtes SOAP en quatrième paramètre à la méthode `__soapCall()`, ou via la méthode `__setSoapHeaders()` juste avant d'exécuter l'appel distant.

Chaque en-tête est une instance de la classe `SoapHeader`. Le constructeur prend en arguments un URI d'espace de nom, un nom de balise XML et une valeur. Optionnellement, il peut aussi prendre un booléen en quatrième position, qui définit si le serveur doit obligatoirement comprendre et traiter l'en-tête pour pouvoir traiter la requête.

```
$params_client = array(
    "location" => "http://example.org/horloge.soap",
    "uri" => "http://example.org/horloge" ,
    "style" => SOAP_RPC ,
    "use" => SOAP_LITERAL ,
);
$horlogeParlante = new SoapClient( NULL, $params_client );
$priorite = 1 ;
$uri = "http://example.org/horlogeControl" ;
$nom = "priority" ;

$headers = array( new SoapHeader($uri,$nom,$priorite) );

$nom = "heureExacte" ;
$params1 = new SoapParam("24h", "mode") ;
$params = array( $params1 ) ;
$options = array(
    "soapaction" => "http://example.org/horloge#heureExacte" ,
);
echo $horlogeParlante->__soapCall($nom,$params,$options,$headers);
```

## Utiliser un autre transport que HTTP

Vous pouvez demander à vos objets clients SOAP de transmettre vos messages sur autre chose qu'une connexion HTTP classique. Il est par exemple possible d'envoyer et recevoir des messages SOAP via la messagerie électronique. HTTP n'est en effet qu'une possibilité parmi d'autres, même si c'est la plus fréquente.

Du côté client, il faut créer une nouvelle classe qui va hériter de `SoapClient` et redéfinir la méthode `__doRequest()`. Quand PHP cherchera à faire un appel avec votre client, il utilisera cette méthode en interne avec quatre arguments :

- le message XML de requête ;
- l'adresse où envoyer la requête ;
- l'action à réaliser (paramètre `soapaction`) ;
- la version SOAP utilisée.

Vous pourrez alors envoyer le message XML par e-mail si vous le souhaitez.

Du côté serveur, vous pouvez manuellement fournir le message XML de requête (récupéré par e-mail) en argument à la méthode `handle()` plutôt que laisser PHP le récupérer tout seul par HTTP. Le message XML de réponse est normalement envoyé directement à l'affichage pour transmission au navigateur. Vous pouvez l'intercepter à l'aide des fonctions de gestion de tampon `ob_start()`, `ob_get_contents()` et `ob_end_clean()` :

```
$requete = /* récupération par email du message */ ;
$server = new SoapServer("horloge.wsdl") ;
ob_start() ;
$server->handle($requete) ;
$reponse = ob_get_contents() ;
ob_end_clean() ;
/* transmission par email de $reponse */
```

### Note

Vous n'avez pas besoin d'utiliser cette méthode pour gérer un service web via HTTP sécurisé (HTTPS). PHP gère en effet le HTTPS de manière transparente. En tant que client, il suffit de lui indiquer une adresse destination en `https`, et en tant que serveur, PHP délègue toute la gestion `https` au serveur web. Si toutefois vous souhaitez utiliser un certificat spécifique pour votre client, vous pouvez renseigner l'adresse de votre fichier de certificat `.pem` dans l'attribut `_local_cert` de votre objet client.

## Gestion des erreurs

Les erreurs SOAP se gèrent toutes via un objet spécifique `SoapFault`. Un objet `SoapFault` contient au moins deux attributs standards : `faultcode` et `faultstring`. Ils représentent le code et le message de l'erreur.

Deux autres attributs sont optionnellement présents : `faultactor` qui identifie l'acteur qui a causé l'erreur et `detail`, pour donner une description plus précise.

## Erreurs reçues par un client SOAP

Si vous manipulez un client SOAP et que le serveur vous retourne une erreur, PHP vous l'enverra par défaut sous forme d'exception avec un objet de la classe SoapFault.

```
<?php
try {
    $client = new SoapClient("fichier.wsdl");
    $result = $client->testMethodeAvecErreur();
} catch( SoapFault $e) {
    echo "L'erreur $e->faultcode ($e->faultstring) est survenue" ;
}
```

Il est toutefois possible de se passer des exceptions et retourner à un comportement plus classique en PHP. Il suffit pour cela de passer la valeur `false` dans le tableau d'options à l'index `exceptions`. Les erreurs sont alors retournées comme un résultat normal et la fonction `is_soap_fault()` permet de les repérer :

```
<?php
$options = array('exceptions'=>false);
$client = new SoapClient("fichier.wsdl", $options);
$result = $client->testMethodeAvecErreur();
if (is_soap_fault($result)) {
    echo "L'erreur $result->faultcode est survenue" ;
} else {
    echo "Le résultat est $result" ;
}
```

## Utilisation des traces

Pour le débogage, PHP nous permet de créer des traces sur les échanges SOAP. L'activation de ces traces se fait via l'option `trace` à l'instanciation du client. On obtient alors quatre méthodes sur l'objet client :

- `__getLastRequest()`
- `__getLastRequestHeaders()`
- `__getLastResponse()`
- `__getLastResponseHeaders()`

On récupère ainsi le corps de la dernière requête, les en-têtes envoyées, le corps de la dernière réponse et les en-têtes reçus.

```
<?php
$options = array(
    'exceptions'=>false,
    'trace'=>true,
);
$client = new SoapClient("fichier.wsdl", $options);
$result = $client->testMethodeAvecErreur();
if (is_soap_fault($result)) {
```

```
    echo "<p>Dernière requête : </p>\n" ;
    echo "<pre>" ;
    echo htmlspecialchars($client->__getLastRequest()) ;
    echo "</pre>" ;
} else {
    echo "Le résultat est $result" ;
}
```

## *Renvoyer une erreur dans un serveur*

Si votre serveur ne peut répondre correctement à son client SOAP, il doit renvoyer une erreur suivant un format prédéfini. PHP se charge de formater correctement le message pour peu que vous utilisiez la classe `SoapFault`.

Le constructeur de `SoapFault` prend en arguments obligatoires un code et un texte d'erreur. Vous pouvez, si vous le voulez, fournir aussi, dans l'ordre, les paramètres `faultactor` et `detail`.

```
<?php
function testMethodeAvecErreur() {
    return new SoapFault("Server", "Un message d'erreur");
}
$server = new SoapServer(null, array('uri'=>"http://example.org"));
$server->addFunction("testMethodeAvecErreur");
$server->handle();
?>
```

Si vous le souhaitez, il est aussi possible d'utiliser une syntaxe d'exception plus cohérente avec le reste de PHP 5 :

```
<?php
function testMethodeAvecErreur() {
    throw new SoapFault("Server", "Un message d'erreur");
}
$server = new SoapServer(null, array('uri'=>"http://example.org"));
$server->addFunction("testMethodeAvecErreur");
$server->handle();
?>
```

## Les templates

---

Plus vos applications grossissent, plus il devient indispensable de les structurer. Il convient de séparer votre logique métier de l’affichage : dissocier les données et la présentation est une des premières nécessités. Dans l’absolu, le programmeur ne devrait pas avoir à se soucier de la manière dont s’affichent les données pour faire son travail.

Vous avez probablement déjà vu des sites professionnels qui changent de graphisme du jour au lendemain sans période de transition. Cette rapidité est l’effet le plus visible d’une telle séparation. Il aura suffi à l’intégrateur de remplacer les différents *templates* (aussi appelés gabarits) de pages web pour que d’un seul coup tout le site change d’aspect. À aucun moment il n’y a eu modification dans la programmation et donc risque d’erreur. Nous verrons dans ce chapitre les différentes approches permettant de dissocier le contenant du contenu, puis nous aborderons trois des principales bibliothèques Open Source de ce domaine.

L’utilisation de moteurs de templates a un coût sur les performances. Les sites à fort trafic utilisent des systèmes de cache pour remédier à ce problème. Ils leur permettent de ne faire les traitements qu’une fois pour plusieurs visiteurs. Grâce à de tels systèmes, il sera possible de multiplier par 3 ou 5 le nombre de pages servies par mois. Nous verrons au chapitre suivant comment gérer ce type d’outil.

### De l’utilité des templates

Dans l’introduction, nous avons parlé de l’exemple d’un changement de présentation sur votre application, mais ce n’est qu’un effet annexe des moteurs de templates. Le principal avantage est que la logique métier de votre application et son affichage sont indépendants.



Parmi les avantages directs de l'utilisation de telles architectures nous pouvons noter :

- pouvoir décharger la création des pages à un graphiste qui n'a pas besoin de connaître PHP ;
- permettre au programmeur de modifier à volonté la logique sans risquer de dégrader l'affichage des pages ou l'obliger à comprendre la structure de l'affichage ;
- avoir plusieurs types de graphismes, par exemple un affichage dédié aux téléphones WAP, un aux téléphones i-mode, un aux navigateurs web classiques, un export sous forme XML, etc. sans que cela nécessite de dupliquer votre code ;
- changer simplement la présentation de votre application ou en proposer plusieurs à l'utilisateur ;
- assurer une indépendance entre les traitements et l'affichage.

## Moteurs de templates Open Source

Les moteurs de templates (le terme français est gabarit, mais il est peu usité) sont nombreux en PHP. Nous avons fait un tri afin de n'en sélectionner que trois. Ces solutions sont les plus abouties ou celles qui disposent de spécificités intéressantes. Notre sélection permet de couvrir la grande majorité des besoins. Parmi les autres solutions non traitées mais intéressantes, vous pourrez trouver, entre autres, Pear:IT, Modelixe, Fast-Template et Vtemplate.

### *Une solution légère : PHPLib*

Figure 23-01

*La PHPLib*



La PHPLib est la bibliothèque la plus ancienne ; elle bénéficie donc d'une maturité importante. On y trouve une syntaxe objet qui allie simplicité et performances. Si vos besoins sont peu complexes, c'est probablement la solution qui prendra le moins de ressources et sera la plus agréable à maintenir.

L'URL du projet est : <http://phplib.sourceforge.net/>

### *Le couteau suisse : smarty*

Figure 23-2

*Smarty*



Smarty est le moteur de templates le plus répandu dans le milieu PHP. Il fait un peu office de couteau suisse en offrant une solution adaptée à la majorité des cas et en permettant d'ajouter simplement des modules pour gérer le reste. Il bénéficie aussi d'un système intéressant qui permet de pré-interpréter les templates afin d'éviter une trop forte explosion du temps nécessaire à l'exécution.

L'URL du projet est : <http://smarty.php.net/>

### Un système original : Templeet

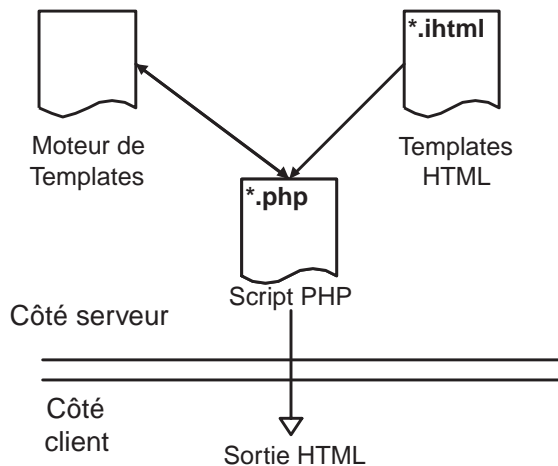
Templeet est un système développé à l'origine pour le site <http://www.linuxfr.org>. La solution retenue est originale car à mi-chemin entre un moteur de templates, un *framework* de publication et un nouveau langage. Ce n'est pas le script PHP qui appelle des templates, mais le template qui appelle des morceaux de PHP afin de récupérer les données.

Un autre gros point fort de Templeet est son système de cache performant qui permet de supporter une très forte charge. Nous en verrons plus loin le fonctionnement dans le chapitre sur les caches.

L'URL du projet est : <http://templeet.org/>

## Différentes approches

Figure 23-3  
Fonctionnement des  
templates



Le principe général des templates est de traiter et produire toutes les données avec PHP sans rien envoyer en sortie, puis d'appeler un moteur de templates en lui fournissant ces données. Le moteur interprète la page HTML générale (appelée template), recherche l'emplacement de chaque donnée et y place la valeur spécifiée. Enfin, le moteur renvoie la page HTML complète vers la sortie standard (l'affichage).

Cette description peut paraître simple, mais des fonctionnements plus complexes sont vite nécessaires pour gagner en performances ou en fonctionnalités. Dans la suite, nous allons étudier les différentes solutions possibles pour un système de templates, avec des exemples de scripts simples.

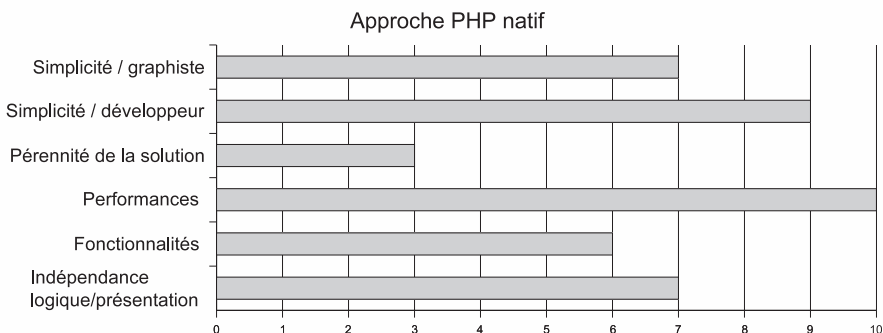
## L'approche PHP natif

### Avantages/désavantages

Cette solution est très simple à implémenter. De plus, elle est celle qui consomme le moins de ressources. Son défaut tient uniquement à la présence de code PHP dans le *template*, ce qui pourrait compliquer la visualisation du template de façon isolée. Le développement du graphisme en sera rendu plus complexe.

Figure 23-4

Approche PHP natif



La simplicité de PHP fait que l'on mélange souvent du PHP dans du HTML, juste pour quelques fonctions. Par exemple, on intègre 5 lignes de script à la fin d'une page HTML pour gérer un compteur de visites.

Cette approche est adaptée pour de petites modifications, mais elle se révèle vite désastreuse côté maintenance lorsqu'elle est utilisée pour des pages contenant beaucoup de code. À chaque modification, vous aurez à comprendre l'imbrication du code PHP et du code HTML, avec le risque de casser l'un en modifiant l'autre, sans compter la difficulté de compréhension.

Il est pourtant possible d'exploiter l'intégration de PHP dans HTML tout en gardant une séparation entre le calcul des données et leur affichage. Une démarche simple et efficace peut être de calculer toutes les données dans le fichier PHP, puis d'appeler le template avec la fonction `include()`. Dans le template se trouvent des appels PHP très simples s'occupant uniquement de l'affichage des données.

Voici un exemple du fichier PHP :

```
<?php
$link = mysql_connect() ;
mysql_select_db('livres', $link) ;
```

```
$q = 'SELECT titre, auteur, resume'
    . 'FROM livres ORDER BY date DESC LIMIT 1' ;
$result = mysql_query($q, $lnk) ;
$livre = mysql_fetch_assoc($result) ;
$titre = $livre['titre'] ;
$resume = $livre['resume'] ;
$auteur = $livre['auteur'] ;
include('template.php') ;
?>
```

Et voici un exemple du fichier de template :

```
<html>
<head><title>
livre <?php echo $titre ?>
</title></head>
<body>
<h1>livre <?php echo $titre ?></h1>
<h2>par <?php echo $auteur ?></h2>
<p><?php echo $resume ?></p>
</body>
</html>
```

#### Note

Comme à chaque exemple, nous vous présentons un code simplifié pour une meilleure compréhension. Il est de votre responsabilité d'ajouter la gestion des erreurs et, éventuellement, de vous préoccuper de la sécurité quand vous implémentez ces solutions.

Il est tout à fait possible d'aller plus loin dans cette optique et de commencer à gérer des boucles et des mises en forme particulières pour les données :

```
<?php
$lnk = mysql_connect() ;
mysql_select_db('livres', $lnk) ;
$q = 'SELECT titre, auteur, resume FROM livres ORDER BY date' ;
$result = mysql_query($q, $lnk) ;
while( $livre = mysql_fetch_assoc($result) ) {
    $livres[] = $livre ;
}
include('template.php') ;
```

Voici le fichier de template :

```
<html>
<head><title>livres</title></head>
<body>
<?php foreach($livres as $livre) { ?>
    <h1>livre
        <?php echo htmlspecialchars(ucfirst($livre['titre'])) ?>
    </h1>
    <h2>par
```

```

    <?php echo htmlspecialchars(ucwords($livre['auteur'])) ?>
</h2>
<p>
    <?php echo htmlspecialchars($livre['resume']) ?>
</p>
<?php } ?>
</body>
</html>

```

Les fonctions `ucfirst()` et `ucwords()` permettent de mettre en majuscules respectivement la première lettre de la chaîne et la première lettre de chaque mot.

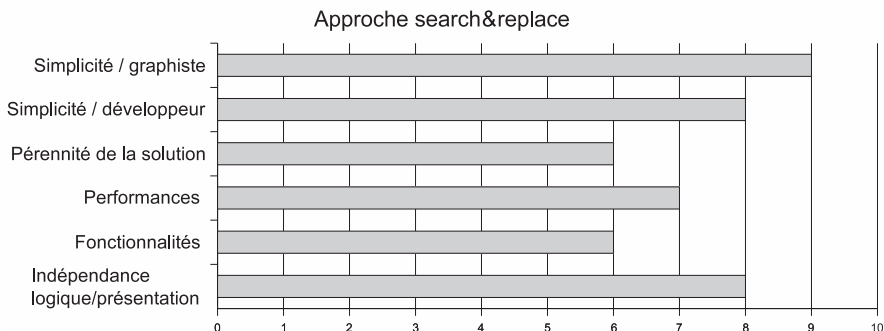
## L'approche *search&replace*

### Avantages/désavantages

Les systèmes de type *search&replace* sont très agréables à utiliser pour le graphiste, même si les fonctionnalités sont vite limitées quand on ne veut pas avoir un langage de template complexe. Malheureusement, cette facilité implique d'utiliser de fortes ressources processeur pour faire les conversions. On peut compter une baisse de performances de 10 à 20 % (pour les meilleurs moteurs) par rapport à la solution native précédente.

Figure 23-5

Approche  
*search&replace*



L'approche *search&replace* est basée sur une absence de code PHP dans le *template*. Elle permet tout d'abord de ne pas imposer l'apprentissage de PHP à l'auteur du graphisme des pages. Ensuite, elle autorise l'édition des pages HTML dans des outils de création tels que DreamWeaver sans en dénaturer le rendu.

Dans ce mode de fonctionnement, on marque les différentes parties de la page HTML avec une syntaxe spéciale et on les remplace à l'exécution par les données précalculées par PHP. Les moteurs les plus répandus utilisent des balises délimitées par des accolades (`{` et `}`). Ainsi, `{titre}` sera remplacé par le titre de la page.

En reprenant notre exemple précédent, on aurait :

```

<?php

```

```
include_once('moteur.php') ;
$link = mysql_connect() ;
mysql_select_db('livres', $link) ;
// On récupère un couple livre/auteur
$q = 'SELECT titre, auteur, resume '
    . 'FROM livres ORDER BY date DESC LIMIT 1' ;
$result = mysql_query($q, $link) ;
$livre = mysql_fetch_assoc($result) ;

// On instancie le moteur de template.
$moteur = new moteur_de_template() ;

// On associe une valeur à une variable de template
$moteur->setVar('titre', $livre['titre'] ) ;
$moteur->setVar('auteur', $livre['auteur'] ) ;
$moteur->setVar('resume', $livre['resume'] ) ;

// On lance le remplacement via le moteur de template
$moteur->affiche('template.html') ;
```

Et on aurait le fichier de template suivant :

```
<html>
<head><title> livre {titre}</title></head>
<body>
<h1>livre {titre}</h1>
<h2>{auteur}</h2>
<p>{resume}</p>
</body>
</html>
```

Le moteur a juste à remplacer le {titre} par la valeur qui lui est fournie via la méthode `setVar()`. C'est de là que vient le nom de la méthode *search&replace* (chercher et remplacer). Une implémentation simpliste de notre moteur de template pourrait être :

```
<?php
// Fichier moteur.php
class moteur_de_template {
    var $cherche = array() ;
    var $remplace = array() ;

    function setVar($nom, $valeur) {
        $this->cherche[] = $nom ;
        $this->remplace[] = $valeur ;
    }

    function affiche($fichier) {
        $html = file_get_contents($fichier) ;
        echo str_replace($this->cherche, $this->remplace, $html) ;
    }
}
```

Bien sûr, ce code ne suffit pas pour avoir un système de templates utilisable. Il faudra coder un système pour avoir des boucles et quelques autres fonctionnalités indispensables. Vous retrouverez dans la description des bibliothèques Open Source l'utilisation du moteur de templates PHPLib, qui fonctionne sur ce principe.

Ces moteurs de templates utilisent des balises telles que `{xx}` pour le contenu et des balises telles que `<!-- #xxx -->` pour les structures de contrôle. Il est ainsi possible de garder une compatibilité avec la syntaxe HTML et d'ouvrir les fichiers avec n'importe quel éditeur HTML en offrant un rendu normal. On peut aussi imaginer un système pour inclure des sous-pages dans d'autres. On aurait alors des templates généraux pour tracer les parties communes à toutes les pages et des templates plus spécifiques à chaque page, qui font appel aux premiers.

De plus, le fait qu'il n'y ait aucun code PHP à l'intérieur du template vous permet de déléguer plus facilement la page à un graphiste web.

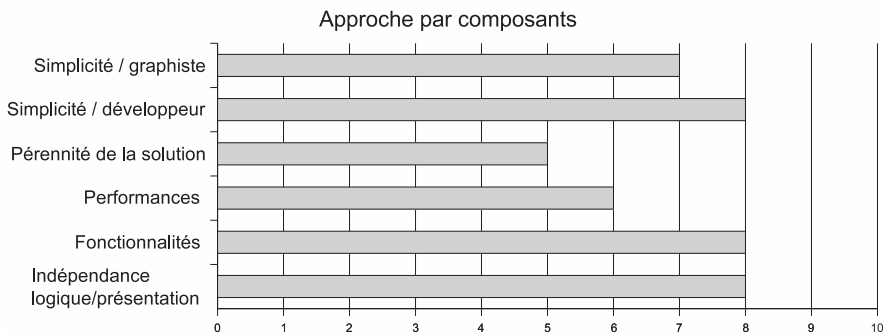
## L'approche par composants

### Avantages/désavantages

L'approche par composants revient ni plus ni moins à ajouter un niveau d'abstraction par rapport aux méthodes de templates précédentes. Chaque composant de la page (en-têtes, boîte d'identification, parties avec les copyright, menu de navigation, etc.) est indépendant des autres. Il peut être modifié de manière séparée et son contenu n'encombre pas la rédaction et l'organisation des pages. Le défaut est qu'il n'y a aucun template global, donc aucune possibilité d'avoir un aperçu général du rendu dans un navigateur avant de tester.

Figure 23-6

*Approche par composants*



L'apparition d'un même composant sur plusieurs pages complique la démarche de templates ; vous aurez à faire beaucoup d'inclusions de sous-templates dans chacune de vos pages. Si de plus le contenu de vos pages est totalement dynamique, l'affichage par un template complet peut ne pas être adapté : il faudrait faire un template pour tous les cas possibles (affichage de la boîte utilisateur ou non, affichage des dernières nouvelles ou non, affichage du forum ou non, etc.).

Il existe une autre méthode pour gérer vos contenus à l'aide de code type : la démarche par composants. Vos pages se contentent alors d'appeler chaque composant souhaité (par exemple la boîte d'authentification). La fonction gérant le composant va intercepter cet appel et calculer seule le contenu (formulaire d'authentification ou lien pour clôture de session) avant de l'afficher.

Vos pages principales ne contiennent alors que les calculs principaux et des appels à des fonctions comme `affiche_fin_de_page()` ou `affiche_menu()`. Vous n'avez pas besoin de savoir comment s'affichent ces composants ni même de connaître leur contenu. La page résultante devrait être très simple et facile à comprendre.

```
<?php
include_once('composants.php') ;
affiche_haut_de_page() ;
affiche_boite_utilisateur() ;
affiche_forum() ;
affiche_bas_de_page() ;
```

Dans une deuxième page, chaque fonction va calculer le contenu du composant et faire appel à un template HTML (par exemple un *search&replace* vu plus haut).

```
<?php
// page composants.php
include_once('moteur.php') ;

function affiche_boite_utilisateur() {
    $moteur = new moteur_de_template() ;
    if ( utilisateur_est_anonyme() ) {
        $moteur->affiche('utilisateur_anonyme.html') ;
    } else {
        $moteur->setVar('nom', recupere_nom_utilisateur() ) ;
        $moteur->affiche('_utilisateur_identifie.html') ;
    }
}
```

Il vous reste à écrire les templates HTML correspondant aux fichiers `utilisateur_anonyme.html` et `utilisateur_identifie.html`.

## Utilisation de XML et XSLT

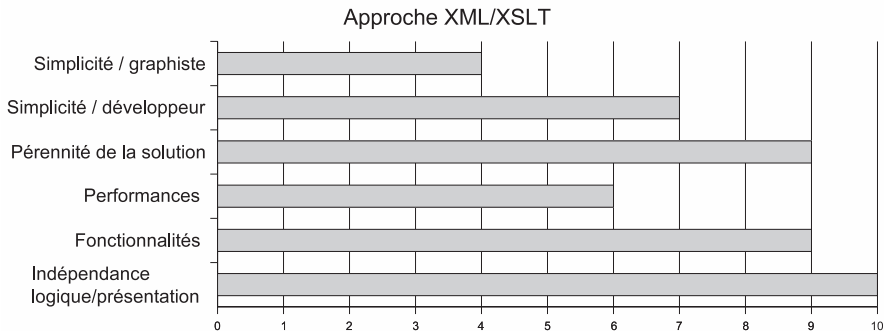
### Avantages/désavantages

L'approche XSLT est équivalente à l'approche par composants, mais les divers composants et leur organisation sont décrits dans un langage XML standardisé. Des modules programmés en C, rapides, pourront faire la transformation. Le code PHP est réduit à la création d'un contenu XML. Le désavantage est l'apprentissage du langage XSLT. S'il n'est pas extrêmement complexe, il demande tout de même une implication importante.



Figure 23-7

Approche  
XML / XSLT



L'approche par composants résout la plupart des besoins mais reste imparfaite. Il est par exemple impossible, dans le calcul du rendu, d'avoir une logique complexe, car chaque composant est totalement indépendant.

Si vos besoins sont plus pointus, une autre approche s'impose : l'utilisation d'un moteur XSLT. Vous faites des sorties simples en XML et le moteur XSLT transforme vos données à l'aide d'une feuille de transformation XSL. Dans votre feuille XSLT, vous pouvez faire tout type de traitements, de conditions ou de transformation. Les possibilités sont en général bien plus grandes que dans un moteur de template classique.

L'avantage vient du fait que vous vous reposez sur des technologies standardisées, répandues et éprouvées. Si un jour vous devez faire intervenir un composant dans une autre technologie que PHP, il pourra s'interfacer avec votre structure existante ; de même, il sera simple de trouver des éditeurs XSLT pour modifier vos feuilles de transformation.

Avec cette solution, vous bénéficiez des importantes ressources XSLT disponibles pour l'export ou l'import de documents. La communication entre votre application et les autres est garantie.

Nous ne développerons pas d'exemple de transformation XSLT ici ; vous pouvez vous reporter au chapitre 21 concernant XML avancé.

## Analyse et choix

Maintenant que nous avons vu les différentes approches possibles, il est temps de les comparer en analysant leurs performances et leurs capacités.

Il y a une suite de critères que vous devez prendre en compte dans votre choix. Ces critères sont à pondérer différemment suivant vos contraintes, mais évitez de donner une trop grande importance aux performances : le coût d'un serveur supplémentaire est souvent négligeable par rapport aux temps de développement, de maintenance et de mise à jour. La simplicité et la pérennité influenceront probablement plus vos coûts sur le long terme. Voici quelques sujets à étudier :

- la pérennité de votre solution ;

- la simplicité des modèles pour les graphistes et les intégrateurs ;
- la simplicité d'utilisation et l'efficacité pour les développeurs ;
- les performances du moteur.

### ***Pérennité de la solution retenue***

Quelle que soit la solution choisie, il faut penser à ce qu'elle deviendra dans le futur. Une solution qui vous convient parfaitement pour l'instant peut se révéler inadaptée par la suite.

Ainsi, si vous choisissez un moteur déjà existant, il faut vous assurer qu'il continuera à être maintenu, que les éventuels bogues seront corrigés et qu'il ne sera pas laissé à l'abandon. Sur ce point, nous ne présentons ici que les moteurs Open Source qui nous paraissent avoir les garanties nécessaires, mais il est possible de mettre en avant le choix d'un moteur XSLT. En effet, cette solution a l'avantage d'être une solution complète et standardisée, qui ne sera pas abandonnée du jour au lendemain.

Les possibilités d'interconnexions de votre application avec le monde extérieur sont aussi à prendre en compte. Il est envisageable dans certains cas que votre application doive s'interfacer avec d'autres technologies dans le futur. Il est même possible qu'une contrainte ou une autre vous amène à changer totalement de langage. L'utilisation de templates en PHP natif vous bloquera dans vos possibilités. Inversement, l'utilisation d'une architecture de type *search&replace* vous permettra de récupérer plus ou moins facilement vos modèles pour les porter vers une autre architecture. Ici aussi, c'est la solution XSLT qui prend l'avantage, car elle sera disponible quel que soit le langage (ou l'application) avec lequel s'interfacer. Il sera même possible de partager les feuilles de transformation entre deux de vos applications afin de faciliter une migration.

Un dernier point à prendre en compte est la possibilité d'extension de votre solution. Si votre application gagne en complexité et s'étoffe dans le futur, votre moteur de templates sera-t-il toujours adapté ? Nul autre que vous ne peut savoir quelles évolutions sont envisageables, mais une solution en PHP natif sera probablement peu intéressante si la quantité de données par page augmente : leur organisation manquera alors de structure.

### ***Simplicité pour les graphistes***

Un reproche fréquent fait aux systèmes de templates est qu'au final ils ne simplifient pas grand-chose : au lieu de devoir faire apprendre la programmation d'un langage simple (PHP) aux graphistes, il faudra leur apprendre un langage de templates plus ou moins bien fait et totalement propriétaire.

Pour des systèmes simples, le problème ne se pose pas, mais il est important d'éviter de chercher à faire trop compliqué. Toute exagération dans la structure aura un

impact direct sur le temps nécessaire au développement des templates et sur leur maintenance.

Des templates en PHP natif seront plus complexes à gérer par des graphistes. Un moteur basé sur XSLT nécessitera la rédaction de feuilles de transformation par des programmeurs et intégrateurs ; il n'est pas vraiment envisageable de les faire gérer directement par un graphiste. Un système *search&replace* est ce qui se révélera plus simple.

D'un autre côté, la gestion XSLT sera facilitée pour les intégrateurs par le fait qu'il existe de nombreuses ressources disponibles et que beaucoup de transformations vers les formats courants sont déjà réalisées et accessibles.

Un autre détail à évaluer est la possibilité d'intégrer des templates externes sans risques de sécurité. On se rend bien compte qu'avec une solution en PHP natif, le développeur du template obtient un accès total au système. Les moteurs simples de *search&replace* limitent très fortement les possibilités d'interaction avec le système. D'autres solutions, comme Smarty, ont un mode bridé permettant d'assurer la sécurité si vous n'avez pas confiance dans les auteurs des templates.

### ***Simplicité pour les développeurs***

On a souvent l'habitude de négliger la simplicité du code au profit des fonctionnalités ou des performances ; le programmeur est là de toute façon pour traiter ce qui est complexe. Pourtant une solution simple à ce niveau entraîne des avantages clairement identifiables : non seulement un code plus rapide à créer, mais aussi avec moins de risques de bogues et une maintenance très facilitée.

Une solution native PHP sera la plus simple à gérer pour des faibles besoins. La suite dépend beaucoup de la structure de votre application, mais l'option XSLT se révélera probablement l'unique solution pour des transformations très complexes.

### ***Les performances du moteur***

Les performances sont probablement le talon d'Achille des moteurs de templates. L'interprétation d'un template en PHP natif ne prend virtuellement aucune ressource, mais les autres solutions sont presque toutes très gourmandes.

Le premier ralentissement vient du fait qu'il faut attendre toutes les données (donc les stocker en mémoire) et charger l'intégralité du template en mémoire. Non seulement le serveur ne peut pas renvoyer le résultat au fil de son calcul, mais en plus il se retrouvera avec des documents potentiellement gros en mémoire.

Le deuxième ralentissement vient de l'interprétation du template en lui-même. Cela est d'autant plus vrai si vous utilisez un langage avec des boucles, des fonctions, des structures de contrôle et des fonctionnalités avancées.

Les performances peuvent chuter de 10 à 20 % pour les systèmes les plus simples de *search&replace* ou s'effondrer à 50 % pour les moteurs les plus complexes (pour plus de détails, vous pouvez vous reporter à l'étude de Globalis sur le sujet, que vous trouverez à l'adresse <http://www.globalis-ms.com/publications.php>). Il est à noter que, malgré la complexité de XSLT, on y retrouve des performances similaires aux autres solutions, car la transformation n'est pas faite en PHP mais avec une bibliothèque C, plus rapide.

Pour compenser ces pertes, les moteurs de templates mettent en place une série d'astuces et des principes de cache. Un système de cache peut vite se révéler indispensable. Vous trouverez une analyse de différentes solutions au chapitre suivant.

## Bibliothèques Open Source

Ces pages ne nous permettent pas de donner un aperçu de toutes les solutions existantes, nous nous contenterons de citer les quelques autres que vous pourriez retrouver avec leurs défauts et leurs avantages :

- **FastTemplate** : il s'agit du portage d'une bibliothèque Perl. Elle vous permettra donc de passer d'un langage à l'autre assez facilement. C'est malheureusement son seul avantage car les performances de l'implémentation PHP sont désastreuses.
- **Pear::IT[X] (ou IsoTemplate)** : il s'agit d'une solution simple de *search&replace* du type de PHPLib, mais probablement moins performante. Son avantage est de faire partie du dépôt Pear (<http://pear.php.net>) et donc de bénéficier d'une assistance constante.
- **Vtemplate** : équivalent de PHPLib, les performances sont similaires, avec un léger avantage à PHPLib.
- **Modelixe** : une solution originale basée sur une syntaxe XML. La syntaxe est complexe et les performances peu intéressantes dans sa version PHP.

## PHPLib

PHPLib est une bibliothèque orientée objet très complète pour PHP. Elle était très répandue du temps de PHP 3. Avec l'arrivée de PHP 4 et l'implémentation par défaut de la gestion des sessions, la PHPLib s'est vue moins utilisée.

Elle contient pourtant un moteur de templates simple et très performant. Vous pouvez n'utiliser que ce sous-système de la PHPLib si vous ne vous servez pas du reste.

### Installation

Vous pouvez télécharger la bibliothèque à l'adresse <http://phplib.sourceforge.net/>. Seul le fichier nommé `template.inc` nous intéresse. Mettez-le à un endroit d'où vous pourrez l'inclure et laissez le reste qui nous est inutile.

## Utilisation basique

Le système de la PHPLib est un moteur *search&replace* très basique. Dans vos templates, les chaînes {variable} sont remplacées par les paramètres fournis au moteur. Voici un exemple de template simple :

```
<html>
<head><title>{titre}</title></head>
<body><p>{texte}</p></body>
</html>
```

Après l'inclusion du script de la PHPLib, il vous faut instancier un objet de la classe Template. L'argument à fournir au constructeur est le répertoire contenant les templates.

```
<?php
include('template.inc') ;
$moteur = new Template('/chemin/vers/les/templates/') ;
```

Vous pouvez charger un template grâce à la méthode `set_file()`. Il faut fournir un nom pour identifier le template et son adresse (relative au préfixe donné au constructeur). Il est aussi possible de charger d'un coup une série de templates en fournissant un tableau associatif comme argument, avec les identifiants comme clés et les adresses comme valeurs.

```
$moteur->set_file('index' , 'index.html') ;
$moteur->set_file( array(
    'articles' => 'articles.html' ,
    'commentaires' => 'commentaires.html'
) ) ;
```

Il reste maintenant à fournir au moteur les valeurs des variables du *template*. La méthode `set_var()` permet de définir les valeurs à remplacer.

```
$moteur->set_var('titre', 'mon titre') ;
$moteur->set_var('texte', 'texte de la page') ;
```

La méthode `parse()` permet de déclencher le remplacement des variables fournies dans un template précis. Le premier argument à fournir est un nom de variable, le second est un identifiant de template utilisé par `set_file()`. Le résultat du remplacement est mis dans la variable spécifiée, qui peut être utilisée par la suite dans un autre *template*, ou affichée. Un troisième argument peut optionnellement être fourni. Il s'agit d'un booléen : s'il est vrai, alors le résultat est ajouté à la variable spécifiée au lieu d'en remplacer le contenu.

```
$moteur->parse('resultat', 'index') ;
```

Pour afficher le contenu d'une variable retournée par `parse()`, il nous reste finalement à utiliser `p()`. Il aurait été aussi possible de faire les deux étapes en une seule fois grâce à `pparse()`.

```
$moteur->p('resultat') ;
```

## Utilisation des blocs et itérations

Il est aussi possible d'utiliser des blocs dans les templates, afin de permettre la répétition d'un même type d'information. Ces blocs sont délimités par `<!-- BEGIN nom_du_bloc -->` et `<!-- END nom_du_bloc -->`. Il s'agit de commentaires HTML, qui ne génèrent donc pas l'édition du template dans un éditeur graphique.

On peut imaginer le template suivant :

```
<html>
<head><title>{titre_page}</title></head>
<body>
<!-- BEGIN article -->
  <h1>{titre_article}</h1>
  <p>{texte_article}</p>
<!-- END article -->
</body>
</html>
```

La méthode `set_block()` permet d'affecter une variable à un bloc. Lors de l'interprétation du template global, le bloc est remplacé par le contenu d'une variable. Le premier argument à fournir est l'identifiant du template utilisé dans `set_file()`, le deuxième est le nom du bloc dans ce *template*, le troisième est le nom de la variable utilisée. Par la suite, l'utilisation reste dans le cadre de ce que nous avons déjà vu.

```
$moteur->set_block('index', 'article', 'var_bloc');
```

L'utilisation finale de notre template ressemblera à ce qui suit :

```
<?php
include('template.inc');
$moteur = new Template('/chemin/vers/les/templates/');
$moteur->set_file('index', 'template.html');
$moteur->set_block('index', 'article', 'var_bloc');
for ($i=0 ; $i < 10 ; $i++) {
  $moteur->set_var('titre_article', 'Article '.$i);
  $moteur->set_var('texte_article', 'contenu '.$i);
  $moteur->parse('var_bloc', 'article', TRUE);
}
$moteur->setVar('titre_page', 'Derniers articles');
$moteur->pparse('resultat','index');
```

## Inclusion de templates externes

Les méthodes `set_file()` et `parse()` suffisent à gérer des fichiers inclus. Pour faire une inclusion :

- déclarez vos templates grâce à `set_file()` ;
- remplacez le contenu du bloc à inclure grâce à `setVar()` ;

- exécutez le remplacement du bloc inclus grâce à `parse()` et mettez le résultat dans une variable du bloc parent ;
- interprétez le reste du bloc parent.

Ainsi, vous venez de remplacer une variable du bloc parent par le contenu d'un sous-*template*.

## Smarty

Le développement de Smarty vient à l'origine d'Andreï Zmievski, un des développeurs de PHP. Vous pouvez d'ailleurs trouver les ressources du projet sur un site de `php.net`, à l'adresse <http://smarty.php.net/>.

Smarty est le couteau suisse des solutions de `templates`. Il offre à peu près toutes les fonctionnalités espérées à travers un langage propre. Malheureusement, c'est aussi ce que certains lui reprochent, car l'utilisation de tout l'outil nécessite un apprentissage de ce langage.

### Principe de fonctionnement

Pour compenser la complexité du langage interne (et donc la lenteur de l'interprétation des templates), Smarty utilise une méthode originale alliant la souplesse des solutions *search&replace* aux performances des solutions en PHP natif.

À la première lecture de vos templates, le moteur les interprète et remplace les pseudo-codes par du code PHP natif. Pour les futures utilisations, Smarty se contentera d'inclure le code produit. Les fonctionnalités restant assez importantes, vous n'aurez pas les performances d'une solution simple, mais le système permet de contrebalancer la grosseur du code et d'arriver à des performances honorables.

### Installation

Une fois l'archive téléchargée, mettez les différents scripts du répertoire `libs` dans un répertoire où ils seront accessibles lors des inclusions (par exemple un répertoire spécifié dans la directive de configuration PHP `include_path` de votre fichier `php.ini` ; voir à ce sujet le chapitre 2 sur l'installation et la configuration de PHP).

Smarty utilise plusieurs répertoires pour ses besoins internes : un pour les templates (`template_dir`, par défaut `./templates/`), un pour les templates convertis en PHP (`compile_dir`, par défaut `./templates_c/`), un pour les fichiers de configuration de vos templates (`config_dir`, par défaut `./configs/`) et un pour le cache des résultats (`cache_dir`, par défaut `./cache/`). Ces répertoires doivent être créés, et les répertoires pour les templates compilés et le cache doivent être accessibles en écriture pour le serveur web. Vous pourrez changer l'adresse de ces répertoires à l'exécution en accédant aux propriétés correspondantes :

```
$smarty->template_dir = './templates/';  
$smarty->compile_dir = './templates_c/';
```

```
$smarty->config_dir = './configs/';  
$smarty->cache_dir = './cache/';
```

## Développement des templates

Nous allons tout d'abord nous occuper du pseudo-langage de Smarty et de ce qu'il est possible de faire dans les templates du point de vue du graphiste. Nous verrons leur utilisation du côté programmeur par la suite.

La syntaxe est entièrement modifiable (on peut par exemple changer les délimiteurs utilisés pour les variables) et les fonctionnalités sont très importantes. Nous nous contenterons de décrire le comportement par défaut et les utilisations les plus courantes. Vous trouverez des renseignements plus complets dans l'aide en ligne à l'adresse <http://smarty.php.net/manual/>.

### Description de la syntaxe

Les commentaires sont délimités par `{* et *}` ; ils servent uniquement à illustrer le template et ne se voient pas dans les pages créées.

```
{* ceci est un commentaire *}
```

Les fonctions sont délimitées par des accolades simples. Les arguments sont passés séparés par des espaces, avec une syntaxe `nom=valeur`.

```
{fonction1}  
{fonction2 arg1="valeur1" arg2="valeur2"}
```

Certaines fonctions sont faites pour contenir du code HTML ; elles fonctionnent comme des éléments XML ou HTML, mais avec des accolades.

```
{if ...}  
code HTML  
{/if}
```

Les attributs sont pour la plupart des chaînes de caractères et doivent être délimités par des guillemets. Les variables sont à préfixer par un `$` comme en PHP, les paramètres de configuration sont délimités par deux `#`. Certaines constantes comme `yes`, `true`, `on`, `off`, `false` et `no` peuvent être utilisées directement.

```
{include file="fichier"}  
{include file=$fichier}  
{include file=#fichier#}  
{html_select_date display_days=yes}
```

Smarty interprète ces variables dans les chaînes entre guillemets de la même manière que PHP. Pour le forcer à interpréter une syntaxe qui peut porter à confusion, il faut la délimiter par des apostrophes inversées (*backticks* : ```).

```
"chaîne $variable"  
"chaîne $variable[0]"  
"chaîne ` $variable `chaîne"
```



Les différents éléments d'un tableau associatif sont accessibles via l'opérateur point . (et non pas des crochets comme en PHP). Les tableaux indexés et les objets sont accessibles par la même notation que PHP.

```
$tableau.ligne.colonne  
$tableau[0]  
$objet->propriete
```

### Les modificateurs

Smarty propose un concept de modificateur pour mettre en forme les variables dans le template plutôt que dans le code PHP qui traite les actions. Pour appliquer un modificateur à une valeur, il suffit de la faire suivre par un *pipe* (caractère |) et le nom du modificateur. Ainsi, le code suivant affiche la variable après l'avoir convertie en majuscules :

```
{ $variable|upper }
```

Certains modificateurs peuvent nécessiter des arguments ; ils sont alors séparés par le caractère :. Le code suivant tronque à 40 caractères et remplace la chaîne tronquée par trois points de suspension :

```
{ $variable|truncate:40:"..." }
```

Les modificateurs peuvent être enchaînés à volonté.

```
{ $variable|upper|truncate:40:"..." }
```

Il serait trop long de décrire tous les modificateurs, mais vous pouvez compter parmi les plus courants :

- `capitalize` : met en majuscule la première lettre de chaque mot.
- `date_format` : formate une date avec le format passé en argument. La syntaxe à utiliser pour le format est la même que pour la fonction PHP `strftime()`.
- `default` : si la valeur est vide ou si la variable n'existe pas, renvoie l'argument fourni à la place.
- `escape` : remplace les caractères spéciaux d'une chaîne. Le paramètre définit le format de la chaîne et les remplacements à effectuer. Avec le paramètre `html`, les caractères `&`, `"` et `<` sont convertis en entités (`&amp;`, `&quote;` et `&gt;`). Avec `url`, les caractères spéciaux sont convertis en `%xx` (où `xx` est la valeur hexadécimale du caractère) pour pouvoir être affichés dans une adresse Internet. Vous pouvez aussi trouver les valeurs `quotes` ou `javascript`.
- `lower` et `upper` : mettent la chaîne respectivement en minuscules et en majuscules.
- `n12br` : équivalent du PHP `n12br()`, c'est-à-dire qu'il remplace les sauts de ligne par des `<br>`.

- `regex_replace` : fait un remplacement par expression régulière. Le premier paramètre est l'expression « capturante », le deuxième est l'expression de remplacement.
- `replace` : remplace une sous-chaîne fournie en premier argument par une autre, fournie en second argument.
- `strip_tags` : équivalent du PHP `strip_tags()`, c'est-à-dire qu'il supprime tout ou partie des tags HTML.

### Les structures de contrôle et les fonctions internes

Comme précédemment, nous ne présentons que les fonctionnalités les plus utilisées ; Smarty est trop complexe pour une description exhaustive.

Les mots-clés sont similaires à ceux de PHP : `foreach` permet de traverser un tableau, `include` d'inclure un autre fichier à interpréter, `if elseif` et `else` de gérer les conditions.

```
{foreach from=$tableau item=valeur}
    contenu: {$valeur}<br>
{/foreach}

{include file="template.html"}

{if $var eq "A"}
    var = A
{elseif $name eq "b"}
    var = B
{else}
    var n'est ni A ni B
{/if}
```

Les opérateurs dans les tests sont les mêmes qu'en PHP, auxquels s'ajoutent des opérateurs dédiés comme `eq` pour vérifier l'équivalence de deux chaînes de texte.

`{rdelim}` et `{ldelim}` permettent de renvoyer les accolades, qui sont autrement interprétées par le moteur.

Le contenu entre `{literal}` et `{/literal}` est renvoyé tel quel, non interprété. C'est principalement utile pour renvoyer du JavaScript sans avoir à utiliser `{rdelim}` et `{ldelim}` à outrance.

Les sections à répéter sont gérées par `{section}`. Il ne s'agit ni plus ni moins que de parcourir des tableaux. L'argument `loop` permet de savoir combien de passages doivent être effectués ; il compte simplement combien d'éléments sont dans la variable fournie. L'argument `name` permet de nommer la boucle ; il est utilisé comme index pour les tableaux.

```
{section name=nom_boucle loop=$titre}
titre de l'article: {$titre[nom_boucle]}<br>
    test de l'article : {$texte[nom_boucle]}<br>
{/section}
```

Ainsi, le code précédent pourrait correspondre au code PHP suivant :

```
for($nom_boucle=0 ; $nom_boucle<count($titre) ; $nom_boucle++) {  
    echo "titre de l'article ",$titre[$nom_boucle],<br>' ;  
    echo "texte de l'article ",$texte[$nom_boucle],<br>' ;  
}
```

### Utilisation des templates

L'utilisation des templates est heureusement plus simple que ne l'est leur pseudo-langage. Commencez par charger le moteur de Smarty avec une simple inclusion, puis instanciez un objet de la classe Smarty :

```
include_once('smarty.class.php') ;  
$smarty = new Smarty() ;
```

Il est ensuite possible de modifier une liste importante de paramètres pour le moteur. Nous nous contenterons de signaler un mode sécurité activable afin de pouvoir intégrer en toute confiance des templates faits par quelqu'un d'extérieur. Les fonctionnalités importantes de Smarty donneraient sinon un contrôle trop important pour un auteur en qui vous n'avez pas confiance.

L'assignation des variables se fait avec la méthode `assign()`. Il est possible de donner en argument un seul couple nom/valeur ou de fournir un tableau associatif.

```
$smarty->assign('nom', 'valeur') ;  
$smarty->assign(array( 'nom1'=>'valeur 1', 'nom2'=>'valeur 2' ))
```

Pour utiliser les blocs à répéter du *template*, il suffit de les considérer comme des tableaux. Pour utiliser notre bloc en exemple plus haut, il nous faut utiliser une assignation du type suivant :

```
$smarty->assign('titre', array( 'titre 1', 'titre 2', 'titre 3' )) ;  
$smarty->assign('texte', array( 'texte 1', 'texte 2', 'texte 3' )) ;
```

Afin de faciliter l'utilisation des boucles, vous pouvez utiliser la méthode `append`, qui ajoute une valeur à un tableau. Le code précédent devient alors :

```
$smarty->append('titre' , 'titre 1' ) ;  
$smarty->append('titre', 'titre 2' ) ;  
$smarty->append('texte' , 'texte 1' ) ;  
$smarty->append('texte', 'texte 2' ) ;
```

Pour afficher le résultat une fois les remplacements faits, il faut utiliser la méthode `display()`, avec le nom du template en argument. La méthode `fetch()` est identique mais retourne le résultat au lieu de l'afficher.

```
$smarty->display("template.html") ;  
$resultat = $smarty->fetch("template.html") ;
```

Smarty permet aussi de déterminer des fonctions ou modificateurs personnalisés à utiliser dans vos templates. Leur définition dépasse le cadre de ce livre et je vous invite à consulter la documentation en ligne du projet, plus complète.

## Utilisation du cache

En raison du grand nombre de fonctionnalités, le cache est indispensable dans Smarty. Vous pouvez l'activer en mettant à 1 l'attribut `caching` avant d'utiliser le template :

```
$smarty->caching = 1;
```

À partir de ce moment, quand le résultat est affiché, il est aussi sauvegardé dans le répertoire de cache. Il y restera jusqu'à ce qu'il expire (le temps d'expiration est d'une heure par défaut). La prochaine fois, le cache sera utilisé au lieu de recalculer le contenu.

Il est aussi possible de définir une date d'expiration personnalisée en mettant la propriété à 2 et en spécifiant une nouvelle période de validité, la valeur -1 définissant un cache qui n'expire jamais :

```
$smarty->caching = 2;  
$smarty->cache_lifetime = 3600 ;
```

Une fois ce cache activé, Smarty fournit une méthode intéressante pour éviter de recalculer les données si celles-ci se trouvent déjà en cache. La méthode `is_cached()` vous permet de savoir si une copie du template est actuellement en cache et en période de validité. Si ce n'est pas le cas, et seulement si ce n'est pas le cas, alors il vous faut calculer les données avant affichage :

```
$smarty->caching = 2;  
$smarty->cache_lifetime = 3600 ;  
if ($smarty->is_cached('template.tpl')) {  
    /* calcul des données et assignations */  
    $smarty->assign( ... ) ;  
    $smarty->assign( ... ) ;  
}  
$smarty->display('template.tpl') ;
```

Ce système permet d'utiliser en priorité le cache et, dans cette hypothèse, d'éviter les importants calculs nécessaires au fonctionnement normal.

Bien entendu, le résultat d'un même template n'est pas toujours identique. Par exemple, pour un même template permettant l'affichage d'articles, il y aura un résultat différent par article. Si vous pouvez individualiser de cette manière plusieurs résultats, il sera tout de même possible d'utiliser le cache en fournissant un identifiant en second argument des méthodes `display()` et `is_cached()`. Smarty gèrera alors plusieurs caches pour la même page indépendamment.

```
$smarty->display('articles.tpl', $identifiant_article);
```

Les résultats mis en cache peuvent être effacés par la méthode `clear_cache()` en fournissant le nom du template en argument. Tous les résultats peuvent aussi être effacés d'un coup avec `clear_all_cache()`.

```
$smarty->clear_cache('template.tpl') ;  
$smarty->clear_all_cache() ;
```

## Templeet

Le moteur Templeet a été créé pour la nouvelle version du site <http://www.linuxfr.org/>. L'accent a été mis sur la simplicité et les performances.

La structure utilisée est originale. Ce sont les templates qui font appel aux fonctions PHP pour obtenir les données et les faire traiter, au lieu que ce soit vos scripts PHP qui traitent la logique et l'envoient au moteur.

Il en résulte que Templeet ne garantit pas l'indépendance du contenu de la logique et de la présentation. Si c'est ce que vous cherchez, alors il vaut mieux vous reporter vers une solution plus classique. À vrai dire, Templeet n'est même pas totalement un système de templates : il s'agit d'un mélange entre un système de publication, un outil de gestion de contenu et un moteur de templates.

Pourtant, l'approche est suffisamment intéressante pour être traitée dans ce chapitre : elle permet une rédaction simple des différentes pages et de leur rendu. En étant rigoureux, il est possible d'avoir une approche par composants intéressante.

L'installation n'est pas décrite dans ces lignes, car elle est entièrement automatisée et il vous suffit de télécharger un fichier PHP qui va vous installer et configurer entièrement l'application. Vous trouverez cet installateur, ainsi que les différentes ressources sur Templeet, à l'adresse <http://www.templeet.org/>.

### Structure générale

Quand le visiteur appelle une page de type `repertoire/sousrep/page.html`, Templeet intercepte l'appel et charge le template `template/repertoire/sousrep/page.tpl`. Il est toutefois possible d'associer plusieurs URL au même template en spécifiant les adresses dans un paramètre de configuration (le tableau `$config['html2template_array']`).

Ce template est ensuite interprété et les chaînes de type `~chaine(...)` y sont vues comme des fonctions à lancer par le moteur. Par exemple, le code suivant affiche le nom du template utilisé :

```
<html>
<head><title>Templeet</title></head>
<body>
<p> Ce template s'appelle : ~get_filename() </p>
</body>
</html>
```

#### Note

Le jeu de fonctions fourni par défaut est très réduit. Par la suite, vous pouvez intégrer des modules supplémentaires selon vos besoins. Cette position volontaire est prise afin d'éviter l'explosion de la quantité de code à interpréter comme on peut le voir sur d'autres moteurs.

Templeet sait aussi gérer une auto-négociation de la langue pour les sites internationaux. Le moteur se sert des informations envoyées dans les en-têtes de la requête HTTP par

votre navigateur pour déterminer la langue à utiliser. Ainsi, si votre navigateur demande prioritairement une page française et en second choix une page anglaise, Templeet charge le template avec comme extension `.fr.tpl` au lieu de `.tpl`. S'il n'existe pas, c'est `.en.tpl` qui est utilisée et ainsi de suite. Si aucune correspondance n'est trouvée entre les langues disponibles dans les templates et celles demandées par le navigateur, le template par défaut est chargé.

## Système de cache

Comme nous l'avons vu plus haut, un système de cache est souvent nécessaire pour accompagner une solution de templates. C'est d'autant plus vrai pour Templeet, qui a été créé pour soutenir une très forte charge. Le système choisi est un cache créé sur demande et directement accessible.

Quand le visiteur demande la page `index.html` et qu'elle n'existe pas, Templeet charge le *template*, l'interprète et enregistre le résultat dans `index.html` avant de la renvoyer au visiteur. Quand le prochain visiteur fera la même requête, le fichier existera déjà et sera servi directement par Apache sans faire intervenir PHP.

Ce système obtient très facilement des performances proches de celles des pages statiques, les fichiers créés étant effacés sur des critères de temps ou sur demande (lors d'une modification des données sources par exemple).

La configuration d'un tel système tient uniquement à la directive `ErrorDocument` d'Apache : quand une URL n'existe pas, le serveur appelle le fichier spécifié (dans notre cas, un script PHP qui reconstruit la page manquante).

```
■ ErrorDocument 404 /templeet.php
```

## Les fonctions et modules fournis

Les différentes fonctions disponibles dans Templeet sont séparées en modules, de façon à n'inclure que ce qui est nécessaire.

Nous ne présentons ici que deux modules pour vous permettre de comprendre le fonctionnement. Les autres modules peuvent vous permettre de gérer les fichiers, les URL, les paramètres passés dans la requête HTTP, d'utiliser des expressions régulières, etc. Vous en trouverez une liste avec la documentation à l'adresse <http://templeet.org/doc.fr.html##fonctions>.

Il faut bien comprendre que les modules doivent permettre de faire l'intégralité de la logique sur votre application. Il n'y a pas de fichier PHP qui traite la logique pour fournir le résultat aux templates ; c'est le template qui calcule lui-même les données et les traite.

### Le module Core

Core est le nom du module principal, celui qui gère les fonctions de base du moteur. Il contient un nombre volontairement réduit de fonctions.

`~set()` et `~get()` permettent de définir ou afficher une variable. Ainsi, le code suivant affiche « valeur » :

```
~set('variable', "valeur")
~get('variable')
```

`~rem()` sert à délimiter un commentaire (donc qui ne se verra pas dans le rendu) :

```
~rem("commentaire de code templeet")
```

`~include()` sert à inclure un autre fichier (relatif au répertoire du template en cours). Ce fichier sera interprété par Templeet de la même façon qu'un autre template. Vous pouvez de plus passer au fichier une série d'arguments : `~include('fichier.html', 'argument 1', 'argument 2', ...)`. Les arguments sont récupérables dans le fichier appelé par la fonction `~parseparam()` et sont numérotés à partir de 1. Ainsi, l'inclusion précédente `~parseparam(2)` affichera « argument 2 » (sans les guillemets).

```
~include('fichier.html', "argument 1", "argument 2", ...)
~parseparam(2)
```

`~eval()` a une signification similaire à son homonyme PHP. La fonction permet d'interpréter la chaîne passée en argument comme des commandes Templeet. Il vous est donc possible de créer du code dynamiquement.

Il existe deux structures de contrôle dans le module principal : une condition et une boucle. `~if()` prend un test en premier argument, le deuxième est évalué si le test est vrai, le troisième si le test est faux. `~while()` évalue le deuxième argument tant que le premier est évalué à vrai.

```
~if( 2 > 3 , "le test est vrai", "le test est faux" )
~set('compteur',1)
~while(~get('compteur')<=3,
      '~get('compteur')
      ~set('compteur ',~get('compteur ')+1)'
    )
```

Vous avez aussi accès à quelques fonctions arithmétiques : `~plus()`, `~minus()`, `~multiple()`, `~divide()` permettent d'utiliser les quatre opérations : addition, soustraction, multiplication, division.

```
~plus(1,1) = 2
~soustraction(2, 1) = 1
~multiple(2,3) = 6
~divide(6,3) = 2
```

## Le module Cache

Le module Cache permet d'ajuster les paramètres de gestion du cache.

`~uncache()` prend en paramètre un fichier de cache (/ définissant le répertoire racine des caches et non le répertoire racine système.). Ce fichier est effacé afin d'être reconstruit au prochain passage. Si vous fournissez un répertoire, c'est tout son contenu qui sera effacé.

```
~uncache("fichier.html")  
~uncache("/")
```

`~dont_cache()` et `~set_expiretime()` permettent de contrôler la gestion du fichier en cours. Le premier permet d'empêcher que Templateet mette le résultat en cache, le deuxième permet de faire recréer le contenu après le nombre de secondes passées en paramètre.

```
~set_expiretime("3600")  
~dont_cache()
```

Il est aussi possible de gérer le cache des éléments inclus avec `~include()`. `~includewith-cache()` permet de créer le cache lors de l'inclusion et `~uncache_include()` permet de le détruire.





# Les systèmes de cache

---

Au cours de ces dernières années, les sites Internet sont devenus de plus en plus dynamiques. Le bon côté des choses est l'amélioration de l'interactivité, mais le revers de la médaille est que les serveurs web sont de plus en plus sollicités.

Dans le cas d'applications nécessitant l'exécution d'un grand nombre de requêtes simultanées, il peut en découler des temps d'attente importants. De trop nombreuses demandes simultanées peuvent également engendrer une surcharge du serveur, ce qui entraînerait l'incapacité et donc le refus du serveur de répondre à certaines requêtes. La conséquence serait alors une indisponibilité du service (on parle de déni de service).

Pour remédier à ce type de problèmes ou tout simplement pour améliorer les temps de réponse, on peut utiliser des systèmes de cache. Cela permet d'éviter de refaire plusieurs fois le même calcul. Nous parlerons tout d'abord des systèmes de cache globaux, puis nous traiterons des caches HTTP, et enfin nous aborderons trois des principaux systèmes de cache Open Source existants.

## De l'utilité des caches

Comme nous le voyons dans d'autres chapitres avec les systèmes de templates ou la création d'images, les ressources utilisées par certaines opérations sont parfois importantes. Il devient alors nécessaire de précalculer le rendu pour alléger la charge du serveur. On évite alors une exécution complète des scripts à chaque requête. Ce système est appelé principe de cache (le cache étant constitué des pages précalculées).

L'unique but des systèmes de cache est de pouvoir utiliser des scripts lents ou lourds pour le serveur, tout en ayant des temps de réponse proches d'un site avec fichiers statiques. Le système de cache ne sera donc pas forcément nécessaire pour une page effectuant une simple requête SQL sur une petite table. À l'inverse, vous en aurez besoin sur une page

nécessitant des calculs lourds comme de nombreuses requêtes SQL ou des créations graphiques.

## Outils de cache Open Source

Il existe de nombreux outils de cache, pour la plupart équivalents dans leurs fonctionnalités. `Pear::Cache_Lite` est par exemple un outil simple destiné à mettre en cache des pages complètes. `Pear::Cache` est un système plus générique permettant de mettre en cache des résultats de requêtes SQL ou toute autre information.

Vous trouverez ci-après une présentation rapide de ces deux outils. Nous reviendrons de façon détaillée sur leur utilisation en fin de chapitre.

- `Pear::Cache`

`Pear::Cache` est la solution de Pear. Il s'agit d'un cache générique avec des conteneurs pour tout types d'objets (il est ainsi possible de gérer le cache de requêtes SQL, d'images, et bien sûr de pages HTML). Adresse du projet : <http://pear.php.net/package-info.php?packid=40>

- `Pear::Cache_Lite`

C'est la solution légère de Pear pour la gestion du cache. La bibliothèque ne gère que les pages HTML et est extrêmement légère, donc performante. Adresse du projet : <http://pear.php.net/package-info.php?packid=99>

## Mise en œuvre

Le but des systèmes de cache est d'éviter de recalculer plusieurs fois un même contenu.

C'est valable par exemple pour :

- une page HTML ;
- le chargement des paramètres de configuration du site ;
- les informations sur le visiteur en cours ;
- le résultat d'une recherche dans la base de données.

Lors du premier calcul, le résultat est stocké sur le serveur (par exemple dans un fichier temporaire). Aux prochaines requêtes, PHP ira récupérer le résultat sauvegardé, ce qui évitera un nouveau calcul.

## Les caches globaux

Ce qu'on appellera un cache global est un ensemble d'informations partagées par tous les scripts de votre serveur. Ces informations sont identiques pour tous les utilisateurs pendant une durée définie. Parmi ces informations, on trouve :

- le calcul de pages web qui ne changent pas en permanence, par exemple une page d'actualité : la recalculer seulement à chaque nouvel article ou seulement de temps en temps peut paraître suffisant ;

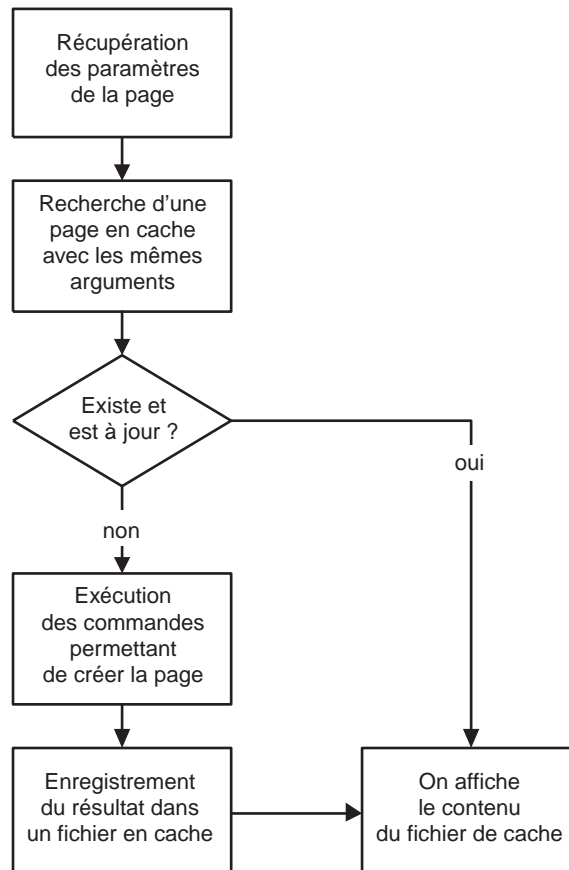
- l'intégration de composants distants : par exemple des fils d'actualité disponibles sur un autre site et qui seraient trop long à télécharger à chaque fois ;
- la création d'images ou de fichiers permanents comme un export PDF ou une archive ZIP.

### Cache d'une page HTML

Si vos pages ne sont pas totalement dynamiques, un système de cache, même rudimentaire, peut vous permettre d'augmenter de manière importante les capacités de votre serveur. En effet, le calcul de la page n'intervenant qu'une seule fois, l'activité principale du serveur web est alors l'envoi de pages HTML statiques (les pages mises en cache). Un gain de 500 % peut facilement être observé, que ce soit en vitesse ou en nombre de requêtes traitées par seconde. Le cache des pages web est même souvent indispensable si vous utilisez un système de templates, comme vu au chapitre précédent.

La figure 24-1 vous montre la structure de base d'un système de cache de pages web.

**Figure 24-1**  
*Principe d'un système de cache*



On peut imaginer un code dérivant du suivant :

```
<?php

// Récupération des paramètres et arguments
$nom_de_la_page = 'test.html' ;
$cache = 'cache/' . $nom_de_la_page ;
// On considère que la durée de validité est d'une heure
$expire = time() - 3600 ;

// Vérification de la présence de la page en cache
// et qu'elle n'est pas trop vieille
if( file_exists($cache) && filemtime($cache) > $expire) {
    // La fonction filemtime() renvoie la date de création du fichier
    // Le cache existe et est à jour, on en affiche le contenu
    readfile($cache) ;
    // On arrête l'exécution
    exit() ;
}

// Le cache n'existait pas ou n'était pas à jour
// On calcule donc la page
ob_start() ;
// ob_start() permet d'ouvrir un tampon ...
/* Ici on calcule la page et on la traite normalement */
$page = ob_get_contents() ;
ob_end_clean() ;

// On écrit le fichier de cache
file_put_contents( $cache, $page ) ;

// On affiche le résultat
echo $page ;
?>
```

Les fonctions `ob_start()`, `ob_get_contents()` et `ob_end_clean()` servent au traitement de tampon (voir chapitre 15 pour plus de détails). La première, `ob_start()`, demande à PHP de ne plus envoyer au client ce qui part normalement à l'affichage, mais de le stocker en interne. Un appel à `echo` ne renvoie rien à l'affichage, mais stocke son résultat dans le tampon. La seconde fonction récupère tout le contenu du tampon et, ici, le met dans la variable `$page`. La dernière fonction efface le contenu du tampon et en arrête le fonctionnement pour revenir à une sortie normale.

Grâce à ces trois fonctions, il est possible de garder un comportement classique pour le calcul de la page et d'insérer les quelques lignes de gestion de cache en haut et en bas des pages sans s'occuper du contenu.

### Accès concurrents au fichier de cache

Dans l'exemple précédent comme dans les suivants, nous ne traitons pas des possibilités d'accès concurrents à un fichier de cache. Si deux accès arrivent simultanément, il est possible que les deux accèdent en écriture au même moment et que le fichier de cache soit corrompu ; ou que le deuxième accède en lecture au fichier de cache en train d'être écrit, résultant en un affichage partiel chez le client.

Dans une mise en œuvre en production il vous faudra prendre en compte une solution à ces problèmes. Deux possibilités s'offrent à vous.

Vous pouvez bloquer tout accès au fichier de cache pendant l'écriture. Dans ce cas, reportez-vous à la fonction `flock()` ou aux fonctions d'utilisation des sémaphores.

Une autre solution est de garantir que la modification du fichier de cache sera faite de façon atomique. Sur les architectures Unix, vous pouvez faire les écritures sur un fichier temporaire et utiliser la fonction `rename()` pour remplacer l'ancien. Les accès en lecture verront l'ancien fichier ou le nouveau mais toujours un fichier entier et valide.

Vous pourrez trouver plus d'informations sur les accès concurrents aux fichiers au chapitre 13. D'autres méthodes restent possibles, par exemple la programmation par sémaphores, mais sont hors du cadre de ce chapitre.

## Cache de fichiers de différents types

La gestion d'autres données en cache peut être gérée de manière similaire. Nous avons vu au chapitre 20 concernant SimpleXML comment traiter un fichier RSS. Nous avons cependant utilisé une stratégie consommatrice en ressources puisque nous faisons appel à chaque fois au fichier distant. Une solution alternative et conseillée consiste à utiliser un cache.

### Information

L'utilisation d'un cache pour la lecture du fil de *news* du site `php.net` a été demandée par les membres de l'équipe de gestion du site. On imagine la charge qui s'appliquerait sans cela.

Ainsi par exemple, l'intégration de fichiers distants pourrait se faire avec le code suivant :

```
<?php
// Récupération des arguments
$url = 'http://www.php.net/news.rss' ;
$cache = 'php_net.rss' ;
$media = 'application/rss+xml' ;
$expire = time() - 3600 ;
// Vérification de la présence de la page en cache
// et qu'elle n'est pas trop vieille
if (!file_exists($cache) || !filemtime($cache)>$expire) {
    copy($url, $cache) ;
}

header("Content-Type: $media") ;
readfile($cache) ;
?>
```

## Cache de configuration

Les pages HTML et les fichiers ne sont pas les seules données qui peuvent bénéficier d'un système de cache. Les données qui nécessitent des temps de calcul importants ou même les données simples à calculer mais fréquemment demandées peuvent trouver une utilité à un cache.

Ainsi, une application avec énormément de paramètres de configuration à interpréter pourrait gagner à utiliser un cache. Dans cet exemple, il serait possible d'importer les configurations de divers endroits comme une base de données, des variables d'environnement, des fichiers XML et des fichiers .ini. Une telle importation ne prend pas un temps exagéré mais devra être faite à chaque requête sur le serveur web, et au final occupera des ressources non négligeables sur le serveur.

L'idéal pour les performances serait de créer un fichier unique contenant les différentes valeurs de configuration après interprétation. On pourrait alors utiliser la procédure suivante :

1. On vérifie la date de dernière modification du fichier de cache.
2. Si cette date est plus vieille qu'un temps défini, alors on lit et interprète les différentes sources, on les fusionne, puis on écrit un fichier de configuration unique.
3. On interprète le fichier de cache avec la fonction `include()`.

### Pourquoi utiliser l'extension PHP pour un fichier de configuration ?

Le choix d'un fichier PHP au lieu d'un format non exécutable (ex. : html) peut surprendre, mais il y a deux avantages directs.

Tout d'abord, l'éventuelle perte de performance due à l'exécution du fichier PHP ne sera pas notable. Au contraire, avec l'utilisation de générateurs de code intermédiaire comme les logiciels Zend Cache ou APC, la phase d'interprétation sera faite une seule fois et on peut voir un gain en performance pour de nombreuses variables.

De plus, les fichiers de configuration sont ceux qui contiennent généralement les mots de passe des différentes ressources et accès. Utiliser un fichier PHP permet de réduire les risques si jamais une faille dans votre serveur web permet d'appeler directement le fichier de cache (il sera interprété et non téléchargé donc aucun mot de passe ne sera divulgué) ou si votre application a une faille permettant de lui faire inclure ce fichier (auquel cas, le seul effet sera la redéfinition des variables et non un affichage). Il ne s'agit pas d'une protection en soi mais d'une garantie supplémentaire au cas où.

## Cache des données utilisateur

Par données utilisateur, on entend les informations qui ne concernent qu'un utilisateur. Parmi les plus courantes, on trouve l'authentification, les préférences d'affichage du site, la reconnaissance du navigateur, le calcul de pages personnelles, les informations déjà tapées dans les formulaires, etc.

## Cache par session

Généralement, ces informations n'ont d'intérêt à être calculées qu'une fois par session. Si vous avez besoin de calculer des informations sur l'utilisateur, même peu souvent, il peut être intéressant de mettre le résultat du calcul en session pour éviter de le refaire à chaque fois. En évitant de recalculer ce genre de paramètres, vous soulagerez votre serveur de nombreux calculs.

Il suffit alors dans ce cas de récupérer les données dans la session en cours (superglobale `$_SESSION[]`, voir le chapitre 11 sur les sessions pour plus de détails) et de ne les calculer que si elles n'y sont pas encore.

Cette méthode peut être utilisée pour les préférences de l'utilisateur et pour toutes les données qui ne sont pas amenées à changer au cours d'une session.

### Données de grande taille

Certaines données ne devraient pas être mises dans le fichier de session. Nous avons vu au chapitre 11 que lors d'un appel à une page utilisant une session, tout le fichier est lu pour être mis en mémoire. De ce fait, stocker en session une image propre à l'utilisateur est une solution coûteuse en temps et en mémoire, donc a priori pas une bonne solution. Cela ralentirait énormément l'interprétation du fichier de session, uniquement pour quelques affichages.

Généralement, on utilise un répertoire temporaire. Lors du calcul d'une image propre à un utilisateur, on la stocke dans le répertoire temporaire en lui donnant l'identifiant de session comme préfixe.

Pour ne pas stocker sur le disque trop de vieux fichiers qui ne servent plus, il reste à implémenter un ramasse-miettes (robot qui lit régulièrement le répertoire temporaire et le répertoire de session pour effacer ce qui est périmé). S'il trouve un fichier avec un préfixe qui ne correspond pas à une session existante (il s'agit donc d'une session expirée), il l'efface. Faire tourner ce robot quelques fois dans la journée peut suffire si vous n'avez pas de problèmes d'espace disque.

Si vos utilisateurs sont authentifiés et peu nombreux, il est même possible de stocker les fichiers de cache de manière permanente dans un répertoire au nom de l'utilisateur.

## Les caches HTTP

Jusqu'à présent, nous avons parlé de cache côté serveur afin d'éviter de recalculer le contenu. Le visiteur, lui, n'en profite que par la rapidité de réponse du serveur.

Pourtant, le visiteur a lui aussi un système de cache, intégré à son navigateur. L'idée est alors de simplement renseigner le visiteur en lui disant qu'une page n'a pas changé depuis sa dernière visite. Il n'est alors plus nécessaire de recalculer la page.

Le protocole HTTP, utilisé pour les pages web, implémente déjà un tel mécanisme pris en charge par tous les navigateurs. Grâce à l'utilisation du système de cache HTTP, il est



possible à peu de coût d'épargner du temps de calcul mais aussi de la bande passante. Ce cache est d'autant plus intéressant qu'il peut être géré par un proxy, intermédiaire entre votre serveur et le visiteur ; dans ce cas, tous les visiteurs situés derrière ce proxy profiteront d'une même page sans nécessiter une quelconque charge du serveur web.

## Dates de mises à jour des fichiers

Le protocole HTTP 1.0 utilise les dates de dernière modification d'une page pour gérer les caches.

Son successeur, HTTP 1.1, permet une autre procédure basée sur des identifiants. Cependant, comme tous les clients HTTP 1.1 gèrent aussi les dates de modification, nous nous contenterons de cette méthode.

À la première requête, le serveur renvoie avec la page un en-tête spécifiant la date de dernière modification du contenu de la page (`Last-Modified: Thu, 24 Sep 2003 00:57:47 GMT`). Vous pouvez vous servir directement de la date de modification des données sources ou, si vous utilisez un cache serveur comme vu précédemment, de la date de modification du fichier de cache.

```
$modif = gmdate('D, d M Y H:i:s', filemtime($fichier_source)) ;  
header("Last-Modified: $modif GMT");
```

Si vous n'avez aucun moyen de déterminer la date de modification du contenu, vous pouvez envoyer la date actuelle.

### Attention

Si vous avez un fichier qui en appelle d'autres, par exemple via des inclusions, il vous faut vérifier la date de mise à jour de chaque fichier utilisé et renvoyer la plus récente.

Lors des requêtes suivantes, le navigateur envoie avec la requête une condition équivalente à « seulement si le contenu a été mis à jour depuis la dernière fois » (`If-Modified-Since: Thu, 24 Apr 2003 00:57:47 GMT`).

Si la page a été modifiée, alors le serveur répond normalement en redonnant une nouvelle date de modification. Sinon, il renvoie un en-tête pour dire au navigateur que la page n'a pas été modifiée (`HTTP/1.x 304 Not Modified`) et ne renvoie pas le contenu de la page (dans le cas d'une page dynamique, il n'est alors pas nécessaire de calculer ce contenu, ce qui est autant de gagné pour le serveur). Si vous ne connaissez pas la date de modification du contenu, vous pouvez par exemple déterminer arbitrairement que la page n'a pas changé si elle date de moins d'une heure.

```
header('Not Modified', TRUE, 304);
```

Dans le cas d'une page qui n'a pas été mise à jour, le navigateur reprend alors la dernière page qu'il avait reçue et l'affiche directement sans avoir à la télécharger. Il y a un double gain pour le serveur puisqu'il y a moins de calculs et que la bande passante est moins

utilisée. On constatera également un gain réel pour le visiteur qui aura un affichage instantané.

Voici une implémentation rapide d'un tel système. Ici, `$fichier` est un fichier de cache sur le serveur, créé comme vu précédemment.

Tout d'abord, on initialise les dates à utiliser ; la première est la référence à envoyer au navigateur (date actuelle), la seconde est la date de dernière modification du fichier à envoyer.

```
/** Initialisation :  
$date_serveur = time() ;  
$date_modif = filemtime($fichier) ;
```

Par la suite, on envoie les différents en-têtes : date actuelle et date de dernière modification.

```
/** Envoi des en-têtes permanents  
// Date du serveur  
header('Date: '. gmdate("D, d M Y H:i:s", $date_serveur) .' GMT');  
// Date de dernière modification  
$modif = gmdate('D, d M Y H:i:s', $date_modif) ;  
header("Last-Modified: $modif GMT");
```

Et enfin, on vérifie si le navigateur avait fait une requête conditionnelle (présence de l'en-tête `If-Modified-Since` qui indique qu'il est déjà venu).

```
/** On vérifie si le contenu a changé  
$if=substr(@$_SERVER['HTTP_IF_MODIFIED_SINCE'],0,29) ;
```

Si c'est le cas et si le contenu n'a pas changé depuis (date de modification plus ancienne que celle reçue dans la requête), alors on lui signale que le contenu n'a pas changé (en-tête `304`).

```
if ($if!='' && strtotime($if)>=$date_modif) {  
    // Le contenu n'a pas changé  
    header('Not Modified', TRUE, 304);  
}
```

Sinon, on lui envoie le contenu de la page.

```
elseif($_SERVER['REQUEST_METHOD'] != 'HEAD') {  
    // Le contenu a changé  
    readfile($fichier) ;  
}
```

## Utilisation des serveurs proxies

Le système vu précédemment est propre à chaque navigateur ; deux utilisateurs différents feront donc toujours deux requêtes consommatrices de ressources au serveur.

Cependant, entre votre serveur et le visiteur peuvent se trouver des serveurs mandataires (proxies). C'est le cas par exemple dans les entreprises, écoles, bibliothèques et la plupart des collectivités, privées ou publiques. Certains fournisseurs d'accès Internet proposent

aussi des proxies à leurs usagers. Ces serveurs intermédiaires peuvent gérer des caches en se basant sur le protocole HTTP ; nous avons donc tout intérêt à en profiter pour économiser encore quelques ressources.

Avec quelques en-têtes HTTP supplémentaires, il est possible de contrôler le fonctionnement des proxies. Ils ont juste besoin de quelques instructions, par exemple le fait de savoir si une page est à usage privé ou public.

- À usage privé, la page sera réutilisée uniquement pour le client ayant demandé la page à l'origine.
- À usage public, le proxy pourra resservir le contenu à tous les visiteurs faisant la même requête.

Les codes suivants envoient une notification pour un usage public :

```
header('Cache-Control: Public, must-revalidate');  
header('Pragma: public');
```

Vous pouvez remplacer `public` par `private` pour un usage privé, ou par `no-cache` si la page ne doit absolument pas être mise en cache (par exemple pour un contenu qui change tout le temps). La directive `must-revalidate` demande aux clients et aux intermédiaires de se conformer strictement à cette déclaration et de ne pas l'interpréter librement.

Par moments, le contenu de la page est un contenu public, mais il dépend tout de même des paramètres de l'utilisateur. Par exemple, si vous envoyez un contenu compressé, vous ne voudrez le faire que pour les clients qui acceptent cette fonctionnalité. Vous pouvez aussi vouloir envoyer un contenu différent suivant la version du navigateur utilisé, les formats reconnus par le client, etc. Il est possible d'affiner le cache en spécifiant que le cache public dépend de la valeur d'un ou plusieurs en-tête(s) de la requête utilisateur. Il vous suffit alors d'envoyer un en-tête `Vary:` en réponse, avec en face la liste des dépendances.

Ici, nous déclarons que le contenu est public mais dépend de la version du navigateur utilisé (en-tête `User-Agent` dans la requête) :

```
header('Vary: User-Agent');  
header('Cache-Control: Public, must-revalidate');  
header('Pragma: public');
```

#### Attention

Si vous utilisez le cache du navigateur et des proxies, certaines requêtes ne seront pas envoyées au serveur. Les pages qui accomplissent des actions ou collectent des statistiques ne pourront plus agir. Le fait que la page soit mise en cache par les proxies doit être pris en compte lors des calculs de visites.

### Mise en place d'un proxy sur le serveur

Les serveurs proxy, bien exploités, peuvent soulager énormément votre serveur web. Actuellement c'est toutefois un gain qui se fait de moins en moins sentir car les particuliers passent désormais rarement par de telles passerelles. De plus, un proxy ne mutualise les

requêtes que d'un faible nombre d'utilisateurs (une entreprise ou une bibliothèque par exemple).

Afin de pallier ce défaut, certains cherchent à forcer le passage par un proxy en installant un directement sur le serveur ; le logiciel Squid est souvent utilisé à cet effet. Ce proxy servira alors de filtre entre les visiteurs et vos scripts, gérant automatiquement certains caches. Vous n'économiserez pas de bande passante avec ce procédé mais le processeur sera nettement soulagé : une seule exécution de script pourra servir plusieurs requêtes HTTP.

### Utiliser la date d'expiration

Dans tout ce qui précède, le visiteur ou le proxy continue à faire des appels au serveur afin de vérifier que sa page est à jour. Afin d'épargner encore plus de traitements au serveur, il est possible d'informer le système de cache qu'il n'est pas nécessaire de vérifier la mise à jour pendant un certain temps. Ainsi, le serveur ne recevrait même plus de requêtes ; c'est l'aboutissement du concept de cache. Un tel système est très utile pour les pages ayant peu de probabilités de changer, ou qui sont appelées fréquemment.

Pour implémenter ce système, il suffit d'envoyer une date d'expiration au client en même temps que la page. Tous les accès futurs à la même page jusqu'à cette date feront appel directement à la page stockée en cache sur le navigateur, et non au serveur. Les dates pertinentes peuvent varier d'une heure pour une page d'accueil à plusieurs jours pour des images ou des fichiers qui ne changent pratiquement pas, mais qui, pour une raison ou une autre, sont gérés par PHP.

L'en-tête HTTP à utiliser est `Expires:` et prend en paramètre une date au même format que celui des dates de modification plus haut :

```
$expires = time() + 3600 // dans une heure
header('Expires: ' . gmdate("D, d M Y H:i:s", $expires) . ' GMT');
```

#### Attention

Étant donné qu'avant l'expiration aucune requête n'est faite au serveur, une fois une date d'expiration donnée vous n'aurez aucun moyen de signaler au client qu'il y a eu une mise à jour du contenu correspondant. La seule option pour que l'affichage du client reflète le nouveau contenu est d'attendre la fin de la période d'expiration.

## Mise à jour du cache

Nous avons vu comment créer un système de cache et l'utiliser. Cependant, il nous faut encore connaître un élément important : comment détecter une mise à jour des données pour recalculer le contenu des données.

## Détection de la modification

La mise à jour lors de la détection d'une modification est une procédure intuitive. Elle est utilisable dans de nombreux cas.

- Quand les données sont présentes sous forme de fichiers, il suffit alors de récupérer la date de modification des fichiers.
- Quand les données sont en base de données avec un champ date mis à jour à chaque changement, il suffit alors de récupérer les données si et seulement si la date est plus récente que celle du fichier mis en cache.

Vous devez utiliser cette méthode si vous pouvez déterminer rapidement et facilement la dernière date de mise à jour de vos données.

## Temps de validité

Malheureusement, il est fréquent qu'on ne puisse pas connaître la date de modification des données, ou avec un temps de calcul trop long. C'est par exemple le cas :

- Quand les données sont sur des serveurs distants ;
- Quand mettre à jour des dates de modification dans la base de données est très coûteux ;
- Quand plusieurs données partagent un même conteneur et que la date de modification du conteneur ne permet pas de déterminer celle du contenu (par exemple, si plusieurs informations partagent le même fichier, on ne connaît que la date de modification du conteneur, le fichier, pas la date de modification d'une information précise).

Dans ces cas, il peut être pertinent de raisonner avec un temps de validité. On considère arbitrairement que le contenu est valide pendant un certain temps, par exemple cinq minutes. Il suffit alors à chaque fois de vérifier si le cache est âgé de plus de cinq minutes, et si oui de le reconstruire.

Cette méthode est toujours délicate à gérer : un temps de validité trop long peut être dommageable à la publication de vos données (quelqu'un pourrait avoir un contenu trop vieux). Un temps de validité trop court pourrait conduire à une reconstruction du cache trop importante. Les problèmes d'accès concurrents avec une reconstruction trop fréquente peuvent même amener des performances plus faibles qu'un système sans cache ; voir l'encadré en début de ce chapitre à ce sujet.

### Astuce

Afin de diminuer le nombre d'accès concurrents en écriture sur le fichier de cache, une méthode habituelle est d'inclure une petite partie aléatoire dans la période de validité lors du calcul. Ainsi, si deux requêtes se font presque simultanément, il est probable qu'une seule reconstruise le cache et que l'autre le lise directement.

## Sites semi-statiques

L'automatisation de la création de pages statiques est probablement la plus performante pour les sites qui ne changent pas de contenu trop fréquemment et les sites dont le contenu mis à jour est assez réduit. C'est par exemple le cas des sites de dépêches ou d'articles.

Même si un article est modifié toutes les cinq minutes, il est plus simple de faire reconstruire le cache par l'outil d'administration lors de la modification que de faire une détection à chaque requête.

Le cache du système de templates Templeet, vu dans le chapitre précédent, implémente un tel système de manière très efficace.

- Un script intercepte toutes les requêtes vers des pages inexistantes. S'il s'agit de pages qui devraient exister (par exemple la liste des articles) alors elles sont construites et insérées au bon endroit dans la hiérarchie du serveur web.
- Lors des visites suivantes, le serveur web verra que les pages sont créées et donc n'exécutera pas le script. Pour le serveur web, et tant que le cache ne sera pas supprimé, les performances seront celles d'un site avec des pages statiques, sans l'occupation serveur d'un script PHP.

Quand un nouvel article est inséré via l'outil d'administration, la page de la liste des articles est supprimée afin d'être mise à jour à la prochaine requête.

Ce système est de loin le plus performant mais peut se révéler difficile à maintenir. En effet, il faut absolument passer par une interface pour toutes les modifications, sinon le cache ne sera pas reconstruit et les modifications ne seront pas prises en compte. La méthode n'est donc envisageable que dans le cadre d'un site de gestion de contenu complet, où les façons de mettre à jour un contenu sont réduites.

Un palliatif à ce défaut peut être l'utilisation d'un robot qui détruit les pages trop vieilles. On se retrouverait alors avec les défauts d'un système avec temps de validité.

Une variante à cette méthode est d'avoir un générateur complet dans l'administration. À chaque mise à jour, on recrée un site complètement statique. C'était une méthode très utilisée il y a quelque temps, quand les puissances des serveurs étaient plus faibles.

## Pear::Cache

Pear::Cache est une solution très générique, faite pour être personnalisable selon vos besoins. Il s'agit d'une bibliothèque relativement bas niveau, facilement extensible et spécialisable. Le but est d'avoir une batterie de classes dérivées pour des applications spécifiques (cache de la page résultat, d'une requête SQL, d'une image, etc.) qui se basent sur la classe générique.

Par défaut, vous avez d'ailleurs accès à quelques types de caches (image, page, page compressée, requête SQL, téléchargement de fichier distant) et quelques types de solutions de stockage (fichier, bases de données, mémoire partagée).

L'installation se fait via l'installateur Pear, il vous suffit de demander le paquetage Cache. Sous Unix, tapez `pear install Cache` en ligne de commande. Vous pouvez aussi la télécharger à partir de l'adresse <http://pear.php.net/package-info.php?pacid=40>.

## La classe générique

La classe générique vous laisse la possibilité d'utiliser du cache sur tout type de données. Malgré cela, elle reste trop bas niveau pour que son utilisation soit agréable. Nous vous recommandons d'utiliser au maximum les classes spécialisées, ou d'en créer une si aucune ne correspond à vos besoins.

La procédure d'utilisation est la suivante :

1. chargement de la bibliothèque ;
2. initialisation et définition du système de stockage ;
3. identification de l'information à utiliser ;
4. récupération de l'information du cache si elle y existe ;
5. création de l'information si le cache n'était pas à jour.

### Chargement de la bibliothèque

Pour charger la bibliothèque, il vous suffit de l'inclure. Si vous utilisez l'architecture Pear, vous devriez déjà avoir les fichiers dans un répertoire inscrit dans la directive `include_path` du `php.ini`.

```
<?php
require_once( 'Cache.php' );
```

### Initialisation et système de stockage

Nous pouvons maintenant initialiser un objet de cache. La classe générique s'appelle simplement `Cache`.

```
<?php
require_once( 'Cache.php' );
$cache = new Cache('file', array('cache_dir' => 'cache/')) ;
```

Dans notre exemple, nous avons fourni deux arguments : le premier est le type de stockage voulu pour les données en cache, le deuxième est un tableau contenant les paramètres nécessaires au système de stockage. Le cas le plus fréquent sera probablement un stockage en fichiers ; dans ce cas, le seul paramètre indispensable est `cache_dir`, le répertoire où doivent être stockés les fichiers.

Vous pourrez trouver d'autres types de conteneurs : `db` utilise la couche d'abstraction base de données de Pear, avec `dsn` (paramètres de connexion utilisés par `pear::db`) et `cache_table` (nom de la table de cache) comme paramètres. `shm` utilise la mémoire partagée, avec comme paramètres `shm_key`, `sem_key`, `shm_size`, `sem_perm`, `shm_perm` (clé pour la portion de mémoire partagée et le sémaphore, taille de la mémoire à utiliser, permissions pour le sémaphore et la mémoire allouée).

### Identification de l'information à utiliser

La classe générique n'implémente que les primitives bas niveau pour vous permettre de faire des classes dérivées plus spécialisées. C'est donc à vous de donner un nom ou un identifiant à la donnée que vous allez utiliser. Pour une page web, l'identifiant est probablement l'adresse de la page.

```
<?php
require_once( 'Cache.php' );
$cache = new Cache('file', array('cache_dir' => 'cache/') );
$id = $cache->generateID('identifiant123456');
```

L'identifiant est ce qui va différencier deux données en cache. Pour un site d'articles, on pourrait mettre le titre de l'article ou son numéro. Pour un site dynamique avec personnalisation par l'utilisateur, on pourrait ajouter le nom d'utilisateur à la chaîne pour différencier les contenus de chaque visiteur.

### Récupération de l'information du cache

Nous pouvons désormais essayer de récupérer la donnée à utiliser, si jamais elle est déjà présente en cache :

```
<?php
require_once( 'Cache.php' );
$cache = new Cache('file', array('cache_dir' => 'cache/') );
$id = $cache->generateID('identifiant123456');
if ($data = $cache->get($id)) {
    // La donnée est en cache
    echo $data ;
}
```

La méthode `get()` prend en paramètre un identifiant pour l'objet caché. Elle renvoie le contenu du cache s'il existe et est à jour, la valeur `FALSE` sinon.

Il nous suffit donc d'essayer de récupérer le contenu. Si la valeur de retour n'est pas `FALSE`, alors nous pouvons l'afficher directement. Sinon, c'est que le cache n'est pas à jour et nous devons le reconstruire.

### Régénération de l'information, cache non à jour

Si la valeur retournée plus haut est `FALSE`, il nous faut alors recalculer le contenu à mettre en cache, le sauvegarder pour une utilisation ultérieure et ensuite l'afficher.

```
<?php
```



```
require_once( 'Cache.php' );
$cache = new Cache('file', array('cache_dir' => 'cache/') );
$id = $cache->generateID('identifiant123456');
if ($data = $cache->get($id)) {
    // La donnée est en cache
    echo $data ;
}else {
    // Pas d'utilisation du cache
    // À vous d'implémenter la façon de créer vos données
    $data = calcul_de_la_donnee() ;
    // Une fois créée, on sauvegarde l'information en cache
    $cache->save($id, $data);
    echo $data ;
}
```

### Autres méthodes utiles

Trois autres méthodes de l'objet générique peuvent être intéressantes : la méthode `isCached($id)` renvoie vrai si l'information avec l'identifiant `$id` passé en paramètre est actuellement en cache. La méthode précédente renvoie la valeur TRUE même pour une donnée expirée, mais vous pouvez tester l'expiration de la donnée avec la méthode `isExpired($id)`. Enfin, `delete($id)` permet de détruire le contenu du cache.

## Classe pour le Cache HTML

La classe précédente n'est pas forcément très pratique d'utilisation pour gérer une simple page à mettre en cache. La classe `Cache_Output` est une spécialisation de la classe `Cache` qui permet de gérer ce cas précis.

Le principe global reste le même, mais quatre méthodes supplémentaires sont disponibles afin de capturer directement le flux de sortie pour le mettre en cache. Grâce à un tel système, nous pouvons faire marcher le cache avec les scripts existants sans trop de modification :

- `Start()` démarre la capture.
- `End()` arrête la capture et enregistre le contenu dans le cache. Cette méthode prend en paramètre un temps de validité pour le cache en secondes.
- `EndPrint()` arrête la capture, enregistre le contenu dans le cache puis l'envoie vers la sortie. Cette méthode prend en paramètre un temps de validité en secondes pour le cache.
- `EndGet()` arrête la capture et renvoie le contenu dans une variable, ne sauvegarde pas le cache.

Ainsi est-il possible d'interagir avec des fonctions d'affichage sans les modifier :

```
<?php
// Initialisation
require_once( 'Cache/Output.php' ) ;
```

```
$cache = new Cache_Output('file', array('cache_dir' => 'cache/')) ;
$id = $cache->generateID('identifiant123456');

// On récupère la page si elle existe déjà
if ($page = $cache->get($id)) {
    // Si la donnée est en cache,
    // on l'affiche
    echo $page ;
    // On arrête l'exécution
    exit;
}

// Le cache est vide ou expiré, on recrée la page
$cache->start($id) ;

/* Ici on calcule la page et on la traite normalement */

// On enregistre la page et on l'affiche
$cache->EndPrint(60) ;
?>
```

Vous pouvez utiliser le cache sur toute une page comme dans l'exemple, ou seulement sur une partie. Dans le premier cas, vous pouvez aussi utiliser la classe `Cache_OuputCompression` qui permet d'envoyer un contenu compressé aux navigateurs.

#### Note

Si la compression est activée, les caches sont stockés dans leur version compressée pour éviter de refaire la compression à chaque fois. En contrepartie, si le navigateur ne comprend pas la compression des pages par gzip, le système décompressera la page en temps réel, ce qui peut se révéler aussi long que la création de la page elle-même. Si vos clients ne comprennent pas la compression (les navigateurs classiques comme Netscape, Internet Explorer, Mozilla ou Opera la comprennent), alors il est probablement plus adapté de la désactiver.

## Autres caches

De même que la spécialisation dédiée aux pages HTML, vous pouvez trouver des classes de cache adaptées à différents types de données.

La classe `Cache_Graphics` permet de mettre en cache des images créées à la volée, `Cache_Function` le résultat de fonctions longues à calculer, `Cache_DB` le résultat de requêtes SQL et `Cache_HTTP_Request` l'utilisation de fichiers distants.

La démarche de toutes ces classes est similaire à la classe générique ; elles ne contiennent que quelques méthodes supplémentaires pour gérer automatiquement les cas particuliers de chaque application.

## Pear::Cache\_Lite

Les solutions de Pear::Cache sont assez génériques pour pouvoir convenir à la majorité des utilisations avec un minimum de modifications. Vous pouvez modifier le support de stockage comme gérer des types de données complexes. Pourtant, cette surenchère de modularité a un coût important en performance. La moindre spécialisation nécessite l'interprétation de quatre fichiers de classe au minimum.

Dans le cas du cache de pages web complètes, l'utilisation de tous ces fichiers et abstractions ajoute une charge significative, alors que l'opération est censée être très simple et peu coûteuse en temps processeur.

Pear::Cache\_Lite est là pour corriger ce problème. Cette bibliothèque n'implémente que le cache de pages web complètes mais le fait dans une optique de hautes performances. Dans ce cadre, il s'agit d'une des seules applications Pear dont la classe n'hérite pas de la classe générique Pear. Seul l'indispensable est chargé par défaut ; tout le reste, comme la gestion d'erreur, ne l'est qu'en cas de problèmes.

Si les fonctionnalités de cette bibliothèque vous suffisent, elle est à préférer à Pear::Cache.

### Utilisation

L'installation se fait via l'outil de gestion Pear, soit par téléchargement sur <http://pear.php.net/>.

L'utilisation est similaire à celle de Pear::Cache, mis à part quelques paramètres.

### Initialisation

```
<?php
// Initialisation
require_once('Cache/Lite.php');
```

Il n'existe plus de fonction pour créer un identifiant, c'est à vous de le déterminer directement.

```
// Création manuelle de l'identifiant
$id = 'identifiant465878786';
```

Quelques paramètres peuvent être fournis à l'initialisation sous la forme d'un tableau associatif :

- `cacheDir` : répertoire où sauvegarder les fichiers de cache ;
- `caching` : booléen, à faux pour désactiver le cache ;
- `lifeTime` : temps de validité du cache en secondes ;
- `fileLocking` : booléen, vrai pour activer le verrou des fichiers (inopérant sur NFS ou un système multi-threads comme IIS) ;

- `writeControl` et `readControl` : des procédures de vérification d'intégrité pour le cache, vrai pour activer ;
- `readControlType` : type de contrôle de lecture si `readControl` est activé (crc32, md5, ou strlen).

```
$options = array(
    'cacheDir' => 'repertoire/temporaire/',
    'lifeTime' => 3600
);
$Cache_Lite = new Cache_Lite($options);
```

### Affichage

Le reste est identique à l'utilisation de `Pear::Cache` :

```
if ($data = $Cache_Lite->get($id)) {
    // Si la donnée est en cache, on l'affiche
    echo $data
} else {
    // Si la donnée est recalculée,
    /* On la calcule et on la met dans la variable $data :
       $data = ...
    */
    $Cache_Lite->save($data);
}
```

### Spécialisations

De même que `Pear::Cache`, `Pear::Cache_Lite` fournit quelques classes spécialisées pour gérer les données.

Ainsi, `Cache_Lite_Output` permet de gérer les caches de pages HTML ; son fonctionnement est similaire à celui de `Cache_Output` :

```
require_once('Cache/Lite/Output.php');
$options = array(
    'cacheDir' => 'cache/',
    'lifeTime' => 60
);
$cache = new Cache_Lite_Output($options);
if (!$cache->start($_SERVER['REQUEST_URI'])) {
    // Reconstruction de la page
    /* fonction_affichage() ; */
    // Enregistrement
    $cache->end();
}
```

`start()` envoie en sortie le contenu du cache s'il est valide, sinon démarre le début de la capture. `end()` arrête la capture et l'envoie sur la sortie après enregistrement.

## Étude de cas

### Cache pour un site d'actualité

#### Contexte

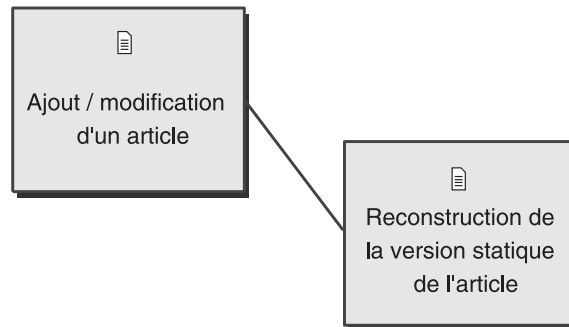
Vous réalisez un site d'actualité administrable à distance par une interface en PHP. Les nouvelles du site sont insérées principalement via cette interface. Le rythme des nouveaux articles est relativement faible, de l'ordre d'un article toutes les heures. Le trafic sur votre site est important et les temps de création des pages affichant les articles peuvent parfois être assez longs.

#### Architecture du système

Étant donné le système d'administration à distance et compte tenu du fait que ce soit le seul moyen d'ajouter de l'information, on peut se reposer sur un système créant les fichiers de cache à chaque nouvel ajout d'article (voir section site semi-statique).

Figure 24-2

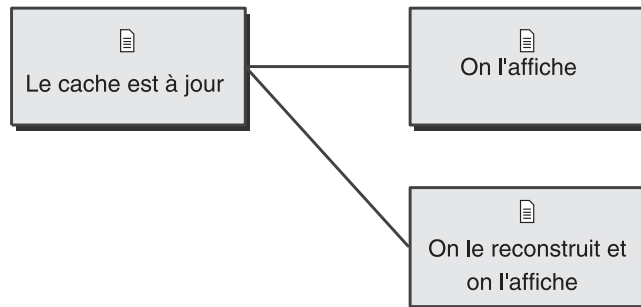
*Ajout d'un article*



De plus, pour assurer des mises à jour, on peut décider d'une durée de vie maximale d'une heure par cache.

Figure 24-3

*Affichage d'une page*



## Réalisation

### Après un ajout ou une modification

On fabrique le premier cache d'un article lorsque celui-ci est créé via l'outil d'administration. Il faut donc proposer l'affichage comme pour une consultation et enregistrer le résultat dans un fichier de cache.

```
<?php
ob_start() ;
// ob_start() permet d'ouvrir un tampon
/* On calcule la page et on la traite normalement */
$page = ob_get_contents() ;
ob_end_clean() ;

// On écrit le fichier de cache
file_put_contents( $cache, $page ) ;
?>
```

### Lors de la consultation

Lors de la consultation d'un article, on aura besoin de savoir s'il faut ou non utiliser un cache. Pour cela, on va créer une fonction qui nous l'indiquera. Cette fonction prendra en paramètre le chemin d'accès au fichier de cache de l'article et la durée de vie du fichier de cache.

```
<?php
function utiliser_cache($chemin_cache, $delais) {
    // A priori on utilise un cache
    $use_cache = true;
    // Si le cache existe, on vérifie sa date
    if (file_exists($chemin_cache)) {
        $t = filemtime($chemin_cache);
        $ledelais = time() - $t;
        $use_cache &= ($ledelais < $delais AND $ledelais >= 0);
    } else
        $use_cache = false;
    // Recalcul obligatoire
    $use_cache &= ($_GET['recalcul'] != 'oui');
    $use_cache &= empty($_POST);

    // On ne recalcule pas pour
    // les moteurs de recherche, proxies...
    if ($_SERVER['REQUEST_METHOD'] == 'HEAD')
        $use_cache = true;
    return $use_cache;
}
```

Ainsi, avec cette fonction, il suffira de l'interroger pour savoir si on lit le fichier de cache ou si on le crée.

Dans notre exemple, nous mettrons toutes les pages de cache dans un répertoire nommé cache.

```
// On récupère les informations sur le document demandé.
$page = 'test.html';
$chemin = './cache/' . $page;
$delai = 3600 ;
if( utiliser_cache($chemin,$delai)) {
    readfile($chemin) ;
    exit() ;
}

// Le cache n'existait pas ou n'était pas à jour
// On calcule donc la page
ob_start() ;
/* Ici on calcule la page et on la traite normalement */
$page = ob_get_contents() ;
ob_end_clean() ;

// On écrit le fichier de cache
file_put_contents( $chemin, $page ) ;

// On affiche le résultat
echo $page ;
?>
```

# Gestion des images

---

De nos jours, les applications professionnelles utilisent en majorité les images de façon très poussée. Les images sont omniprésentes, que ce soit pour agrémenter un texte, résumer des chiffres ou présenter un concept. Internet n'échappe pas à cette règle, bien au contraire. De par son aspect multimédia, il utilise encore plus le visuel.

La création d'images à la main est cependant extrêmement coûteuse en temps et en compétences. On imagine facilement les gains offerts par un système de création automatique d'images intégré à vos applications. PHP vous permet d'implémenter ces créations dynamiques et offre de nombreuses fonctionnalités. Il existe de plus des bibliothèques externes d'excellente facture qui vous permettront des manipulations très simples pour des résultats visuels complets.

## Utilité de la gestion d'images

PHP utilise la bibliothèque GD pour manipuler les images. Il s'agit d'une extension permettant de créer et modifier assez facilement des fichiers images (JPEG, PNG, WBMP, etc.). Il vous sera par exemple possible avec PHP de l'utiliser pour créer des graphiques dépendant de vos données stockées dans un SGBD. Il est aussi possible d'accéder aux informations IPTC (<http://www.iptc.org>) stockées dans les images reconnaissant cette norme.

**IPTC (International Press Telecommunications Council)**

IPTC est l'organisme de normalisation du secteur de la presse. Il a défini une norme permettant d'inclure des informations de copyright dans des fichiers images. Cette méthode est très utilisée par les journalistes et autres photographes.



Les manipulations d'images sont toutefois des procédés très coûteux en ressources mémoire et processeur. Ces fonctions doivent donc être utilisées à bon escient. L'utilisation d'un système de cache peut être opportun.

Voici quelques cas d'utilisation :

- élaboration de statistiques (diagrammes à barre, graphiques sectoriels, camemberts 3D, etc.) ;
- redimensionnement d'images (création de miniatures, homogénéisation des tailles, etc.) ;
- superposition d'images (pour signer des images) ;
- création de menu en mode image de façon dynamique ;
- création de compteurs de visites ;
- transformation d'images en niveaux de gris ;
- utilisation de filtres sur les images (élimination du bruit, flou gaussien, etc.).

## Prérequis techniques

PHP permet de créer dynamiquement des images grâce à la bibliothèque GD.

Pour utiliser ces fonctions, il faut donc que PHP soit installé avec l'extension GD. Sur Microsoft Windows, il faut décommenter la ligne `extension=php_gd2.dll` dans le fichier `php.ini`. Sur les autres systèmes comme Linux, il vous faut passer le paramètre `--with-gd` lors de la compilation. Pour plus de détails, rendez-vous dans la partie installation de PHP au chapitre 2.

## Initialisation et utilisation

Quatre étapes principales se distinguent quant à la fabrication d'une image :

1. création du modèle d'image sur lequel on va travailler (allocation des ressources mémoire, chargement de l'image originelle) ;
2. travail sur le modèle (ajouts, modification de formes, etc.) ;
3. fabrication de l'image finale (envoi au navigateur ou sauvegarde en tant que fichier sur le disque) ;
4. effacement des données de la mémoire.

### *La création du modèle de l'image*

Comme dans la vie réelle, pour dessiner, il faut disposer d'une feuille blanche ou d'un dessin déjà réalisé sur lequel vous allez faire des ajouts. Pour ce faire, nous allons utiliser

deux fonctions principales : `imagecreate()` et `imagecreatefromjpeg()` (ou `imagecreatefrompng()`, selon le type de fichier lu).

### Le format GIF

On remarquera la réapparition de l'importation de fichier GIF via la fonction `imagecreatefromgif()`. Effectivement, la bibliothèque GD avait arrêté de prendre en charge le GIF depuis sa version 1.3.

Le format GIF standard utilise une compression de type LZW (Lempel Ziv Welch), protégée par un brevet déposé par la société Unisys. Les auteurs de logiciels produisant du format GIF doivent ainsi payer des droits de licence. Cette situation est bien sûr très embarrassante, car le format GIF s'est peu à peu imposé comme standard pour les graphiques : à ses débuts, aucun droit n'était exigé dessus. La société Unisys veut maintenant exploiter la situation. L'équipe de GD a alors retiré les fonctions de lecture et écriture vers le format GIF à partir de leur version 1.3.

La licence implique que des droits sont redevables en cas de création de fichier GIF, mais la lecture est tout à fait libre. Ainsi, l'équipe de développement a décidé de remettre en place la fonction d'importation d'images au format GIF via la fonction `imagecreatefromgif()`. La fonction de sauvegarde au format GIF est toujours absente pour le moment, mais le brevet ne devrait pas tarder à expirer et elle sera probablement réintroduite à ce moment-là. Entre temps, vous pouvez toujours convertir vos images au format PNG, qui offre plus de fonctionnalités, un meilleur rendu et une meilleure compression.

Chacune de ces fonctions va vous renvoyer un identifiant de ressource. Il s'agit d'une donnée interne qui permettra d'identifier l'image que vous êtes en train de manipuler. Il vous sera redemandé dans toutes les fonctions de manipulation d'image suivantes.

### Créer une « feuille blanche »

```
■ imagecreate( largeur, hauteur );
```

Pour créer une feuille blanche sur laquelle on travaillera, il faut utiliser la fonction `imagecreate()`. Celle-ci va s'occuper de réserver l'espace mémoire nécessaire pour la création d'une image vierge possédant les dimensions indiquées en paramètres. La fonction renverra un identifiant représentant cette image. Vous allez ainsi créer votre feuille blanche sur laquelle vous allez par la suite dessiner.

```
<?php
// Allocation des valeurs
$largeur = 200;
$hauteur = 200;

// Création de la feuille blanche
$image = imagecreate($largeur,$hauteur);

/* À cet instant, PHP vient de réserver en mémoire de l'espace pour stocker votre image */
?>
```

`imagecreate()` est utilisé pour créer des images disposant d'une palette de 256 couleurs (souvent nommées couleurs indexées). Cela correspond en fait au nombre de couleurs maximal que peuvent contenir des fichiers GIF et certaines variantes de PNG.

Si vous travaillez sur des images disposant d'une palette de couleurs plus importante (JPEG, PNG 32 bits), utilisez la fonction `imagecreatetruecolor()`. Cette dernière fonction est toutefois plus gourmande en mémoire.

```
imagecreatetruecolor(largeur, hauteur);
```

### Utiliser une image existante

Si vous ne désirez pas travailler sur une feuille blanche, mais sur une image existante, il existe plusieurs fonctions basées sur le même principe. Vous trouverez dans la liste suivante une liste des différentes fonctions utilisables suivant le type d'image que vous avez au départ.

```
// Utiliser une image au format JPEG
imagecreatefromjpeg(cheminfichierimage)

// Utiliser une image au format GIF
imagecreatefromgif(cheminfichierimage)

// Utiliser une image au format PNG
imagecreatefrompng(cheminfichierimage)

// Utiliser une image au format BMP
imagecreatefrombmp(cheminfichierimage)

// Utiliser une image au format XPM
imagecreatefromxpm(cheminfichierimage)

// Utiliser une image au format XBM
imagecreatefromxbm(cheminfichierimage)

// Utiliser une image au format WBMP
imagecreatefromwbmp(cheminfichierimage)
```

La fonction `imagecreatefromjpeg()` permet ainsi d'ouvrir une image JPEG à partir de l'adresse de son fichier et renvoie un identifiant du même type que celui reçu par `imagecreate()`.

```
<?php
// Récupération des images
$image1 = imagecreatefromjpeg('./images/logo.jpg');
$image2 = imagecreatefrompng('./images/logo.png');

// On peut aussi ouvrir une image distante
$url = 'http://php.net/images/php.gif';
$image3 = imagecreatefromgif($url);

/* À cet instant, PHP vient de réserver en mémoire de l'espace pour stocker vos images */
?>
```

## L'allocation des couleurs

```
imagecolorallocate( image, rouge, vert, bleu );
```

Certains formats d'images sont dits « indexés ». Ils ont en fait un nombre de couleurs limité, déterminé à l'avance. Vous ne pouvez donc pas spécifier directement un code couleur dans les fonctions pour dessiner des figures ou écrire du texte : votre couleur risquerait de ne pas exister dans l'index.

Pour utiliser une couleur, nous devons donc demander à GD de nous retourner un identifiant correspondant à la couleur que nous souhaitons utiliser. Cette opération est faite par la fonction `imagecolorallocate()` qui prend en paramètres un identifiant d'images et trois composantes qui correspondent respectivement aux niveaux de rouge, vert et bleu (RVB). Ces composantes sont des entiers de 0 à 255.

```
<?php
// Allocation des valeurs
$largeur = 200;
$hauteur = 200;
// Création de la feuille blanche
$image = imagecreate($largeur,$hauteur);
// Allocation des couleurs
$noir = imagecolorallocate($image,0,0,0);
$rouge = imagecolorallocate($image,255,0,0);
$vert = imagecolorallocate($image,0,0xFF,0);
// Libération de l'espace mémoire
imagedestroy($image);
?>
```

### Remarque

La première couleur allouée devient automatiquement la couleur d'arrière-plan de l'image. Notons également qu'il est bien sûr possible de changer cette couleur d'arrière-plan.

Cette explication devrait vous suffire pour commencer. Nous verrons plus loin qu'il peut être nécessaire de redéfinir la palette de couleurs pour éviter d'être bloqué par les couleurs allouées dynamiquement (lors d'une récupération d'image PNG et GIF notamment).

## Libérer les ressources mémoire

```
imagedestroy( identifiant_image );
```

Nous abordons tout de suite cet aspect, car il est nécessaire d'y faire attention afin de ne pas surcharger votre serveur. PHP charge en mémoire toute l'image sur laquelle il est en train de travailler ; cela peut représenter une taille non négligeable. Une fois que vous avez fini d'utiliser l'image, il vous faudra toujours penser à libérer la mémoire occupée

par l'image temporaire avec la fonction `imagedestroy()`. Elle prend un identifiant d'image en unique paramètre.

```
<?php
// Récupération des images
$image1 = imagecreatefromjpeg('./images/logo.jpg');
$image2 = imagecreatefromgif('./images/logo.gif');
$image3 = imagecreatefrompng('./images/logo.png');
/* À cet instant, PHP vient de réserver en mémoire de l'espace pour stocker vos images */

// Libération de l'espace mémoire
imagedestroy($image1);
imagedestroy($image2);
imagedestroy($image3);
?>
```

## Affichage de l'image sur le navigateur

### Remarque

Si vous affichez directement une image fabriquée, elle sera reconstruite à chaque appel ou rafraîchissement de la page. Un tel comportement entraîne une consommation importante de ressources sur votre serveur. Si votre contenu le permet, vous devriez envisager l'utilisation d'un cache (voir le chapitre 24 à ce sujet).

## Intégration d'image dans une page HTML

Il n'est pas possible d'intégrer directement une image dans une page HTML. PHP vous renverra soit un document HTML, soit un contenu binaire représentant une image. Vous pouvez en revanche insérer une balise image classique (``) et utiliser l'adresse d'un script PHP dans le lien comme source de l'image. Ce script pourra alors, lui, renvoyer le contenu d'une image (et uniquement de l'image) au navigateur.

## Déclaration de l'image avant envoi

Les images manipulées peuvent être soit sauvegardées dans des fichiers, soit directement envoyées au navigateur. Dans ce dernier cas, il vous faut déclarer au navigateur que ce qu'il va recevoir est une image et spécifier son format avant d'envoyer l'image.

En effet, lors de l'exécution d'un script sur un serveur web, PHP envoie par défaut au navigateur un en-tête HTTP pour dire que le contenu envoyé est une page HTML. Si nous voulons envoyer une image à la place, il nous faut changer cette déclaration. L'en-tête HTTP concerné est `Content-Type`. La valeur à envoyer dépend du format de l'image. Pour une image PNG, c'est `image/png`, pour une image JPEG, c'est `image/jpeg`.

Vous pouvez envoyer cet en-tête avec la fonction `header()`. L'appel doit se faire avant d'envoyer l'image au navigateur, probablement en haut de votre script (pour plus d'informations sur les en-têtes HTTP et le contexte Web, vous pouvez consulter le chapitre 9).

```
header('Content-Type: image/png');
```

**Attention**

L'en-tête HTTP doit être écrit tel que dans l'exemple. Des espaces surnuméraires risqueraient d'empêcher sa compréhension par certains navigateurs.

Si vos images ne sont pas lisibles sur le navigateur, vérifiez bien que vous envoyez l'en-tête adéquat tout en haut de votre script. Vous ne devez par ailleurs envoyer aucun autre contenu que votre image : pas de HTML ni même d'espaces ou lignes vides.

**Envoi de l'image au navigateur**

Une fois que vous avez déclaré votre contenu comme étant une image, vous pouvez envoyer directement les données vers le navigateur.

```
imagepng( id_image [, fichier] ) ;  
imagejpeg( id_image [, fichier, qualite] ) ;
```

La fonction `imagepng()` prend un identifiant d'image en paramètre, la convertit au format PNG et en affiche le contenu sur la sortie (le navigateur). La valeur de retour est `TRUE` en cas de réussite, `FALSE` en cas d'échec.

```
<?php  
$image = imagecreatefromgif('test.gif');  
header('Content-Type: image/png') ;  
imagepng($image);  
?>
```

Des fonctions identiques existent pour les autres formats. Vous pouvez par exemple utiliser `imagejpeg()` pour le format JPEG ou `imagewbmp()` pour le format BMP Windows.

**Enregistrer l'image dans un fichier**

```
imagepng( id_image, [adresse_fichier] ) ;
```

Plutôt que d'afficher toujours directement le contenu d'une image sur le navigateur, vous pouvez préférer l'enregistrer sur le disque pour obtenir un fichier image classique. Il est par exemple possible d'ouvrir ainsi toutes les images d'un répertoire, de les redimensionner et de les réenregistrer dans leur nouvelle taille afin de les homogénéiser. Vous pouvez aussi utiliser l'enregistrement pour sauvegarder votre image en cache et ne pas la reconstruire à chaque affichage.

L'enregistrement d'une image se fait exactement comme la dernière étape d'envoi d'une image au navigateur. Il suffit juste de donner l'adresse du fichier cible à la fonction `imagepng()` (ou équivalent).

```
<?php  
$image = imagecreatefromgif('test.gif');  
header('Content-Type: image/png') ;  
imagepng($image, 'test.png');  
?>
```

## Définir la qualité d'une image JPEG

La fonction `imagejpeg()` dispose d'un troisième paramètre qui permet de spécifier la qualité d'encodage. La valeur varie de la plus forte compression (1) à la compression la plus fine (100).

```
<?php
$image = imagecreatefromjpeg('test.jpg');
header('Content-Type: image/jpeg') ;
imagejpeg($image, 'test.jpg',95);
?>
```

## Travail sur une image

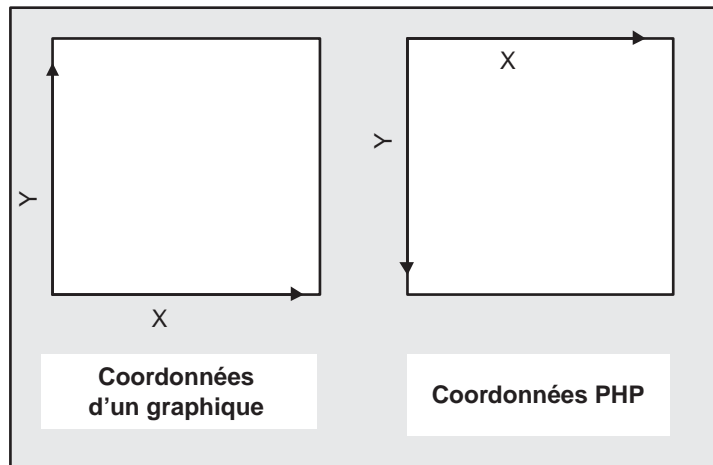
Après avoir vu les différentes possibilités pour créer notre support de travail (feuille blanche ou image existante), nous allons maintenant voir comment travailler sur ce support pour modifier le contenu de l'image.

### Le référentiel

La première chose à aborder est le système de coordonnées de l'image. En effet, si vous êtes habitué aux valeurs X et Y qui ont pour origine le coin inférieur gauche de votre image, PHP pourrait vous dérouter. Le point référence des images dans la bibliothèque GD est en effet le point supérieur gauche de l'image (voir figure 25-1).

Figure 25-1

*Le système de coordonnées de PHP*



### Tracer des formes

La bibliothèque GD fournit un panel de fonctions permettant de créer des formes primaires telles que des rectangles, des ellipses (donc des cercles), des arcs ou des lignes.

Le tableau 25-1 présente quelques-unes des fonctions de création de formes disponibles. Deux de ces fonctions seront décrites en détail par la suite pour vous en expliquer le fonctionnement.

Généralement, ces fonctions prennent en premier paramètre l'identifiant de l'image dans laquelle la forme doit être créée, puis les coordonnées de la forme, et enfin la couleur de l'élément (soit des bords, soit du remplissage). Elles retournent `TRUE` si l'élément a pu être dessiné et `FALSE` dans le cas contraire.

**Tableau 25-1 Fonctions de création de formes**

Fonction	Description
<code>imagearc()</code>	Crée un arc à partir de son centre, sa largeur et sa hauteur. <code>imagefilledarc()</code> permet de tracer cet arc de cercle et de le remplir.
<code>imagedashedline()</code>	Trace un trait pointillé entre deux points avec la couleur spécifiée.
<code>imagepolygon()</code>	Crée un polygone dont les points sont stockés dans un tableau de valeurs. Le nombre de points à utiliser dans le tableau est à spécifier dans le paramètre suivant. <code>imagefilledpolygon()</code> permet de tracer un polygone identique mais rempli avec la couleur spécifiée.
<code>imagerectangle()</code>	Crée un rectangle dont les coins supérieur gauche et inférieur droit sont spécifiés en argument. <code>imagefilledrectangle()</code> permet de tracer ce même rectangle et de le remplir de couleur.
<code>imageellipse()</code>	Crée une ellipse à partir des coordonnées de son centre, d'une largeur et d'une hauteur. <code>imagefilledellipse()</code> remplit l'ellipse après le tracé.
<code>imageline()</code>	Trace une ligne entre deux points avec la couleur spécifiée.
<code>imagefill()</code>	Effectue un remplissage avec la couleur indiquée, dans l'image, à partir d'un point de coordonnées. Le remplissage se limite aux contours.

### Tracé d'un arc de cercle

```
imagearc( image, centre_x, centre_y, largeur, hauteur,  
          angle_début, angle_fin, couleur );
```

La fonction `imagearc()` permet de créer un arc, de centre (`centre_x`, `centre_y`), dont la largeur et la hauteur sont en pixels. Les paramètres `angle_début` et `angle_fin` représentent les angles de début et de fin (le degré 0 est à 3 heures) en tournant dans le sens des aiguilles d'une montre. Un exemple est donné avec le code suivant, dont le résultat est disponible à la figure 25-2.

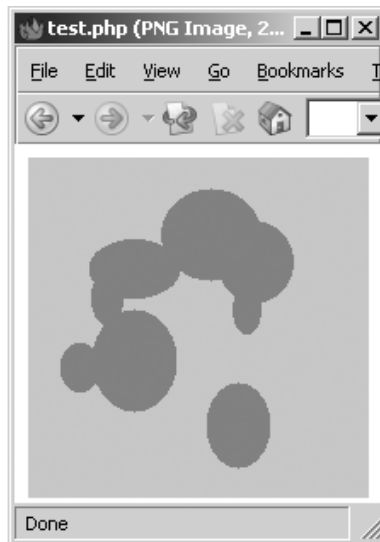
```
<?php  
$largeur = 200;  
$hauteur = 200;  
$im = imagecreate($largeur,$hauteur);  
$gris = imagecolorallocate($im,200,200,200);  
$red = imagecolorallocate($im,255,0,0);  
$vert = imagecolorallocate($im,0,0xFF,0);
```



```
// Ajout des arcs sur l'image
for ($i=1;$i<10;$i++){
    $a_x = mt_rand(30,170);
    $a_y = mt_rand(30,170);
    $a_tx = mt_rand(0,60);
    $a_ty = mt_rand(0,60);
    imagefilledarc($im,$a_x,$a_y,$a_tx,$a_ty,0,360,$red,IMG_ARC_PIE);
}

header('Content-Type: image/png');
imagepng($im);
imagedestroy($im);
?>
```

**Figure 25-2**  
*Tracé d'arcs  
de cercle*



### Tracé d'un rectangle

```
imagefilledrectangle( image, x1, y1, x2, y2, couleur )
```

La fonction `imagefilledrectangle()` permet de dessiner un rectangle de couleur dans l'image `$image`, en commençant par le sommet supérieur gauche (`x1, y1`) et finissant au sommet inférieur droit (`x2, y2`). Le terme *filled* dans le nom de la fonction implique que le rectangle sera rempli avec la couleur spécifiée en dernier paramètre. Un exemple est donné avec le code suivant et la figure 25-3.

```
<?php
$largeur = 190;
$hauteur = 200;
```

```
$image = imagecreate($largeur,$hauteur);

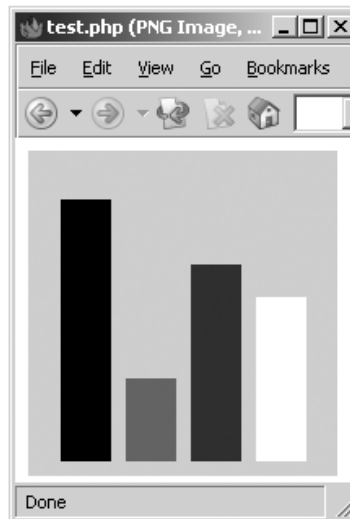
// Allocation des couleurs
$gris = imagecolorallocate($image,207,207,207);
$noir = imagecolorallocate($image,0,0,0);
$gris2 = imagecolorallocate($image,99,99,99);
$gris3 = imagecolorallocate($image,43,43,43);
$blanc = imagecolorallocate($image,255,255,255);

// Ajout d'un rectangle sur l'image
imagefilledrectangle($image,20,30,50,190,$noir);
imagefilledrectangle($image,60,140,90,190,$gris2);
imagefilledrectangle($image,100,70,130,190,$gris3);
imagefilledrectangle($image,140,90,170,190,$blanc);
header('Content-Type: image/png');

// Fabrication de l'image
imagepng($image);
imagedestroy($image);
?>
```

**Figure 25-3**

*Tracé de rectangles  
pleins*



## Écrire du texte

PHP permet aussi de dessiner des chaînes de caractères dans une image grâce à une grande variété de fonctions dédiées. Cela autorise la création de légendes ou de boutons avec un texte dynamique.

Le tableau 25-2 présente les principales fonctions de gestion des chaînes de caractères dans les images.

**Tableau 25-2 Fonctions d'affichage de chaînes de caractères**

Fonction	Description
imagechar()	Crée un caractère à l'emplacement (x,y). La police du caractère peut être choisie parmi les polices par défaut (1 à 5) ou bien une police personnalisée que vous avez ouverte précédemment avec <code>imageloadfont()</code> .
imagecharup()	Crée un caractère orienté à l'horizontale (rotation de 90°) à l'emplacement (x,y). La police du caractère peut être choisie parmi les polices par défaut (1 à 5) ou bien une police personnalisée que vous avez ouverte précédemment avec <code>imageloadfont()</code> .
imagefontheight()	Retourne la hauteur de la police utilisée.
imageloadfont()	Charge la police dont le nom est passé en argument et retourne son identifiant.
imagestring()	Crée une chaîne de caractères à l'emplacement (x,y). La police du caractère peut être choisie parmi les polices par défaut (1 à 5) ou bien une police personnalisée que vous avez ouverte précédemment avec <code>imageloadfont()</code> .
imagestringup()	Crée une chaîne de caractères orientée verticalement (rotation de 90°) à l'emplacement (x,y). La police du caractère peut être choisie parmi les polices par défaut (1 à 5) ou bien une police personnalisée que vous avez ouverte précédemment avec <code>imageloadfont()</code> .

## Gestion des polices de caractères

### Polices par défaut

PHP propose cinq polices de caractères par défaut, numérotées de 1 à 5. Notre exemple, dont le résultat est visible en figure 25-4, vous permet d'afficher les polices de base.

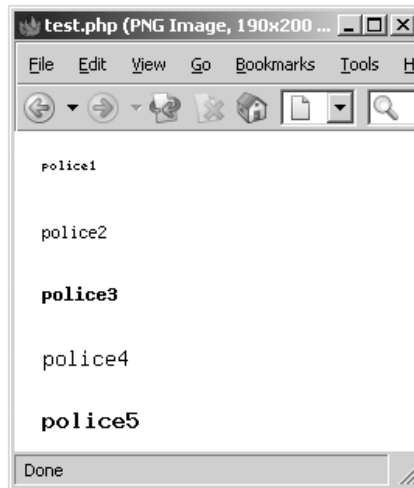
```
<?php
$largeur = 190;
$hauteur = 200;
$image = imagecreate($largeur,$hauteur);
$blanc = imagecolorallocate($image,255,255,255);
$noir = imagecolorallocate($image,0,0,0);

// Ajout d'un rectangle sur l'image
imagestring($image,1,10,10,'police1',$noir);
imagestring($image,2,10,50,'police2',$noir);
imagestring($image,3,10,90,'police3',$noir);
imagestring($image,4,10,130,'police4',$noir);
imagestring($image,5,10,170,'police5',$noir);

header('Content-Type: image/png');
imagepng($image);
imagedestroy($image);
?>
```

Figure 25-4

*Les cinq polices  
par défaut*



### Choix de la police de caractères

Il est possible de spécifier une police de caractères personnalisée grâce à la fonction `imageloadfont()`. Elle vous retournera un identifiant que vous pourrez utiliser en paramètre pour écrire votre texte. Le format du fichier de police passé en paramètre dépend du système sur lequel PHP fonctionne.

```
imageloadfont( adresse_du_fichier_de_police );
```

### Taille d'une police de caractères

Les fonctions `imagefontwidth()` et `imagefontheight()` vous renverront la largeur et la hauteur de la police passée en unique paramètre. Il est alors facile de connaître le nombre de pixels que va occuper la chaîne entière :

```
$largeur = imagefontwidth($police) * strlen($text);  
$hauteur = imagefontheight($police);
```

### Écriture du texte dans l'image

La principale fonction permettant d'écrire un texte est `imagestring()`. Elle insère un texte à partir d'une position donnée et dans une certaine police de caractères. La police de caractères est soit un chiffre entre 1 et 5 compris, soit un identifiant retourné par `imageloadfont()`, comme vu précédemment.

```
imagestring( image, police, x, y, texte, couleur);
```

### Texte vertical

On utilise la fonction `imagestringup()` pour insérer un texte sur une ligne verticale dans l'image :

```
imagestringup( id_image, police, x, y, chaine, couleur)
```

## Exemple

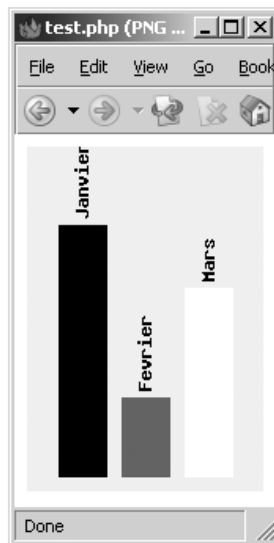
L'exemple suivant utilise les fonctions de tracé de rectangles et d'écriture de texte pour réaliser un diagramme en barres avec légendes (voir figure 25-5).

```
<?php
$largeur = 150;
$hauteur = 220;
$image = imagecreate($largeur,$hauteur);
$gris = imagecolorallocate($image,240,240,240);
$noir = imagecolorallocate($image,0,0,0);
$gris2 = imagecolorallocate($image,99,99,99);
$blanc = imagecolorallocate($image,255,255,255);

// Ajout d'un rectangle sur l'image
imagefilledrectangle($image,20,50,50,210,$noir);
imagefilledrectangle($image,60,160,90,210,$gris2);
imagefilledrectangle($image,100,90,130,210,$blanc);
imagestringup($image,3,28,45, 'Janvier',$noir);
imagestringup($image,3,68,155, 'Fevrier',$noir);
imagestringup($image,3,108,85, 'Mars',$noir);
header('Content-type:image/png');
imagepng($image);
imagedestroy($image);
?>
```

**Figure 25-5**

*Image réalisée avec des rectangles et du texte*



## Copie d'une zone d'image

Il est possible de copier une partie d'une image ouverte dans une autre (ou de copier une partie d'une image dans elle-même). Vous devez spécifier à la fonction `imagecopy()` l'identifiant de l'image destination, l'identifiant de l'image source, les coordonnées du point où copier sur l'image destination, les coordonnées références du point à partir duquel copier dans l'image source, et enfin les largeur et hauteur du rectangle à copier. La partie copiée remplace ce qu'il y avait avant dans l'image destination.

```
imagecopy ( image_destination, image_source, x_destination,  
            y_destination, x_source, y_source,  
            largeur, hauteur ) ;
```

### Attention

Contrairement aux habitudes, les fonctions de copie d'image demandent de spécifier la destination avant la source dans les paramètres.

## Copie et fusion d'images

Il est possible de demander à PHP de fusionner les deux images lors d'une copie, plutôt que de remplacer. Il vous suffit d'utiliser `imagecopymerge()` de la même façon que `imagecopy()`, mais de spécifier en plus une opacité. À 0, l'image de destination ne sera pas modifiée ; à 100, le rendu sera identique à une copie simple.

```
imagecopymerge( image_destination, image_source,  
                x_destination, y_destination,  
                x_source, y_source,  
                largeur, hauteur, opacite ) ;
```

## Copie et redimensionnement

La fonction `imagecopyresized()` copie une zone rectangulaire d'une image vers une autre en la redimensionnant. Vous devez donc spécifier les largeurs et hauteurs utilisées sur les deux images. Si ces dernières sont différentes, la partie de l'image copiée sera étirée ou réduite.

```
imagecopyresized( image_destination, image_source,  
                  x_destination, y_destination,  
                  x_source, y_source,  
                  largeur_destination, hauteur_destination,  
                  largeur_source, hauteur_source ) ;
```

En utilisant plutôt `imagecopyresampled()`, la zone copiée sera ré-échantillonnée afin de conserver une clarté d'image correcte.

## Rotation d'une image

`imagerotate()` fait tourner une image d'un certain angle (en degrés). Le dernier paramètre permet de spécifier la couleur à utiliser pour les parties de l'image qui seront à découvert après l'opération.

```
imagerotate( image, angle, couleur )
```

### Note

`imagerotate()` redimensionne le cadre de l'image de manière à la contenir toute entière.

## Gestion de la palette de couleurs

Nous avons vu au début de ce chapitre la fonction `imagecolorallocate()`, qui permet d'allouer une couleur. Sur certains formats d'image, les couleurs sont en effet en nombre limité. On crée alors une palette de couleurs pour indexer les différentes couleurs utilisées. Les images GIF sont par exemple limitées à 256 couleurs.

Pour des images en « vraies couleurs », vous n'avez pas besoin de vous préoccuper des index. C'est par exemple le cas pour des JPEG ou certaines variantes de PNG.

### Allocation d'une couleur avec transparence

```
imagecolorallocatealpha( image, rouge, vert, bleu, alpha ) ;
```

Dans les formats d'image le permettant (PNG par exemple), il est possible de définir des couleurs partiellement transparentes ; on parle alors de couleurs avec un canal alpha. Pour allouer de telles couleurs, il vous suffit de faire appel à `imagecolorallocatealpha()` en ajoutant une valeur de transparence en plus des trois composantes de couleur. La transparence peut aller de 0 (couleur opaque classique) à 127 (couleur totalement transparente). Un identifiant de couleur vous sera renvoyé comme pour un appel à `imagecolorallocate()`.

### Récupérer l'identifiant d'une couleur proche

```
imagecolorclosest( image, rouge, vert, bleu ) ;  
imagecolorclosestalpha( image, rouge, vert, bleu, alpha ) ;
```

Plutôt que d'allouer une nouvelle couleur dans l'index, il est possible de demander à GD de retourner la couleur la plus proche dans l'index avec `imagecolorclosest()`. La fonction `imagecolorclosestalpha()` fonctionne de manière similaire, mais gère aussi un paramètre de transparence.

### Retirer une couleur de l'index

```
imagecolordeallocate( image, couleur ) ;
```

La fonction `imagecolordeallocate()` permet de retirer une couleur de l'index à partir de son identifiant. Vous pouvez ainsi diminuer la taille de votre palette et le poids de l'image totale si elle est indexée (ou libérer une place dans l'index pour une autre couleur).

### Couleur d'un pixel de l'image

```
imagecolorat( image, x, y );
```

Si vous modifiez des images existantes, il peut être utile de connaître la couleur exacte de certaines parties de l'image. Vous pouvez l'obtenir avec `imagecolorat()`, en passant en paramètres les coordonnées du pixel dans l'image. PHP vous retournera un identifiant de couleur.

### Couleur transparente

```
imagecolortransparent( image, couleur );
```

Certains formats d'image permettent de gérer une transparence binaire (c'est le cas de GIF et de certaines variantes de PNG). Vous devez prendre une couleur arbitraire (généralement une que vous ne risquez pas d'utiliser) et décider que cette couleur apparaîtra transparente. La fonction `imagecolortransparent()` permet de faire une telle association sur la couleur passée en argument.

### Copie d'une palette de couleurs

```
imagepalettecopy( destination, source );
```

La fonction `imagepalettecopy()` permet de copier la palette d'une image vers une autre. Cela permet effectivement d'éviter de perdre ou de tronquer des couleurs quand des palettes de couleurs limitées sont associées aux images.

## Connaître la taille d'une image

```
getimagesize( adresse_fichier );
```

Il est parfois important de connaître la taille d'une image pour effectuer un traitement sur elle (redimensionnement, couplage, etc.).

La fonction `getimagesize()` détermine la taille de l'image dont l'adresse de fichier est passée en argument. La valeur de retour est un tableau qui contient la largeur en pixels à l'index 0 et la longueur à l'index 1.

#### Note

Il est possible d'utiliser cette fonction sur un fichier distant. Dans ce cas, PHP télécharge le fichier pour calculer la taille, ce qui peut impliquer un temps d'attente non négligeable.



## Astuces et remarques

### Éviter les fausses couleurs

Quand vous travaillez avec des images extérieures telles que des photos ou des captures d'écran, il vous est probablement nécessaire de travailler en « vraies couleurs ». Si vous ne le faites pas, dans le cas d'une image GIF ou de certaines images PNG, vous allez vous limiter à la palette de couleurs définie par l'image.

Dans vos manipulations, GD prend à chaque fois la couleur la plus proche de ce que vous demandez. Cette couleur peut toutefois être totalement différente de celle attendue (si votre palette ne contient que des verts, vous ne pourrez jamais y insérer du rouge).

Pour éviter cet effet voile noir, il suffit de remplacer `imagecreate()` par `imagecreatetruecolor()`.

### Limite de temps

Certaines images peuvent demander un temps de calcul important. Vous risquez alors de dépasser le temps alloué à votre script PHP (par défaut 30 secondes). Si la directive de configuration PHP `safe_mode` n'est pas activée sur le serveur, vous pouvez modifier la variable de configuration `max_execution_time` pour mettre un temps d'expiration plus adapté (le temps est donné en secondes, 0 pour désactiver la limite).

```
max_execution_time = 0
```

Vous pouvez aussi modifier cette valeur au niveau du script et non de la configuration globale avec la fonction `set_time_limit()`. Chaque appel remet à zéro le compteur de temps et alloue un nouveau délai.

```
set_time_limit( 120 );
```

Faire attendre longtemps l'utilisateur ou charger votre serveur n'est généralement pas une bonne idée. Si cela est cohérent, pensez à un système de cache.

### Malvoyants et référencement

On a tendance à l'oublier, mais il est important de penser au plus grand nombre lors d'un développement. Ainsi, pour les images, il convient de renseigner les attributs `alt`, `title` et `longdesc` de la balise `img` dans le code HTML. Le premier définit un texte alternatif (la page doit être lisible normalement si on remplace les images par le contenu de l'attribut `alt`) et les deux autres définissent un titre à l'image et une description du contenu.

Remplir ces balises est d'autant plus important que tous les moteurs de recherche ou les logiciels informatiques agissent comme des aveugles. Ils ne comprendront pas les icônes et les textes dans les images. Sans contenu alternatif, votre page risque d'être mal référencée ou indexée dans les moteurs de recherche par exemple. Plus de renseignements à ce propos peuvent être trouvés sur [http://openweb.eu.org/articles/accessibilite\\_images/](http://openweb.eu.org/articles/accessibilite_images/).

## L'outil Open Source de gestion d'albums photos : Gallery

De nombreux outils Open Source existent dans le domaine de l'image. Nous présentons juste après la bibliothèque JpGraph, qui est une des plus belles réussites en matière de bibliothèque graphique. Dans un premier temps, nous allons vous présenter le logiciel de gestion d'albums photos Gallery (<http://gallery.menalto.com>).

Incontournable, le logiciel Gallery remplit la majorité des fonctions qu'on peut attendre d'un album photo en ligne.

Son installation se fait via une interface spécifique rendant cette étape relativement simple. Les utilisateurs ont la possibilité de créer et de maintenir leurs propres albums par l'intermédiaire d'une interface intuitive.

La gestion de photos inclut la création automatique de miniatures, le redimensionnement de l'image, la rotation, le tri, l'attribution d'un libellé, etc.

Les albums peuvent avoir des permissions individuelles selon l'utilisateur enregistré.

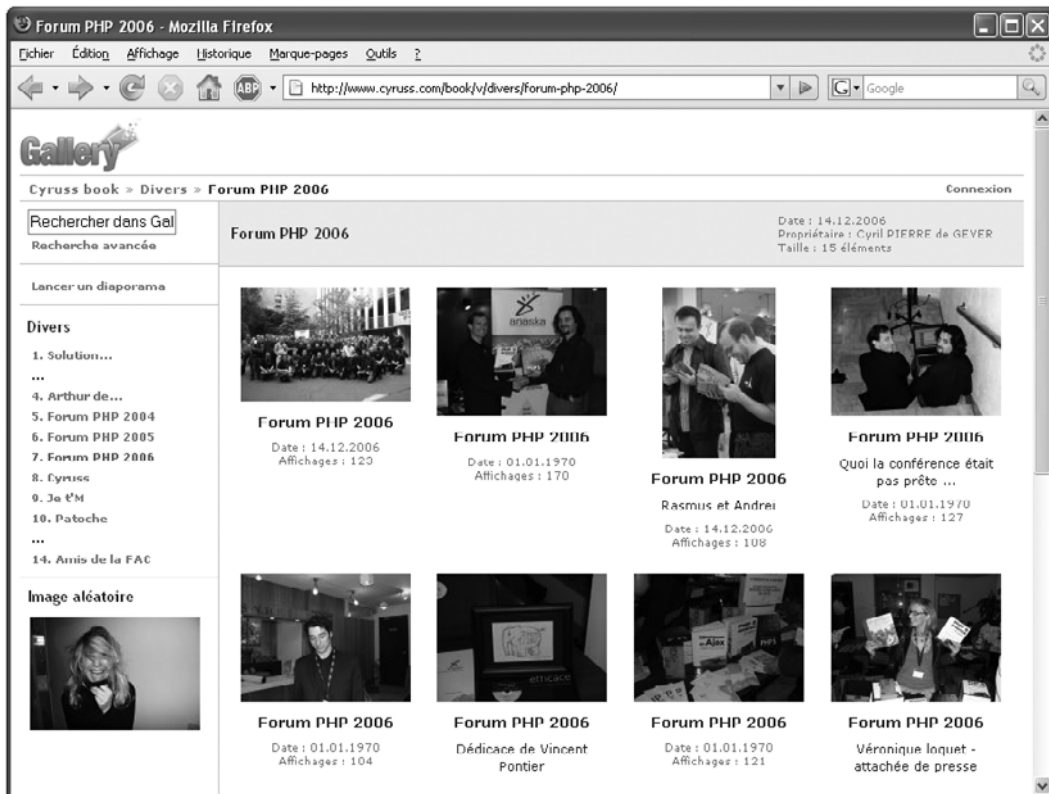


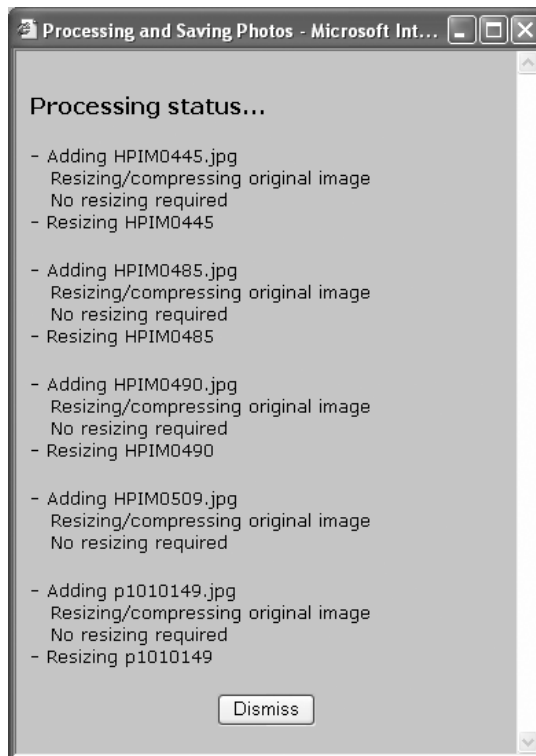
Figure 25-6

Le logiciel Gallery

La partie d'administration est également très simple et intuitive. On retrouve dans la figure suivante le visuel correspondant à l'insertion de nouvelles images dans un portfolio.

Figure 25-7

*Ajout d'images avec  
Gallery*



## La bibliothèque Open Source JpGraph

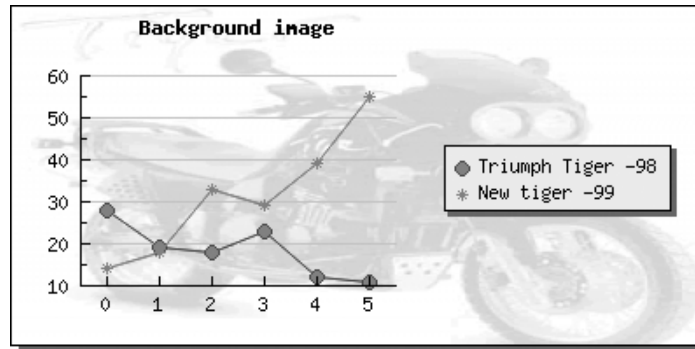
Il existe plusieurs classes d'objets qui implémentent une interface simple pour fabriquer des graphiques. Il n'est donc pas nécessaire de faire appel aux fonctions bas niveau de la bibliothèque GD. Une de ces bibliothèques est la JpGraph, choisie ici parce que ses performances sont remarquables et parce que son développeur promet encore de nombreuses améliorations (Joan Persson est l'auteur de cette bibliothèque, sous licence QTL, libre d'utilisation pour des applications non commerciales). Si vous souhaitez utiliser cette bibliothèque pour des applications commerciales, rendez-vous sur le site pour plus d'informations (<http://www.aditus.nu/jpgraph/>).

### Note

L'alternative la plus intéressante à JpGraph se nomme Artichow (<http://www.artichow.org>). Elle bénéficie d'une qualité au moins aussi importante et d'une licence plus permissive.

Figure 25-8

Exemple de graphique fabriqué par la JpGraph



Les fonctionnalités de la bibliothèque JpGraph sont :

- Les échelles variables : il est possible de combiner les échelles linéaires, des énumérations et des échelles logarithmiques sur un même graphique.
- La prise en charge de deux axes d'ordonnées différents sur le même graphique : un axe à gauche et un à droite du graphique avec des échelles différentes.
- L'utilisateur peut préciser la position des axes.
- Les images conçues peuvent être mises en cache pour réduire la charge du serveur.
- Les images map (<imagemap> en HTML) sont autorisées pour produire des images cliquables.
- Les formats des graphiques proposés sont les lignes, les cumuls, les histogrammes simples et cumulés, les graphiques en étoile et les graphes en secteur ; ces derniers peuvent être dessinés en perspective.
- La gestion des légendes est automatisée.
- La fonction d'anti-aliasing spécifique, qui évite de trop lourds calculs aux serveurs web.

La JpGraph est complètement orientée objet. Le système est totalement cohérent, dans la mesure où toutes les méthodes respectent la même convention de nommage et les mêmes règles d'appel. Vous utilisez par exemple toujours la méthode `SetColor()`, quel que soit le type de l'objet (axe, grilles, textes, titres, lignes, etc.).

## Installation et configuration

Les étapes d'installation sont les suivantes. Téléchargez les fichiers de la bibliothèque sur le site de l'auteur : <http://www.aditus.nu/JpGraph/>. Copiez les fichiers de la distribution sur votre serveur web.

Lancez ensuite les exemples situés dans le sous-répertoire de la distribution pour vérifier si tout fonctionne. En cas de problème, une fois vérifié que gd est bien activé, vous

pouvez modifier le fichier `jpg-config.inc.php` pour que les chemins d'accès au répertoire cache et aux fichiers des polices de caractères soient corrects. Pour utiliser les fonctions de cache, vous devrez autoriser les scripts web à écrire dans le répertoire cache que vous avez indiqué.

Lorsque vous vous êtes assuré que les exemples sont opérationnels, vous pouvez utiliser la bibliothèque à partir de vos propres scripts. Pour cela, copiez les fichiers `JpGraph.php` et `JpGraph_XXX.php` dans le répertoire que vous utilisez pour stocker vos classes PHP :

```
<?php
include ('JpGraph.php');
include ('JpGraph_line.php');
// Code qui utilise les fonctions de dessin de lignes
// de la bibliothèque JpGraph
?>
```

### Remarque

Nous avons vu l'existence d'un paramètre de configuration nommé `include_path`. Il pourrait être intéressant d'y ajouter le chemin où sont stockés les fichiers de la JpGraph.

Le tableau 25-3 indique les points à prendre en compte dans le fichier de configuration.

**Tableau 25-3 Configuration de la JpGraph**

Constante	Valeur par défaut	Description
ERR_DEPRECATED	FALSE	Provoque une erreur fatale si des fonctions ou des valeurs incorrectes sont envoyées.
READ_CACHE	TRUE	Utilisation du cache. Dans ce cas, le logiciel vérifie si l'image a déjà été calculée ; si c'est le cas, celle du cache est envoyée.
CACHE_DIR	./JpGraph_cache	Répertoire cache. PHP doit pouvoir écrire dans ce répertoire.
TTF_DIR	./ttf	Répertoire des polices de caractères TTF.
DEFAULT_GFORMAT	auto	Format graphique utilisé. Si aucun n'est précisé, le logiciel choisira, dans l'ordre, png, gif ou jpg.

## Architecture de la JpGraph

**Tableau 25-4 Les différents fichiers de la JpGraph**

Fichier	Utilité	Remarque
JpGraph.php	Bibliothèque de base	Toujours l'inclure
JpGraph_error.php	Graphique d'erreurs	
JpGraph_pie.php	Graphes en secteurs : camemberts	

Tableau 25-4 Les différents fichiers de la JpGraph (*suite*)

Fichier	Utilité	Remarque
JpGraph_pie3.php	Graphes en secteurs 3D	Est associé au fichier JpGraph_pie.php
JpGraph_line.php	Graphiques en ligne	
JpGraph_bar.php	Histogrammes	
JpGraph_scatter.php	Points	
JpGraph_log.php	Échelles logarithmiques	
JpGraph_spider.php	Graphique en étoile	
JpGraph_canvas.php	Classe avec primitives pour réaliser des dessins	À utiliser avec JpGraph pour dessiner avec PHP comme avec un autre langage de programmation.

Utiliser la bibliothèque JpGraph n'est pas extrêmement complexe du fait d'une implémentation logique écrite suivant une méthodologie objet. Nous allons détailler quelques-uns des types de graphiques élaborés par cet outil.

Pour accéder aux méthodes de la bibliothèque, il faut inclure au moins deux fichiers : la bibliothèque de base et la bibliothèque concernant le type de graphique souhaité. Par exemple, pour un graphique avec lignes, nous aurions le code suivant :

```
<?php
include ('JpGraph.php');
include ('JpGraph_line.php');
...
// Code utilisant la bibliothèque
...
?>
```

### Création d'un graphique

On crée un graphique en instanciant la classe `Graph`. Vous devez passer en paramètre la largeur et la hauteur de l'image en pixels. La structure de vos fichiers utilisant la JpGraph est la suivante :

```
// ... Inclusion des bibliothèques nécessaires
include ('JpGraph.php');
include ('JpGraph_line.php');

$graph = new Graph($width, $height);
// ... code permettant de fabriquer le graphique souhaité
$graph->Stroke();
```

## Envoi et enregistrement de l'image

Après la création de votre objet `Graph`, vous devez ajouter tout le code permettant de construire votre image et, pour finir, vous calculez et envoyez l'image en utilisant la méthode `Stroke()`. Cette méthode définit toute seule les en-têtes HTTP nécessaires pour déclarer l'image au navigateur.

### Attention

Au risque de se répéter, un script renvoyant une image au navigateur ne doit renvoyer que cela : ni HTML, ni texte, ni même des espaces ou des lignes vides.

Il est également possible de ne pas afficher l'image à l'écran, mais de la sauvegarder :

```
$graph->Stroke('/data/www/images/logo.png' );
```

### Format de l'image

Pour choisir le format de l'image que vous souhaitez créer, il y a deux possibilités. Vous pouvez changer le format par défaut en utilisant une constante :

```
define('DEFAULT_GFORMAT' , 'jpeg');
```

ou définir ce format au niveau de l'image en appelant la méthode `SetImgFormat()` :

```
$graph->img->SetImgFormat( 'jpeg' );
```

## Gérer les polices de caractères

La `JpGraph` permet de travailler, entre autres, sur les polices TrueType. Il convient d'abord de copier vos polices dans un répertoire spécifique et de l'indiquer à la bibliothèque (dans le fichier `JpGraph.php`) :

```
define("TTF_DIR", "/usr/local/fonts/ttf/");
```

Le système de la `JpGraph` est un peu complexe, car pour spécifier la police et le style en général, il faut observer les règles suivantes. Pour une famille de caractères, spécifiez `FF_` avant le nom de la police. Pour un style, spécifiez `FS_` avant le style. Pour la taille (corps), utilisez des valeurs numériques en points.

```
<?php
$graph->title->SetFont( FF_FONT2);
$graph->title->SetFont( FF_FONT2, FS_BOLD);
$graph->title->SetFont( FF_ARIAL);
$graph->title->SetFont( FF_ARIAL, FS_BOLD,24);
?>
```

## Propriétés et méthodes communes

Nous allons voir ici les méthodes communes aux différents types de graphiques. Celles-ci sont contenues dans le fichier `JpGraph.php`, qui contient toutes les informations relatives aux méthodes de base de la bibliothèque. Il est nécessaire de l'inclure dans tous vos fichiers utilisant la `JpGraph`.

Les images sont toutes créées à partir d'un objet de la classe `Graph`. Son constructeur prend en arguments une largeur et une hauteur pour l'image à générer. Les paramètres suivants sont optionnels et permettent de définir le nom du fichier en cache, sa durée de vie et s'il sera utilisé en ligne ou s'il servira juste à calculer une image cachée. Cet objet contient toutes les méthodes pour personnaliser et créer votre graphique.

```
$img = new Graph( $largeur, $hauteur,  
                 $cache='', $vie_cache=0, $enligne=TRUE );
```

### Les propriétés

Chaque graphique dispose de trois types de titre accessibles via les attributs `Graph::title`, `Graph::subtitle` et `Graph::subsubtitle`. Chaque graphique dispose de plus d'une légende avec l'attribut `Graph::legend`.

```
<?php  
$graph = new Graph(300,200);  
$graph->SetTitle('Toto');  
$graph->SetSubtitle('Titit');  
$graph->SetSubsubtitle('tralala');  
$graph->SetLegend('legende');  
?>
```

### Les méthodes

Avec les méthodes communes, vous pouvez :

- ajouter à votre graphique des objets graphiques (lignes, graphes à barres, texte, etc.) avec la méthode `Add()`,
- spécifier la marge d'un graphique avec `Graph::SetMargin()`,
- définir la couleur du graphique avec `Graph::SetMarginColor()`,
- ajouter un ombrage à votre graphique avec `Graph::SetShadow()`,
- effectuer une rotation avec `Graph::SetAngle()`,
- ajouter une image de fond avec `Graph::SetBackgroundImage()`.

## Les graphiques à base de lignes

Le fichier `JpGraph_line.php` contient tous les objets et méthodes permettant de concevoir des graphiques utilisant des lignes. Il faut donc l'inclure après la bibliothèque de base `JpGraph.php` pour pouvoir l'utiliser.



Une fois les fichiers de bibliothèques inclus, il faut créer une instance de l'objet `Graph`, puis définir le type de graphique désiré avec la fonction `SetScale()`. Pour les graphiques basés sur des lignes, on utilise le type `textlin`.

Pour ajouter des points sur notre graphique en lignes, on utilise des objets de la classe `LinePlot`. Son constructeur prend en argument un tableau simple contenant les valeurs utilisées.

On ajoute ensuite l'objet construit à l'objet `Graph` créé plus tôt en utilisant la méthode `Add()`. On peut alors afficher le résultat avec la méthode `Stroke()`. Un exemple de résultat est donné à la figure 25-9.

```
<?php
include ('JpGraph.php');
include ('JpGraph_line.php');

// Il faut mettre des valeurs dans un tableau
// Vous pouvez les récupérer d'une base de données ou autres...
$ydata = array(6,5,25,12,5,10,32,13,5,21);

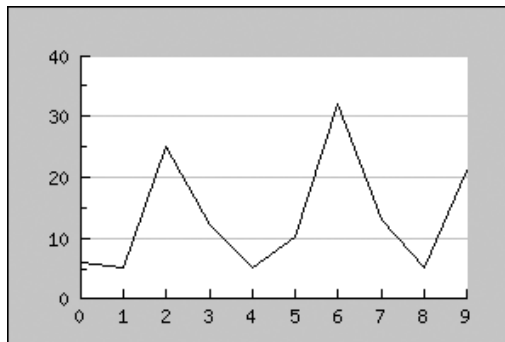
/* On crée l'objet Graph. Ces deux appels sont toujours nécessaires.*/
$graph = new Graph(300,200);
$graph->SetScale('textlin');
// On crée un tracé
$lineplot=new LinePlot($ydata);

// On ajoute ce tracé à l'objet Graph
$graph->Add($lineplot);

// On affiche le graphique
$graph->Stroke();
?>
```

**Figure 25-9**

*Un exemple simple de graphique*



Cet exemple simple nous permet de visualiser à quel point l'utilisation de la JpGraph peut nous permettre de créer des graphiques de très bonne facture sans efforts.

Apprendre les fonctions ci-dessus pourrait vous sembler rébarbatif, mais l'avantage de cette bibliothèque est que ces fonctions se ressemblent et vous permettront par simple déduction logique de concevoir des graphiques de toutes sortes.

### Options avancées

Nous allons maintenant utiliser cet exemple simple pour le personnaliser et le rendre plus complet. Commençons par ajouter un ombrage à ce graphique en utilisant la méthode `SetShadow()` de l'objet `Graph` :

```
■ $graph->SetShadow();
```

Ensuite, définissons un titre à notre graphique :

```
■ $graph->title->Set('Exemple simple');
```

Puis définissons un titre aux axes x et y :

```
■ $graph->xaxis->title->Set('Titre sur X' );  
■ $graph->yaxis->title->Set('Titre sur Y' );
```

Nous pouvons également modifier la couleur de la ligne :

```
■ $lineplot->SetColor('blue');
```

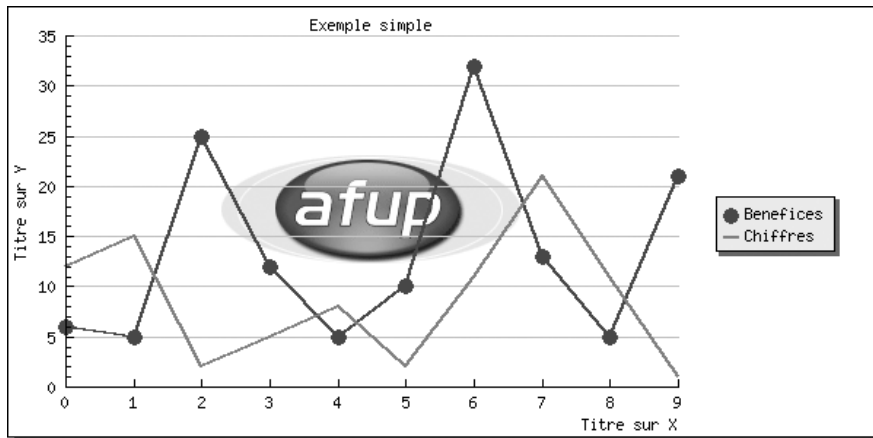
Cet exemple complet est illustré à la figure 25-10 :

```
<?php  
include ('JpGraph.php');  
include ('JpGraph_line.php');  
$ydata = array(6,5,25,12,5,10,32,13,5,21);  
$ydata2 = array(12,15,2,5,8,2,11,21,11,1);  
  
// Création de l'objet Graph  
$graph = new Graph(600,300);  
$graph->SetScale('textlin');  
$graph->SetShadow();  
  
// Définition de l'image de fond  
$graph->SetBackgroundImage('linux2.png',BGIMG_CENTER);  
  
// Ajustons la marge  
$graph->img->SetMargin(40,140,20,40);  
  
// Définissons les légendes et leur position  
$graph->title->Set('Exemple simple');  
$graph->xaxis->title->Set('Titre sur X' );  
$graph->yaxis->title->Set('Titre sur Y' );  
$graph->legend->Pos( 0.05,0.5, 'right' , 'center');
```

```
// On crée les tracés
$lineplot=new LinePlot($ydata);
$lineplot->SetColor('blue');
$lineplot->SetLegend ('Benefices');
$lineplot->SetWeight(2);
$lineplot->mark->SetType(MARK_FILLEDCIRCLE);
$lineplot2=new LinePlot($ydata2);
$lineplot2->SetColor("red");
$lineplot2->SetLegend('Chiffres');
$lineplot2->SetWeight(2);
$graph->Add($lineplot);
$graph->Add($lineplot2);
$graph->Stroke();
?>
```

**Figure 25-10**

*Un exemple plus poussé de graphique*



## Les graphiques en camembert

La JpGraph peut construire un autre type de graphique commun : les camemberts en 2D ou en 3D. La principale différence avec le type précédent basé sur des coordonnées X et Y est l'appel au constructeur de classe principal. Ainsi, on instancie un objet PieGraph au lieu d'un objet Graph.

Il nous faut donc inclure le fichier JpGraph de base et le fichier JpGraph\_pie.php (ou JpGraph\_pie3d.php pour les graphes en trois dimensions).

### Les graphiques en camembert 2D

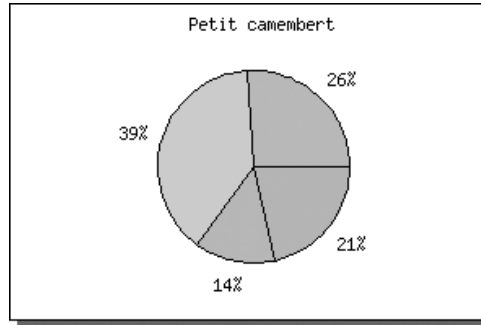
Procédons par un exemple simple. Vous verrez que le schéma est le même que précédemment. Son résultat est donné à la figure 25-11 :

```
<?php
include ('JpGraph.php' );
include ('JpGraph_pie.php');
```

```
$data = array(40,60, 21,33);  
$graph = new PieGraph (300,200);  
$graph->SetShadow();  
$graph->title->Set('Petit camembert');  
$p1 = new PiePlot($data);  
$graph->Add( $p1);  
$graph->Stroke();  
?>
```

**Figure 25-11**

*Un exemple simple  
de graphique en  
camembert*



Remarquons quelques points :

- Par défaut, les couleurs ont été automatiquement assignées.
- Le premier quart commence au degré 0 (à 3 heures).
- Par défaut, les pourcentages sont calculés en fonction des valeurs et inscrits automatiquement.

Il est possible de changer presque tous les paramètres de l'affichage :

- modifier le point de départ avec la fonction `SetStartAngle()`,
- enlever les lignes bordant le camembert avec la fonction `ShowBorder()`,
- changer la couleur de la bordure avec la fonction `SetColor()`,
- changer la taille et la position du camembert avec les fonctions `SetSize()` et `SetCenter()`.

### Les graphiques en camembert 3D

Créer des camemberts en trois dimensions n'est pas plus compliqué que la création des camemberts en deux dimensions. Au lieu de faire appel au constructeur `PiePlot()`, on fera appel au constructeur `PiePlot3D()`.

Regardons à la figure 25-12 le résultat que nous renvoie le même code que ci-dessus pour un graphique en 3D.

```
<?php  
include ('JpGraph.php' );
```

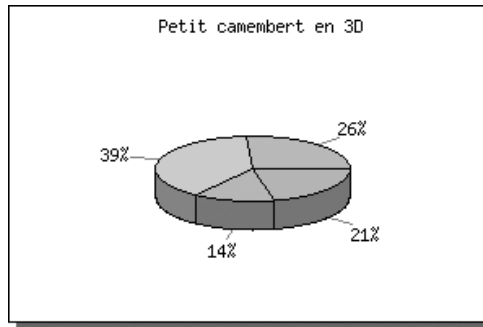
```

include ('JpGraph_pie.php');
include ('JpGraph_pie3d.php');
$data = array(40,60, 21,33);
$graph = new PieGraph (300,200);
$graph->SetShadow();
$graph->title->Set('Petit camembert en 3D');
$p1 = new PiePlot3D($data);
$graph->Add($p1);
$graph->Stroke();
?>

```

**Figure 25-12**

*Camembert à trois dimensions*



Comme en deux dimensions, il est possible de mettre en avant un ensemble de chiffres via la fonction `ExplodeSlice()`, qui permet d'extraire une tranche de l'ensemble.

On peut également modifier l'angle de vue du camembert via la fonction `SetAngle()` de l'objet `PiePlot3d` (voir figure 25-13) :

```

<?php
include ('JpGraph.php' );
include ('JpGraph_pie.php');
include ('JpGraph_pie3d.php');
$data = array(40,60, 21,33);
$graph = new PieGraph (500,300);
$graph->SetShadow();
$graph->title->Set('Petit camembert en 3D');
$graph->img->SetMargin(40,140,20,40);
$p1 = new PiePlot3D($data);


$p1->SetAngle(30);

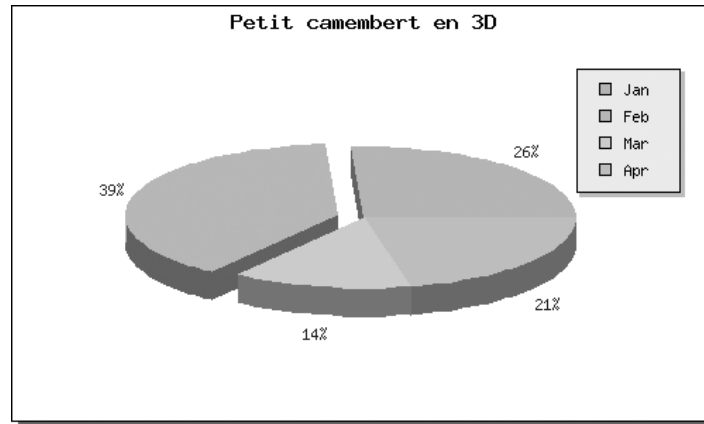


$p1->ExplodeSlice(1);


$graph->SetLegends($gDateLocale->GetShortMonth());
$graph->Add($p1);
$graph->Stroke();
?>

```

**Figure 25-13**  
*Camembert à trois dimensions découpé*



## D'autres types de graphiques

La JpGraph permet de construire un panel exhaustif de graphiques. Parmi ceux-là, on compte les graphiques en étoile, les diagrammes de GANT ou des histogrammes avec gradients.

Les graphiques en étoile sont généralement utilisés pour comparer des valeurs objectives à des valeurs constatées. Il y a autant de rayons que de données dans les séries. Un titre peut être affecté à chaque rayon avec la méthode `SetTitle()`, à laquelle on passe en paramètre un tableau de valeurs.

Le résultat de l'exemple suivant est donné à la figure 25-14 :

```
<?php
include ('JpGraph.php');
include ('JpGraph_radar.php');

// Instanciation de l'objet RadarGraph
$graph = new RadarGraph(500,300, 'auto');
$graph->SetShadow();
$graph->axis->SetWeight(2);

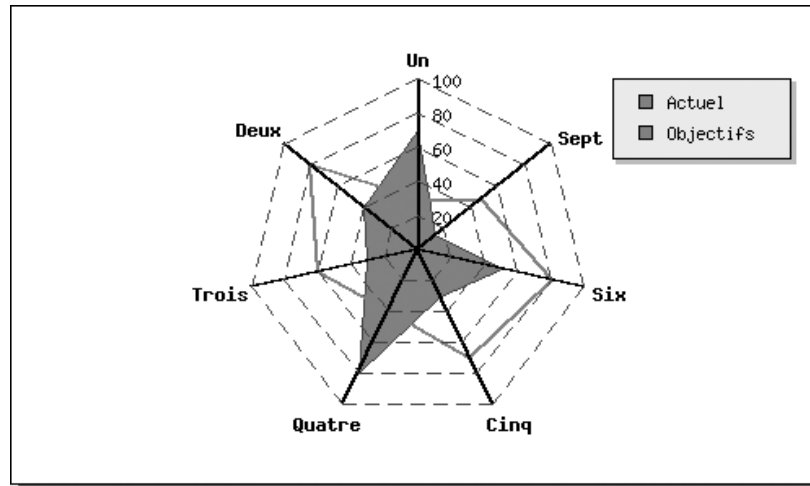
// Définition des lignes
$graph->grid->SetLineStyle('longdashed');
$graph->grid->SetColor('blue');
$graph->grid->Show();
$graph->HideTickMarks();
$titre = array('Un', 'Deux', 'Trois', 'Quatre', 'Cinq', 'Six', 'Sept');
$graph->SetTitles($titre);
```

```
// Création du premier graphique en étoile
$plot = new RadarPlot(array(30,80,60,40,71,81,47));
$plot->SetLegend('Objectifs');
$plot->SetColor('red','lightred');
$plot->SetFill(false);
$plot->SetLineWeight(2);

// Création du second graphique en étoile
$plot2 = new RadarPlot(array(70,40,30,80,31,51,14));
$plot2->SetLegend('Actuel');
$plot2->SetColor('blue','lightred');

// Ajout des lignes au graphique
$graph->Add($plot2);
$graph->Add($plot);
// Affichage du graphique
$graph->Stroke();
?>
```

**Figure 25-14**  
*Un exemple de  
graphique en étoile*



## Étude de cas

### *Redimensionner des images*

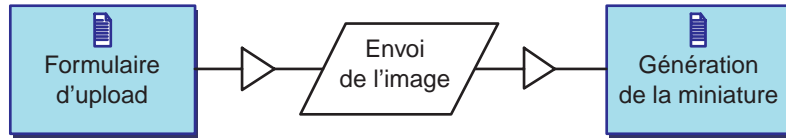
#### Cadre de travail

Dans le cadre de la création d'une galerie de photos, vous avez besoin d'une fonction permettant de créer des miniatures. Cela vous permettra notamment d'économiser de la bande passante et du temps à vos visiteurs lors de leurs visites.

## Architecture du système

Nous allons placer une première page permettant d'envoyer une image au serveur (*upload*). Cette image est récupérée. On fabrique alors une miniature que l'on affiche à l'écran et que l'on sauvegarde. Le processus global peut être vu à la figure 25-15.

**Figure 25-15**  
*Processus de la création de miniatures*



## Réalisation

La réalisation passe par la création de deux fichiers : le premier permet à l'utilisateur de choisir l'image qu'il souhaite envoyer sur le serveur, le second permet de construire une miniature et de sauvegarder les deux images.

### Fichier 1 : formulaire de téléchargement

Cette première page consiste en un formulaire permettant de choisir une image disponible localement (sur le poste du client) et de l'envoyer.

Contrairement aux formulaires simples qui n'envoient que du texte, ce formulaire envoie des données de différents types. Aussi faut-il spécifier un type d'encodage spécifique (`enctype='multipart/form-data'`). Pour plus d'informations, rendez-vous au chapitre 8 traitant des formulaires.

```
<html>
<head><title>Formulaire d'upload</title></head>
<body>

<!-- Création du formulaire
ATTENTION, n'oubliez pas de mettre le enctype="multipart/form-data"
dans la balise form !!
-->

<form enctype="multipart/form-data"
method="post" action="miniature.php">
<p>
Votre image : <br>
<input type="file" name="image" size="20"><br>
<input type="submit" name="envoi" value="ok">
<p>
</form>
</body>
</html>
```



## Fichier 2 : création d'image

À ce stade, on dispose de ce qui nous a été envoyé par le formulaire. Pour rendre ce script plus lisible, nous n'allons pas faire de validation et nous allons considérer que les données reçues sont bonnes.

À quelle taille redimensionner l'image ? En général, on définit une taille fixe ou un pourcentage. Dans notre cas, nous diviserons l'image par cinq (20 %).

```
<?php
// Récupération de l'image envoyée via la superglobale $_FILES[]
$img = imagecreatefromjpeg ($_FILES['image']['tmp_name']);

/*Pour redimensionner, il faut connaître sa largeur et sa longueur.
On les récupère grâce à la fonction getimagesize()*/

// Taille de l'image
$size=getimagesize($_FILES['image']['tmp_name']);

// Largeur de l'image (à l'index 0)
$larg=$size[0];

// Longueur de l'image (à l'index 1)
$long=$size[1];

// On redimensionne l'image
$larg=$larg*20/100;
$long=$long*20/100;

// Image de destination
$img_dest=imagecreatetruecolor($larg,$long);

/*Création de l'image avec les nouvelles dimensions.
La fonction imagecopyresampled copie, redimensionne, ré-échantillonne une image.
Ici, on copie l'image uploadée dans l'image de destination avec
les nouvelles dimensions */
$copy=imagecopyresampled($img_dest,$img,0,0,0,0,$larg,$long,$size[0],$size[1]);

// On spécifie le type de fichier créé
header('Content-Type: image/jpeg');

// Nom du fichier image
$fichier=$_FILES['image']['name'];

/*
Envoi de l'image sur le navigateur
et sauvegarde ce celle-ci dans le dossier image
*/
imagejpeg($img_dest);
imagejpeg($img_dest,'image/'.$fichier);
```

```
/*  
Il ne faut surtout pas oublier de libérer la mémoire après l'affichage.  
Destruction de notre image  
*/  
imagedestroy($img_dest);  
  
?>
```

## Superposer des images

### Cadre de travail

Un de vos clients, photographe, souhaite mettre sur Internet quelques-unes de ses œuvres. Il souhaite indiquer son copyright et empêcher dans la mesure du possible la récupération de ses photographies par des tiers.

### Architecture du système

Nous allons créer une fonction prenant en entrée deux images et retournant une image qui superpose les deux. Cette image finale sera copiée dans un répertoire défini dans le script.

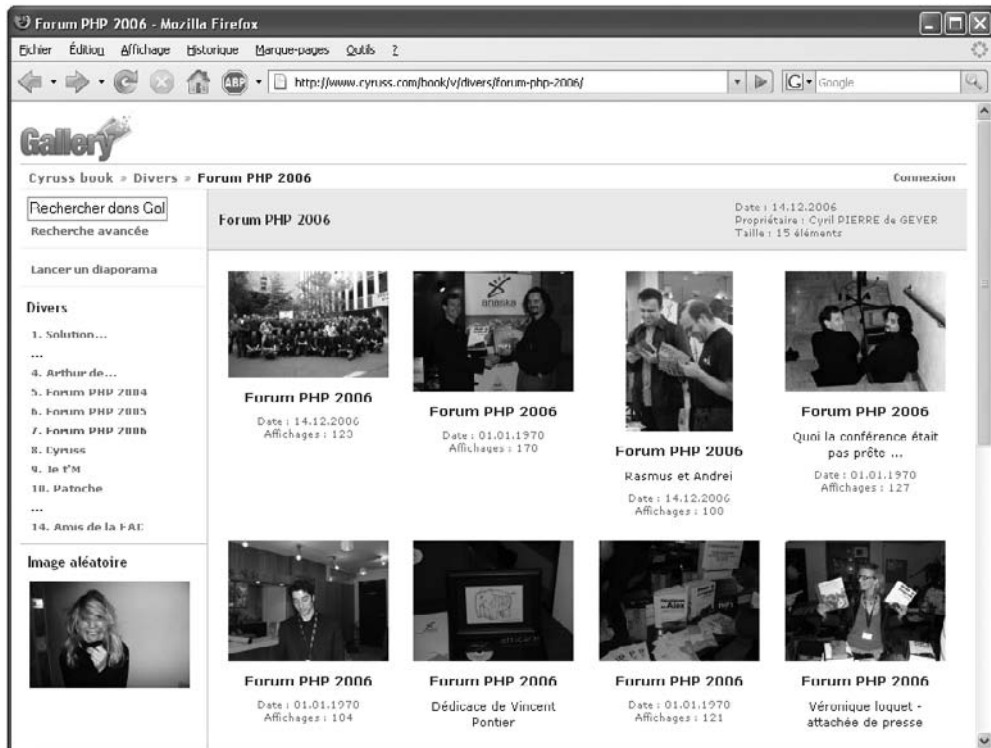


Figure 25-16

Superposition d'images

## Réalisation

```
<?php

/*
Le paramètre img_src est l'image source (ici la photographie).
Le second paramètre img_cop est le copyright à superposer.
*/

function superpose($img_src,$img_cop){

    // Positionnement de l'image principale
    $image_s=imagecreatefromjpeg($img_src);

    // Dimensions de l'image principale
    $larg=imagesx ($image_s);
    $long=imagesy ($image_s);

    // Création de l'image copyright
    $image_c=imagecreatefromjpeg($img_cop);

    // Dimensions de l'image du copyright
    $larg_cop=imagesx ($image_c);
    $long_cop=imagesy ($image_c);

    // On calcule la position sur l'axe des abscisses du copyright
    $x=$larg-$larg_cop;

    // On réalise la superposition
    imagecopymerge($image_s, $image_c, $x, 0, 0, 0, $larg_cop, $long_cop, 100);

    // On spécifie le type de fichier créé
    header('Content-Type: image/jpeg');

    // Envoi de l'image
    imagejpeg($image_s);

    // Sauvegarde de l'image
    imagejpeg($image_s,'image/'.$img_src);

    imagedestroy($image_s);
}

// Appel de la fonction
superpose('test.jpg','logo.jpg');
?>
```

## Expressions régulières

---

Dans le cadre de la réalisation d'une application, il est fréquent d'avoir besoin de travailler sur des chaînes de caractères complexes et changeantes. Parfois ces chaînes ne peuvent pas être repérées simplement avec les fonctions de traitement de chaînes de caractères. C'est là qu'entrent en jeu les expressions régulières.

Les expressions régulières (aussi appelées expressions rationnelles) permettent de définir un modèle et de tenter de l'appliquer à une chaîne de caractères. Il s'agit de donner des instructions à PHP du type « chercher la chaîne de caractères qui commence par un A majuscule ou un B, qui finit par le même caractère, qui contient trois mots parmi les suivants ». Une telle recherche prendrait plusieurs dizaines d'instructions avec les outils de traitements de chaînes classiques, alors qu'il est possible de les traiter avec un motif unique en utilisant les expressions régulières.

Les expressions régulières décrites dans ce chapitre sont des expressions dites compatibles Perl. C'est une version très majoritairement compatible avec l'implémentation faite dans le langage Perl.

### Syntaxe

La syntaxe des expressions régulières peut paraître très complexe si vous vous enfoncez dans les fonctionnalités avancées. Il y aurait probablement de quoi faire un livre dédié.

Nous vous conseillons de tout lire une fois sans vous arrêter pour savoir ce qui existe et où le trouver. Par la suite, ne regardez que ce qui vous sert et sautez les passages qui vous semblent inutilement complexes pour ce que vous voulez faire.

Ceci dit, si vous vous contentez des motifs simples, vous aurez un outil très puissant assez facilement compréhensible. Par exemple, le code suivant recherche si une suite d'au moins 7 chiffres hexadécimaux et au plus 11 est présente dans la variable `$texte` :

```
preg_match('/[0-9a-f]{7,11}/', $texte) ;
```

## Protections et échappements

Dans toute la suite, vous verrez diverses syntaxes qui ont des significations particulières. Pour éviter à ces syntaxes d'être interprétées si vous ne le voulez pas, vous aurez à utiliser un caractère de protection : la barre oblique inverse (caractère `\`).

Notez aussi que les règles habituelles des chaînes de caractères s'appliquent. Ainsi, dans une expression régulière, la chaîne `\n` a une signification particulière (fin de ligne). Si ce que nous recherchons est réellement une barre oblique inverse suivie de la lettre `n`, nous allons devoir la protéger et donc utiliser `\\n` à la place.

Pour affecter cette chaîne à une variable il faut, comme pour toute chaîne de texte PHP, la délimiter par des guillemets. Dans une chaîne entre guillemets, c'est PHP qui interprète la double barre oblique inverse et la convertit en une simple avant de l'utiliser. Pour éviter cette interprétation par PHP, il faut redoubler chaque barre oblique inverse et donc utiliser le code suivant :

```
$recherche = "\\n" ;  
preg_match($recherche, $texte) ;
```

### Note sur les échappements dans les chaînes de caractères

Rappelez-vous bien qu'il faut protéger vos caractères une fois pour le système d'expressions régulières, et une fois pour PHP si vous l'utilisez entre guillemets dans le code. Il en résultera le plus souvent des suites de barres obliques inverses assez impressionnantes ; c'est normal, ne vous en formalisez pas.

Pour éviter d'avoir trop de barres obliques inverses dans vos définitions, vous pouvez utiliser des délimitations de chaînes PHP avec des apostrophes. Entre apostrophes, les barres obliques inverses ne sont pas obligatoirement échappées. Cette syntaxe est beaucoup plus compréhensible sur certains motifs. Elle est à l'inverse gênante pour des motifs complexes : si vous avez un `n` avant une barre oblique inverse, il faut effectivement le doubler comme dans une chaîne entre guillemets. Vous aurez alors à vérifier si l'échappement est nécessaire ou non.

Globalement, les caractères d'échappement et les notations de caractères spéciaux sont les mêmes qu'en Perl et en langage C. Elles sont similaires à celles utilisées par PHP pour le traitement normal des chaînes de caractères.

## Délimitation et présentation

### Syntaxe générale d'une expression

Une expression régulière est constituée de deux parties :

- La chaîne de recherche est la partie qui décrit ce que le moteur doit chercher.

- Les modificateurs permettent de changer quelques options sur la façon dont fonctionne le moteur. Ils sont optionnels et, dans un souci de simplification, nous ne les utiliserons pas dans un premier temps.

### Les délimiteurs

Afin d'opérer la séparation entre la chaîne de recherche et les modificateurs, on entoure la première par des délimiteurs. Le plus souvent on utilise la barre oblique (caractère /), mais n'importe quel caractère non alphanumérique (qui n'est ni une lettre ni un chiffre) autre que la barre oblique inverse (caractère \) est autorisé.

Une expression régulière a donc la forme suivante :

```
| /chainederecherche/modificateurs
```

Vous pouvez aussi utiliser comme délimiteurs une paire d'accolades, de crochets ou de parenthèses. Ces différentes syntaxes sont toutes valides :

```
| {chainederecherche/modificateurs  
| #chainederecherche#modificateurs  
| @chainederecherche@modificateurs  
| [chainederecherche]modificateurs  
| {chainederecherche}modificateurs
```

Quel que soit le délimiteur de fin utilisé, s'il est lui-même présent dans la chaîne de recherche, il devra y être protégé par une barre oblique inverse. Pensez donc à bien choisir votre délimiteur pour en choisir un qui ne soit pas présent dans la chaîne de recherche, et éviter les erreurs.

Dans la suite de ce chapitre, nous utiliserons exclusivement la barre oblique comme délimiteur pour vous simplifier la compréhension.

## Chaîne de recherche simple

Pour les exemples, nous utiliserons la fonction `preg_match()`. Elle prend en premier argument une expression régulière et en second une chaîne de caractères référence. Si l'expression régulière correspond à tout ou partie de la chaîne référence alors la fonction renvoie la valeur `TRUE`, sinon elle renvoie `FALSE`.

Ainsi, le code suivant vérifie la présence de la chaîne « chat » dans « voici un chaton » :

```
<?php  
$texte = 'voici un chaton';  
if ( preg_match('/chat/', $texte) ){  
    echo 'La variable $texte contient la chaîne chat ' ;  
} else {  
    echo 'La variable $texte ne contient pas la chaîne chat ' ;  
};  
?>
```

Pour une chaîne de recherche aussi simple, `preg_match()` fait un travail similaire à celui de `strpos()`.

## Construction d'expression

### Choix entre plusieurs chaînes de caractères

Pour plus de complexité, nous allons chercher une chaîne qui contient soit « chien » soit « chat ». La barre verticale (*pipe*, caractère |) sert de délimiteur entre deux alternatives. Ainsi, chat|chien valide les chaînes contenant au moins un des deux mots.

Les deux portions de code suivantes sont équivalentes :

- avec une expression régulière :

```
<?php
$texte = 'voici un chat';
if ( preg_match('/un chat|un chien/', $texte) ) {
    echo 'un chat ou un chien' ;
} else {
    echo 'ni chien ni chat' ;
}
?>
```

- avec les fonctions classiques :

```
if ( strpos($texte, 'un chat') !== FALSE
    || strpos($texte, 'un chien') !== FALSE
) {
    echo 'un chat ou un chien' ;
} else {
    echo 'ni chien ni chat' ;
}
```

Pensez donc bien que si vous recherchez réellement le caractère |, il vous faut le protéger avec une barre oblique inverse :

```
<?php
$texte = " chat chien " ;
$et = "/chat\\|chien/" ;
preg_match($et, $texte) ; // Renvoie FALSE

$ou = "/chat|chien/" ;
preg_match($ou, $texte) ; // Renvoie TRUE
?>
```

Si vous voulez une alternative uniquement sur une sous-chaîne, vous pouvez la délimiter par des parenthèses. Vous verrez par la suite que ces parenthèses ont d'autres conséquences.

```
<?php
$texte = "j'ai un chat " ;
$recherche = "/j'ai un (chat|chien)/" ;
preg_match($recherche, $texte) ; // Renvoie TRUE
?>
```

## Liste de caractères autorisés

Si c'est entre différents caractères que vous devez chercher des alternatives, il existe alors une syntaxe réduite possible : lister tous les caractères autorisés en les délimitant par des crochets.

Ainsi, `[0123456789]` permet de valider n'importe quel chiffre. Le code suivant cherche la présence d'une heure dans `$texte` :

```
<?php
$texte = 'le chat est revenu à 12h45 ';
$recherche = '/[012][0123456789]h[01245][0123456789]/' ;
echo preg_match($recherche, $texte) ;
// Renvoie TRUE (donc affiche 1) si une heure est présente
?>
```

Généralement, il est long d'énumérer tous les caractères ; il est donc possible de spécifier en une fois toute une liste. Entre crochets, deux caractères séparés par un tiret (caractère -) représentent une liste. Par exemple, `[0-9]` valide n'importe quel chiffre et `[a-f]` les lettres entre a et f comprises.

Voici l'exemple précédent présenté autrement :

```
<?php
$texte = 'le chat est revenu à 12h45 ';
$recherche = '/[0-2][0-9]h[0-5][0-9]/' ;
echo preg_match($recherche, $texte) ;
// Renvoie TRUE (donc affiche 1) si une heure est présente
?>
```

### Remarque

Si vous souhaitez réellement utiliser le tiret comme un caractère possible et non comme un séparateur de liste, il vous faut soit le préfixer par une barre oblique inverse, soit le mettre en dernier dans les crochets.

## Liste de caractères interdits

Vous pouvez aussi définir une classe de caractères par négation. Si vous faites commencer la classe par un accent circonflexe, elle valide tous les caractères non listés. Ainsi, `[^0-5]` valide tout caractère qui n'est pas un chiffre entre 0 et 5 compris.

Attention toutefois à bien comprendre comment fonctionne la recherche ! L'expression `/[^a]/` peut valider une chaîne qui contient la lettre a. Il lui suffit qu'un seul caractère « non a » soit trouvé pour valider. Il ne s'agit pas d'une exclusion des caractères concernés, mais d'une recherche de caractères autres.



## Classes de caractères prédéfinies

Pour simplifier l'utilisation, certaines alternatives sont déjà construites. Ainsi, `[:blank:]` valide n'importe quel caractère blanc et `[:digit:]` n'importe quel chiffre décimal.

Ces classes de caractères sont entourées par un crochet et un deux-points. Voici la liste des plus utilisés avec leur description :

- `lower` : lettres minuscules ;
- `upper` : lettres majuscules ;
- `alpha` : les caractères alphabétiques ;
- `digit` : les chiffres décimaux (équivalent de `[0-9]`) ;
- `xdigit` : chiffres hexadécimaux ;
- `alnum` : chiffres et lettres ;
- `ascii` : caractères 0 à 127 ;
- `blank` : espace ou tabulation ;
- `cntrl` : caractère de contrôle ;
- `punct` : caractères imprimables sauf lettres et chiffres ;
- `graph` : tous les caractères imprimables sauf l'espace ;
- `print` : tous les caractères imprimables avec l'espace ;
- `space` : espace blanc.

De même que dans les groupes de caractères génériques, il est possible de fonctionner par négation en préfixant le nom de la classe par un accent circonflexe (par exemple `[:^alpha:]`).

```
<?php
$texte = 'abcdefgh 0123456789' ;

preg_match('/:digit:]{7,11}/', $texte) ;
// TRUE : il existe une suite de 10 chiffres

preg_match('/:^alpha:]{7,11}/', $texte) ;
// TRUE : il existe une suite de 10 caractères
// qui ne sont pas des lettres alphabétiques
?>
```

## Codes spéciaux prédéfinis

En plus des classes de caractères, le moteur nous met à disposition plusieurs codes utilisables simplement :

- Le point correspond à n'importe quel caractère sauf un caractère de fin de ligne (si le modificateur `s` est activé, alors il peut aussi être un caractère de fin de ligne).

- `\d` correspond à un chiffre décimal quelconque.
- `\w` est un caractère pouvant constituer un mot.
- `\s` est un espace blanc.

Ces trois codes ont leur équivalent majuscule `\D`, `\W` et `\S`, qui consistent en une négation (`\D` valide n'importe quel caractère sauf un chiffre).

```
<?php
$texte = 'abcdefgh 0123456789' ;

preg_match('/\d{7,11}/', $texte) ;
// TRUE : il existe une suite de 10 chiffres

preg_match('\D{7,11}/', $texte) ;
// TRUE : il existe une suite de 10 caractères
// qui ne sont pas des chiffres
?>
```

## Gestion des occurrences multiples

### Nombre d'occurrences fixe

Si vous faites suivre un caractère d'un nombre entre accolades, le moteur d'expressions régulières l'interprète comme une répétition. Ainsi, `abcd{3}` est équivalent à `abcdddd` (la dernière lettre est répétée trois fois).

```
<?php
$texte = "abcdddd " ;
$recherche = "/d{3}/" ;
preg_match($recherche, $texte) ; // Renvoie TRUE
// car la chaîne contient trois fois "d" à la suite
?>
```

Comme nous le verrons plus loin, il est également possible de définir un nombre d'occurrences sur plusieurs caractères en groupant une suite de caractères.

### Nombre d'occurrences délimité

Vous pouvez laisser une liberté dans la répétition, par exemple chercher une chaîne répétée entre 3 et 5 fois. Pour cela, mettez dans les accolades le deuxième nombre à la suite du premier, en les séparant par une virgule : `abcd{3,5}` valide « `abcdddd` », « `abcdddd` » et « `abcdddd` ».

```
<?php
$texte = "abcdddddef " ;
$recherche = "/abcd{3,9}ef/" ;
preg_match($recherche, $texte) ; // Renvoie TRUE
?>
```

## Nombre d'occurrences semi-délimité

Si vous omettez le minimum, il sera considéré comme nul et si vous omettez le maximum, il sera considéré comme infini. `abcd{2,}` cherche un `d` répété au moins deux fois. `abcd{,10}` cherche un `d` répété au plus 10 fois. Attention, dans ce cas le motif de recherche valide aussi la chaîne « `abc` » (lettre `d` répétée 0 fois).

## Syntaxe réduite

Le symbole `+` est le code réduit équivalent à `{1,}` (au moins une fois). Le symbole `*` est le code réduit équivalent à `{0,}`. Quand il ne suit pas une parenthèse ouvrante, un astérisque, le symbole d'addition ou un point d'exclamation, le point d'interrogation est l'équivalent de `{0,1}`.

```
<?php
$texte = "abcddddef " ;
$recherche = "/abcd+efg?/" ;
preg_match($recherche, $texte) ; // Renvoie TRUE
?>
```

## Capture gloutonne ou non gloutonne

La capture d'une expression de répétition (par exemple `.*`) est dite gloutonne, c'est-à-dire qu'elle va essayer de valider autant de caractères que possible.

Ainsi, le motif `/A.*A/` appliqué à la chaîne « `AwAxyzA` » capture l'ensemble de la chaîne (jusqu'à la lettre suivant le `z`), et pas seulement les trois premiers caractères. Le moteur essaie de valider le plus de caractères possibles, puis revient en arrière quand il voit que le reste de l'expression ne correspond pas.

Il est possible de demander aux répétitions de ne pas être gloutonnes (c'est-à-dire de capturer le moins de caractères possible ; dans notre cas, le motif n'utiliserait que les trois premiers caractères). Pour cela, il suffit d'ajouter un point d'interrogation après le symbole de répétition. Notre motif en exemple serait donc `/A.*?A/`.

## Répétition de sous-chaînes

Répéter un caractère n'est généralement pas suffisant. Vous pouvez aussi répéter une chaîne complète ; il faut alors la délimiter par des parenthèses :

```
<?php
$texte = "123465454545645 " ;
$recherche = "/(45){3,5}/" ;
preg_match($recherche, $texte) ; // Renvoie TRUE
// car la chaîne contient 3 "45" consécutifs
?>
```

## Assertions

Une assertion est une façon de prédire une situation, par exemple de prédire qu'un caractère est le dernier d'une ligne. Si ce n'est pas vrai, alors l'expression est refusée, sinon elle est validée. Par exemple, `\w+(?=;)` s'assure qu'un mot est suivi d'un point-virgule (l'assertion est le motif `(?=;)`).

Une assertion ne « consomme » pas de caractères, c'est-à-dire qu'elle ne sert qu'à faire une vérification. Ainsi, en considérant que `(?=b)` veut dire « suivi par la lettre b », le motif `a(=b)c` ne valide pas la chaîne « abc » : il s'agit bien d'un « a suivi par un b » mais la lettre suivante est un b, pas un c.

### Imposer les premiers et derniers caractères

Par défaut, la recherche est appliquée à toute la chaîne. L'expression régulière se contente de vérifier la présence n'importe où dans la chaîne de référence.

Vous pouvez forcer la recherche à commencer ou finir avec le premier ou dernier caractère de la chaîne référence. Si le motif de recherche commence par `\A`, alors la chaîne recherchée doit absolument commencer au premier caractère de la chaîne référence. Si le motif de recherche finit par `\z` alors la chaîne recherchée doit absolument finir par le dernier caractère de la chaîne référence :

```
<?php
$texte = "abcdefghij" ;

preg_match("/\Aabc/", $texte) ; // Renvoie TRUE
preg_match("/\Adef/", $texte) ; // Renvoie FALSE
// Le texte n'est pas au début de la chaîne référence

preg_match("/hij\z/", $texte) ; // Renvoie TRUE

preg_match("/def\z/", $texte) ; // Renvoie FALSE
// Le texte n'est pas à la fin de la chaîne référence

preg_match("/def/", $texte) ; // Renvoie TRUE
preg_match("/\Aabcdefghij\z/", $texte) ; // Renvoie TRUE
preg_match("/\Adef\z/", $texte) ; // Renvoie FALSE
?>
```

Le symbole `^`, quand il ne suit pas une ouverture de parenthèses ou de crochets, peut être utilisé à la place de `\A`. (ce comportement n'est plus vrai si le modificateur `m` est activé).

#### Note

Lorsque de plus le modificateur `D` est activé, le symbole `$` perd son sens originel et peut être utilisé à la place de `\z`. Nous vous recommandons toutefois de ne pas utiliser cette possibilité car elle pourrait induire en erreur lors d'une future relecture.

L'exemple précédent donnerait :

```
<?php
$texte = "abcdefghij" ;

preg_match("/^abc/", $texte) ; // Renvoie TRUE
preg_match("/hij$/", $texte) ; // Renvoie TRUE
?>
```

### Délimitation des lignes

En ajoutant `\Z` après un caractère, vous demandez au moteur de ne valider le caractère que s'il est directement suivi par une fin de ligne ou par la fin de la chaîne référence.

```
<?php
$texte = "abcde\nfghij" ;
preg_match("/abc\\Z/", $texte) ; // Renvoie FALSE
preg_match("/cde\\Z/", $texte) ; // Renvoie TRUE
preg_match("/hij\\Z/", $texte) ; // Renvoie TRUE
?>
```

#### Attention

Le `\Z` délimite une fin de ligne quand il est en majuscule. Si vous utilisez une lettre minuscule il ne validera que la fin de la chaîne référence, comme vu précédemment

Le symbole `$`, quand il ne suit pas une ouverture de parenthèses ou de crochets, peut être utilisé à la place de `\Z`. (ce comportement n'est plus vrai si le modificateur `D` est activé).

#### Note

Lorsque de plus le modificateur `m` est activé, le symbole `^` perd son sens originel et peut être utilisé pour valider aussi un début de ligne (pas uniquement le début de la chaîne référence). Nous vous recommandons toutefois de ne pas utiliser cette possibilité car elle pourrait induire en erreur lors d'une future relecture.

### Délimitation des mots

Vous pouvez vérifier la présence d'une délimitation de mot (passage d'un caractère `\w` à `\W` ou l'inverse) avec le code `\b`. Inversement `\B` valide uniquement s'il n'est pas sur une délimitation de mot.

```
<?php
$texte = "abcde fghij" ;
preg_match("/d\\B\\e\\b/", $texte) ; // Renvoie TRUE
// d n'est pas à la fin d'un mot
// e l'est
?>
```

Le `\b` permet en fait de vérifier des événements. On peut considérer deux classes de caractères : `\w` (caractères d'un mot) et `\W` (caractères autres que ceux d'un mot, c'est-à-dire ponctuation et espacement). Dans ce cas, `\b` veut dire « le caractère précédent et le suivant appartiennent à des classes différentes » et `\B` veut dire « le caractère précédent et le suivant appartiennent à la même classe ».

### Assertion sur la suite

Grâce aux assertions, il est possible de faire une hypothèse sur ce qui suit la capture. Une assertion positive (vérifier que ce qui suit est bien ce qui est prévu) se fait en l'entourant par `(?= et )`. Une assertion négative se fait en l'entourant par `(?! et )`.

Par exemple, la recherche `/chat(?!on)/` valide une chaîne contenant la chaîne « chat » non suivie par « on ».

```
<?php
preg_match("/chat(?=on)/", "chaton") ; // Renvoie TRUE
preg_match("/chat(?!on)/", "chaton") ; // Renvoie FALSE
preg_match("/chat(?=on)/", "chatière") ; // Renvoie FALSE
?>
```

### Assertion sur ce qui précède

Il est aussi possible de faire une assertion sur les caractères qui précèdent. Le code est `(?<=assertion)` pour les assertions positives et `(?<!assertion)` pour les assertions négatives.

```
<?php
preg_match("(?<=petit )chaton/", "petit chaton"); // Renvoie TRUE
preg_match("(?<!petit )chaton/", "petit chaton"); // Renvoie FALSE
?>
```

Les assertions arrière ont toutefois une limitation puisque la taille doit être identique pour toutes les alternatives. La seule exception est pour les alternatives du niveau supérieur. Ainsi `(?<=ab(c|de))` est interdit car on ne connaît pas la taille utilisée. Il aurait fallu écrire `(?<=abc|abde)`.

## Captures

### Faire une capture

Par défaut, quand le moteur d'expressions régulières rencontre un couple de parenthèses dans le motif de recherche, il enregistre tout ce qui est validé dans cette parenthèse. Ainsi, dans le motif `/chat(\w+)/` appliqué à « chaton tout petit », le moteur enregistrerait « on » en mémoire. De telles parenthèses sont dites « capturantes ».

### Empêcher une capture

Pour empêcher la capture d'une parenthèse, il vous suffit d'ajouter `?:` juste après l'ouverture. Dans le motif `/chat(?:\w+)/`, rien n'est capturé. De telles parenthèses sont dites « non capturantes ».

## Utiliser le résultat d'une capture

Ces valeurs sauvegardées sont réutilisables avec la notation qui est appelée référence arrière. Il s'agit de nombres précédés par une barre oblique inverse. Le code `\1` fait référence au contenu de la première parenthèse, `\2` à la deuxième, etc. De plus, certaines fonctions permettent de récupérer les différentes valeurs sauvegardées dans un tableau.

### Note

Comme décrit dans l'encadré en haut de ce chapitre, si la syntaxe `\1` est utilisée dans la déclaration de chaîne entre guillemets, elle devra être écrite `\\1`.

Les parenthèses capturent par ordre d'apparition dans le motif de recherche. Quand des parenthèses sont imbriquées, celle de niveau supérieur se retrouve en premier.

## Exemples d'utilisation des captures

Il y a trois utilisations possibles pour les captures :

- Pour spécifier des répétitions dans un même motif – Par exemple, pour spécifier que la lettre A doit être entourée par deux caractères identiques, il est possible de donner le motif `/(. )A\1/`. Ainsi, le caractère validé par le caractère *joker* (le point) est capturé, puis répété après la voyelle.
- Pour opérer des remplacements sélectifs – Certaines fonctions vous permettent d'utiliser les expressions régulières pour faire des remplacements. Ainsi, une recherche avec un motif `/un( .* ) chat/` et un remplacement `deux \1s chiens` remplacerait « un petit chat » par « deux petits chiens ».
- Pour récupérer des informations avec des motifs de recherche complexes – Ainsi, pour récupérer les adresses électroniques mises en lien dans une page, il suffit d'une expression régulière qui fasse une capture sur l'adresse elle-même. C'est aussi utile pour récupérer plusieurs informations en une fois avec un même motif complexe.

## Modificateurs

Jusqu'à présent, nous avons laissé de côté les modificateurs ; il est temps de s'en servir. Un modificateur est un paramètre supplémentaire modifiant le fonctionnement de l'expression.

Ces modificateurs se positionnent juste après le délimiteur de fin :

■ `/expression/délimiteur`

Voici la liste des modificateurs utilisables :

- `i`

La recherche ne prend pas en compte la casse (différence majuscules et minuscules).

- m  
Recherche multiligne, voir les paragraphes concernant les assertions `^` et `$`.
- s  
Le caractère *joker* (`.`) peut être utilisé pour valider un caractère de fin de ligne.
- D  
Fin de chaîne seulement, voir les paragraphes concernant les assertions `^` et `$`.
- U  
L'expression utilise des répétitions non gloutonnes.
- u  
La chaîne de recherche est en UTF-8.

### Fonctions de rappel

Il existe un autre modificateur, propre à PHP. Il s'agit de la lettre `e`. Quand il est utilisé dans une fonction de remplacement, le remplacement a lieu puis le texte résultat est évalué en tant que code PHP.

En fournissant un remplacement de type `mafonction('\1')`, vous pouvez associer directement une action à chaque chaîne capturée.

### Modificateurs locaux

Il est possible de mettre certains modificateurs pour des sous-parties de l'expression de recherche, par exemple pour définir qu'un mot particulier doit être recherché sans vérifier la casse.

Cette option n'est possible que dans des parenthèses. Pour des parenthèses non capturantes, il faut insérer le modificateur entre le point d'interrogation et les deux points : `(?i:expression)`. Pour des parenthèses capturantes, vous pouvez ajouter des parenthèses internes contenant un point d'exclamation et les modificateurs : `((?i)expression)`.

## Les fonctions

Il n'existe que quelques fonctions qui marchent avec les expressions régulières. Elles suffisent normalement à faire l'essentiel des traitements de texte.

### Chercher une correspondance

La fonction la plus usitée en expressions régulières est probablement la recherche. Son but est double : vérifier la correspondance d'une chaîne de caractères avec un masque et récupérer en une fois des sous-parties de la chaîne complexe à extraire.

L'utilisation basique de la fonction `preg_match()` prend deux paramètres : l'expression régulière et la chaîne de caractères où l'appliquer. Elle renvoie la valeur `TRUE` si la chaîne



de caractères valide l'expression (c'est-à-dire si la chaîne recherchée a été trouvée) et FALSE sinon.

```
preg_match('/chat/i', 'voilà un Chaton') ; // Renvoie TRUE
```

### Récupérer des sous-chaînes

En fournissant une variable en troisième paramètre, les captures seront retournées en tableau dans cette variable. Le texte total capturé par l'expression est mis à l'index 0, le contenu de la première parenthèse capturant à l'index 1, la deuxième à l'index 2 et ainsi de suite.

```
preg_match('/(c)h(.)t/i', 'voilà un Chaton', $resultat) ;  
// Renvoie TRUE  
echo $resultat[0] ; // Renvoie Chat  
echo $resultat[1] ; // Renvoie C  
echo $resultat[2] ; // Renvoie a
```

Si la constante `PREG_OFFSET_CAPTURE` est utilisée comme quatrième paramètre, les positions dans la chaîne source de chaque capture sont retournées. Ainsi, pour chaque capture, au lieu de renvoyer directement la chaîne capturée, PHP retourne un tableau avec la chaîne capturée à l'index 0 et la position dans la chaîne initiale à l'index 1.

### Rechercher à partir d'une certaine position

Vous pouvez fournir une position de caractère en cinquième paramètre à `preg_match()`. PHP ne lance alors l'expression régulière qu'à partir de cette position. Vous avez la possibilité d'ignorer la partie de la chaîne qui a déjà été traitée.

### Rechercher sur plusieurs sources

Pour vérifier simplement en une fois la présence du motif de recherche dans plusieurs chaînes de caractères, vous pouvez utiliser la fonction `preg_grep()`. Elle prend en paramètres l'expression régulière à rechercher et un tableau de chaînes de caractères, puis renvoie un tableau identique, mais ne contenant que les éléments pour lesquels on a trouvé le motif recherché.

### Rechercher toutes les occurrences

La fonction `preg_match_all()` fonctionne de manière similaire à `preg_match()`, mais quand cette dernière s'arrête à la première concordance, la première recherche le motif autant de fois que possible. Si, par exemple, on a un motif qui recherche et capture les liens HTML, `preg_match()` ne voit que le premier lien, tandis que `preg_match_all()` les renvoie tous. La valeur de retour de cette fonction est un entier indiquant le nombre de concordances trouvées.

Après une capture, les recherches suivantes commencent à la fin de la capture précédente. Ainsi le masque `/aaa/` utilisé sur la chaîne « aaaaaa » ne renvoie que deux résultats et non quatre.

Le troisième argument de la fonction, au lieu d'être une liste des différentes captures de la première correspondance, est un tableau à deux dimensions. Pour déterminer l'ordre des dimensions, si on classe d'abord par la correspondance ou par l'index de la parenthèse capturante, on utilise un quatrième paramètre :

- S'il contient la constante `PREG_SET_ORDER`, l'index 0 contient la liste des différentes captures (par les parenthèses) de la première correspondance trouvée, l'index 1 la liste pour la deuxième correspondance, etc.
- S'il contient la constante `PREG_PATTERN_ORDER`, l'index 0 contient la liste des correspondances complètes trouvées, l'index 1 la liste des captures de la première parenthèse pour les différentes correspondances, l'index 2 la liste des captures de la deuxième parenthèse, etc.

Comme pour `preg_match()`, le dernier paramètre (optionnel) sert à indiquer dans la chaîne la position à partir de laquelle commencer les recherches.

```
<?php
$texte = "abac" ;
$cherche = "/a(.)/" ;

preg_match_all($cherche, $texte, $result, PREG_SET_ORDER) ;
echo $result[0][0] ; // Renvoie ab
echo $result[0][1] ; // Renvoie b
echo $result[1][0] ; // Renvoie ac
echo $result[1][1] ; // Renvoie c

preg_match_all($cherche, $texte, $result, PREG_PATTERN_ORDER) ;
echo $result[0][0] ; // Renvoie ab
echo $result[0][1] ; // Renvoie ac
echo $result[1][0] ; // Renvoie b
echo $result[1][1] ; // Renvoie c
?>
```

## Faire des remplacements

Nous avons parlé plus haut d'utiliser les parenthèses capturantes pour faire des remplacements. Il est possible, grâce à la fonction `preg_replace()`, de remplacer ce qui est trouvé par l'expression régulière.

La fonction prend en premier paramètre l'expression de recherche, en deuxième la chaîne de remplacement et en troisième la chaîne où faire les remplacements :

```
<?php
$texte = 'un chien dans la rue' ;
$regex = '/chien/' ;
$remplace = 'chat' ;
echo preg_replace($regex, $remplace, $texte) ;
// Affiche un chat dans la rue
?>
```

## Réutilisation des chaînes capturées

Il est possible de réutiliser les captures faites par les parenthèses à l'aide de la syntaxe `\x` où `x` est le numéro de la parenthèse dont le contenu est à remplacer.

```
<?php
$texte = 'un chien dans la rue' ;
$regex = "/un (\\w+) dans la rue/" ;
$remplace = "deux \\1s sur le banc" ;
echo preg_replace($regex, $remplace, $texte) ;
// Affiche deux chiens sur le banc
?>
```

Une syntaxe alternative à la barre oblique inverse est le symbole dollar. Attention toutefois, quel que soit le symbole utilisé, pensez à le protéger par une barre oblique surnuméraire s'il est utilisé dans une chaîne de caractères définie par des guillemets dans PHP.

```
<?php
$texte = 'un chien dans la rue' ;
$regex = "/un (\\w+) dans la rue/" ;
$remplace = "deux \\$1s sur le banc" ;
echo preg_replace($regex, $remplace, $texte) ;
// Affiche deux chiens sur le banc
?>
```

Par défaut, `\12` représente donc la douzième parenthèse capturante. Si vous avez besoin d'utiliser la première capture suivie du chiffre 2, vous pouvez entourer le numéro de la parenthèse par des accolades et utiliser le dollar en place de la barre oblique inverse. Nous aurons donc `$(1)1` dans notre exemple).

## Limitation des remplacements

Par défaut, `preg_replace()` remplace toutes les occurrences trouvées. Vous pouvez limiter le nombre de remplacements en ajoutant le maximum en quatrième paramètre lors de l'appel.

```
<?php
$texte = '1234567890' ;
$cherche = './.' ;
$remplace = 'X' ;

echo preg_replace($cherche, $remplace, $texte) ;
// Affiche XXXXXXXXXX

echo preg_replace($cherche, $remplace, $texte,3) ;
// Affiche XXX4567890
?>
```

## Rechercher sur plusieurs textes

Si à la place du texte référence, vous fournissez une liste de textes dans un tableau, le remplacement est fait sur chaque chaîne, l'une après l'autre. La fonction retourne alors la liste des chaînes résultantes dans un tableau.

```
<?php
$texte = array( '12', 'ab' );
$cherche = '/.(.)/' ;
$remplace = "X\\1" ;

$resultat = preg_replace($cherche, $remplace, $texte) ;
echo $resultat[0] ; // Renvoie X2
echo $resultat[1] ; // Renvoie Xb
?>
```

## Rechercher plusieurs motifs

Si vous fournissez un tableau d'expressions régulières au lieu d'une seule, `preg_match()` les cherche une à une pour faire le remplacement spécifié.

```
<?php
$texte = '1234567890' ;
$cherche = array( '/12/', '/56/' );
$remplace = 'XX' ;

echo preg_replace($cherche, $remplace, $texte) ;
// Affiche XX34XX7890
?>
```

Rechercher plusieurs motifs en une fois est plus rapide que faire plusieurs appels à la fonction de remplacement.

## Faire plusieurs remplacements

Il est de même possible de faire plusieurs remplacements en une seule fois. Il suffit alors d'envoyer des listes pour les expressions régulières et chaînes de remplacement.

La première expression régulière est utilisée avec la première chaîne de remplacement et ainsi de suite. Si le tableau contenant les chaînes de remplacement est plus petit que celui des expressions régulières, il est complété avec des chaînes vides.

```
<?php
$texte = '1234567890' ;
$cherche = array( '/12/', '/56/' );
$remplace = array( '/__/', '/**/' );

echo preg_replace($cherche, $remplace, $texte) ;
// Affiche /__34/**7890
?>
```

## Utiliser des fonctions de rappel

Si vous utilisez le modificateur `e` vu plus haut dans les expressions, la chaîne de remplacement sera interprétée en PHP.

Il existe toutefois une fonction plus simple pour utiliser des fonctions de rappel (*callback* en anglais). La fonction `preg_replace_callback()`, au lieu de remplacer directement les chaînes trouvées, envoie le tableau contenant les captures à une fonction PHP, et utilise le résultat pour faire le remplacement.

```
$texte = "Date:31/12/2003 \n Date:01/01/2004";
$recherche = "/\s*Date:([0-3]\d).([0-2]\d).(20\d\d)\s*/" ;

function remplace($capture) {
    // $capture[0] contient toute la chaîne
    // $capture[1] contient le jour
    // $capture[2] contient le mois
    // $capture[3] contient l'année sur deux chiffres
    return "Date: 20$capture[3]-$capture[2]-$capture[1]";
}

echo preg_replace_callback($recherche, 'remplace', $texte) ;
// Affiche Date: 2003-12-31 et Date: 2003-01-01
// sur deux lignes
```

### Note

Comme à chaque fois que PHP vous demande une fonction de rappel, vous pouvez à la place utiliser une méthode d'objet. Il vous faut alors envoyer un tableau avec l'objet référence à l'index 0 et le nom de la méthode à utiliser à l'index 1. Si vous utilisez une méthode statique, vous pouvez spécifier le nom de la classe à la place de l'objet.

## Échappement et protections

Afin de vous aider à manier les chaînes de caractères dans les expressions régulières, vous pouvez utiliser la fonction `preg_quote()`. Elle analyse la chaîne passée en argument et échappe (en préfixant par une barre oblique inverse) tous les caractères spéciaux des expressions régulières (`. \ + * ? [ ^ ] $ ( ) { } = ! < > | :`). Ainsi, la chaîne retournée peut être utilisée telle quelle dans une expression régulière, sans qu'elle ait une signification particulière.

Le délimiteur d'expression n'est pas échappé par défaut, car il est au choix de l'utilisateur. Afin de résoudre ce problème, vous pouvez donner le délimiteur utilisé en second paramètre, et il sera ajouté aux caractères à protéger.

## Performances

Les expressions régulières sont très pratiques, car elles permettent de chercher et d'utiliser des motifs très complexes. Cependant, mal employées, elles peuvent se révéler très consommatrices en ressources pour le serveur.

D'un point de vue général, si vous pouvez vous passer des expressions régulières et faire une recherche à l'aide de quelques lignes simples utilisant `strpos()`, `substr()` et les fonctions classiques de traitement de chaînes, vous y gagnerez probablement en performance.

Pour certains motifs, il serait toutefois trop long ou trop complexe de passer par une série de fonctions classiques. Utiliser des expressions régulières peut amener à une recherche simple et rapide, à condition d'en comprendre le fonctionnement.

### *Fonctionnement du moteur*

**Note**

L'explication est grossière, ne rend pas compte des détails ou des optimisations du moteur, et peut entraîner des approximations. Il s'agit juste d'avoir une idée assez générale du fonctionnement, pas de l'avoir en détail avec exactitude.

On peut essayer de faire un aperçu avec la chaîne de référence «`_1234567890_`» et le motif de recherche `/1(2346|2345)/`.

Le moteur n'a pas le choix pour la première lettre du motif ; il cherche un 1. Il va donc se positionner sur le deuxième caractère de la chaîne et le valider.

Il passe alors à la lettre suivante dans le motif. Il a là une alternative (2346 ou 2345), il essaie en premier la possibilité (2346). Il essaie et valide le chiffre 2, puis le 3 et le 4. Une fois arrivé au chiffre 4, il cherche le 6 mais c'est un 5 qui est présent : le motif ne correspond pas.

Le moteur va donc faire machine arrière jusqu'à la dernière alternative qu'il avait laissée de côté (2345). Il revient donc en arrière de trois caractères et recommence, pour cette fois-ci réussir.

Cette petite démonstration met en œuvre la démarche globale du moteur : il essaie à chaque fois la première possibilité qui correspond jusqu'à que cela n'aille plus ; il revient alors en arrière pour réessayer une autre possibilité. On peut d'ailleurs remarquer ici la nature gloutonne des expressions régulières : on avance tant qu'on peut, quitte à revenir en arrière par la suite en cas d'échec.

## Stratégies

Ici, la démarche est simple, mais si la quantité d'alternatives est importante, le moteur peut essayer une grande quantité de combinaisons avant de trouver la bonne.

Il est donc important d'essayer de faciliter le parcours du moteur d'expressions régulières. Plusieurs stratégies sont possibles, suivant les probabilités d'apparitions et les motifs à rechercher.

Si, par exemple, vous avez une alternative entre une très longue suite et une très courte, vous avez intérêt à lui faire essayer la courte en premier pour éviter qu'il ne teste la longue pour rien.

Essayez de regrouper au maximum les alternatives pour éviter qu'il ne teste trop de fois les même parties. Dans notre exemple, le masque `/1234(5|6)/` aurait été plus adapté.

Évitez d'utiliser des masques trop génériques quand ce n'est pas nécessaire. Un motif comme `.*` va faire parcourir toute la chaîne par le moteur puis le faire revenir sur ses pas caractère par caractère jusqu'à trouver le bon.

Essayez aussi de faire en sorte que le cas le plus probable se résolve de la manière la plus simple.

## Boucles infinies

Un autre risque est de définir des boucles infinies. Le fonctionnement glouton du moteur peut amener des surprises. Par exemple, un motif contenant `(\d?)*` peut faire une boucle interminable.

En effet, avec par exemple la chaîne référence «a», le moteur va d'abord essayer de voir si «a» est un chiffre. Ce n'en est pas un, mais comme le chiffre est optionnel (point d'interrogation), le moteur continue en considérant qu'il vient de rencontrer un chiffre répété zéro fois.

Le problème, c'est que ce chiffre répété zéro fois est lui-même encapsulé dans un groupe pouvant intervenir un nombre illimité de fois. Sans optimisations et astuces venant du moteur, il pourrait tourner en boucle en trouvant un nombre infini de chiffres répétés zéro fois.

Heureusement, certaines syntaxes sont automatiquement repérées par le moteur et engendrent des erreurs au bout de quelques boucles ; d'autres sont comprises et le moteur évite de boucler dedans. Il convient toutefois de faire attention à ce que le moteur peut faire comme trajet et éviter de lui faire répéter des boucles qui peuvent être vides.

# 27

## Sécurité

---

Avec la professionnalisation de l'utilisation de PHP, la sécurité est un concept qui vient désormais souvent à l'esprit pendant la conception d'une application. Il y a encore peu de temps, cet élément était trop souvent oublié.

Pourtant, même aujourd'hui, le sujet reste largement méconnu des développeurs et des architectes. On ne pense trop souvent qu'à la sécurité des logiciels et on laisse des failles de sécurité à cause d'une mauvaise configuration ou d'une mauvaise programmation des scripts.

PHP ne fait pas exception à la règle : si le logiciel lui-même n'a eu que peu de vulnérabilités dans son histoire récente, les applications PHP jouissent d'une mauvaise réputation pour ce qui est de la sécurité. Le défaut vient probablement de la simplicité du langage, qui fait oublier au développeur les risques possibles. Ce chapitre a pour but d'avoir à l'esprit le concept de sécurité, et de savoir comment le mettre en œuvre.

### Qu'est-ce que la sécurité ?

Avant d'entrer dans le vif du sujet, nous tenions à insister un peu plus longuement sur l'importance de ce chapitre. Cette présentation n'a pas pour but de vous alarmer, mais peut-être de vous aider à considérer des aspects que vous auriez sous-estimés.

Tout le monde s'accorde à dire que la sécurité est importante, mais ce qu'il y a derrière est parfois un peu flou. Il nous faut en premier définir à qui s'adresse cette sécurité : celui qui met à disposition l'applicatif, qu'on va appeler le gestionnaire, et celui qui s'en sert, l'utilisateur.



## Préoccupations du gestionnaire

Parmi les problèmes du gestionnaire, on trouve quatre éléments connus :

- la disponibilité de l'appli et son bon fonctionnement (pas de ralentissement, de blocage, d'interruptions de service) ;
- la non-divulgence des données confidentielles (base de clients, statistiques des ventes) ;
- l'intégrité des données (pas d'effacement des bases de données, d'insertion de données externes) ;
- l'intégrité du site et son image (pas de détournement ou d'utilisation abusive).

La sécurité des données est une préoccupation naturelle du développeur. Il s'agit majoritairement de mettre des contrôles d'accès pour que personne n'accède à des données auxquelles il n'a pas droit.

L'intégrité du site et de son image sont en revanche des concepts souvent plus complexes à imaginer. La protection de votre application passera probablement par une surveillance régulière de son contenu et un poste de modérateur si du contenu en provenance des utilisateurs est publié. Nous ne détaillerons pas ce dernier aspect, car il est totalement non technique ; rappelez-vous juste qu'il est important de surveiller les évolutions de contenu.

## Préoccupations de l'utilisateur

La partie la moins prise en compte dans les démarches sécurité est celle des préoccupations de l'utilisateur. Ses besoins sont pourtant connus et assez proches de ceux du gestionnaire :

- la disposition sans interruption de ses données ;
- la non-divulgence de ses données personnelles ;
- l'intégrité de ses données personnelles ;
- le non-détournement de son identité et de son image.

Malgré les ressemblances, il est important de séparer les profils gestionnaire et client. En effet, si le gestionnaire surveille attentivement ce que peut essayer de faire l'utilisateur, ce dernier se méfie aussi du gestionnaire.

Ainsi, la disposition du service n'implique pas uniquement, contrairement au gestionnaire, que l'essentiel de l'application soit disponible mais que *la* donnée nécessaire le soit. L'interruption momentanée d'une partie restreinte de l'appli peut être perçue comme une indisponibilité du site en entier si la donnée manquante est importante.

Pour exemple, dans un service de consultation de messagerie électronique par le Web, un gestionnaire accepterait probablement que la réception des courriers soit différée pendant

deux heures, le temps d'une maintenance, si la consultation des anciens messages reste disponible ; une interruption du site entier de durée plus faible aurait été, elle, inimaginable. L'interruption sera même probablement invisible pour la plupart des utilisateurs. Du côté utilisateur, inversement, si le problème survient justement quand il a absolument besoin d'un message d'un client, sa préoccupation n'aura pas été prise en compte. Cet utilisateur aurait quant à lui préféré la panne totale, mais plus courte.

Ce même schéma se répète sur les trois autres items : la non-divulgence des données utilisateur implique par exemple que les données ne soient pas lues ou connues du public, mais ne le soient pas non plus du gestionnaire et de ses contacts. L'importance des données est elle aussi relative, selon qu'on se place du côté gestionnaire ou du côté utilisateur. Pour l'utilisateur, il est plus grave d'avoir perdu des favoris ou des préférences que de vous voir perdre l'historique des ventes de l'année précédente. Le non-détournement de son image signifie l'impossibilité d'utiliser son identité ou de détourner ses contributions.

### **Importance des données utilisateur**

Pour avoir un exemple concret, prenons le cas d'un gros hébergeur gratuit que nous ne citerons pas. Il y a quelques années, un problème de sécurité avait entraîné la divulgation de plusieurs milliers d'identifiants et mots de passe d'utilisateurs. Heureusement, l'incident a été remarqué très rapidement, les comptes ont été bloqués et une procédure a été mise en place pour débloquer les comptes avec une identification non basée sur le mot de passe. Si on met de côté l'effet sur la réputation, le problème peut sembler mineur pour l'utilisateur : il n'y a pas eu d'intrusion. Pourtant il n'en est rien. Les utilisateurs utilisent en très grande majorité un mot de passe unique sur toutes leurs ressources informatiques. Ce mot de passe peut donner accès à leur compte de courrier électronique, à leur fournisseur d'accès (et chez certains, facturer directement leur compte), à l'extranet de leur société, etc. On peut même imaginer pire, si des noms et adresses avaient été récupérés en même temps, car certains utilisent leur code de carte bancaire comme mot de passe.

### ***Pourquoi parler de l'utilisateur ?***

Si nous nous attardons sur les préoccupations de l'utilisateur, c'est finalement qu'assez souvent elles entrent en conflit avec celles du gestionnaire. Ce sera à vous de trancher entre les deux.

Par exemple, le fait que le mot de passe de l'utilisateur soit crypté dans vos bases de données, que ses données soient cryptées avec son mot de passe (que vous ne connaissez pas, vu qu'il est crypté) est une bonne chose pour lui. Il sait que vous pourrez accéder aux données, car il donne son mot de passe en clair à chaque connexion, mais en son absence, comme vous ne le stockez pas, ses données seront en sécurité. Le gestionnaire, lui, aimera pouvoir accéder aux informations en lecture, par exemple pour faire des statistiques, réorganiser le stockage en base de données, etc.

La démarche naturelle du gestionnaire est de trancher en sa propre faveur et c'est contre cela qu'il faut vous mettre en garde. En effet, l'application est à la destination de l'utilisateur. Même si son but est de vous rapporter quelque chose, elle ne remplira généralement pas son rôle si l'utilisateur n'en est pas satisfait.

### Les risques

Si l'utilisateur voit son adresse électronique partir aux quatre vents sur des robots envoyant des listes de spam, ou voit son numéro de compte bancaire diffusé, le problème sera autant le vôtre que celui de l'utilisateur. L'effet peut être double.

Le premier effet peut être le non-respect de vos engagements. Ce peut être l'engagement pris face aux utilisateurs, mais aussi l'engagement pris par votre société envers l'acheteur de votre application. De même, par exemple, assurer la sécurité des données nominatives et personnelles que vous récoltez est une obligation légale en France, quel que soit l'engagement pris face à vos utilisateurs.

Le deuxième effet peut être presque plus grave. Un problème de sécurité important peut très bien survenir et passer inaperçu, ou être oublié des utilisateurs peu après l'incident. Inversement, un problème même léger peut très vite faire boule de neige. La conséquence pourra être la fuite de vos utilisateurs ou l'association d'une mauvaise image à votre société.

### Déontologie

Il est important de souligner le fait que les données de vos clients sont sous votre responsabilité, que ce soit moralement ou légalement. C'est à vous de garantir leur non-divulgateion ou leur non-utilisation par de tierces parties. On a tendance à surveiller en premier ses propres données, mais vous ne devriez jamais négliger celles des autres, qui sont sous votre charge et pour lesquelles on vous a fait confiance. Si vos clients vous demandent des comportements à risque (comme être capable de redonner l'ancien mot de passe s'il est oublié), en tant que professionnel, c'est à vous d'expliquer pourquoi il ne faut pas le faire et d'essayer de refuser.

## Configuration et sécurité

Maintenant que nous avons abordé en détail les points à prendre en considération dans une démarche de sécurité, il est temps de passer aux détails des erreurs possibles et des méthodes pour les éviter.

Avant de passer au code PHP lui-même et à son utilisation, il est important de s'intéresser à la configuration du système sur lequel il tourne. Si la configuration du serveur est hors du cadre de ce livre, il y a plusieurs options sur la configuration de PHP qui peuvent directement influencer sur la sécurité de votre système.

## Interface avec le serveur web

PHP peut s'exécuter de trois façons : en ligne de commande avec une version spécifique, en module intégré dans une application et via l'interface CGI (*Common Gateway Interface*). Chaque type d'exécution a quelques options relatives à la sécurité.

### Mode CGI

Exécuté en mode CGI, PHP communique avec le serveur via des variables d'environnement ; l'exécutable est isolé du reste du serveur.

### Emplacement de l'exécutable PHP

On voit fréquemment les interpréteurs de scripts dédiés au Web installés directement dans le répertoire CGI du serveur (souvent `cgi-bin`). C'est toutefois une pratique déconseillée s'il est possible de faire autrement.

Dans le cas d'un interpréteur placé directement dans le répertoire CGI, on s'expose à deux types d'attaques :

- `/cgi-bin/php?fichier` ;
- `/cgi-bin/php/dir/page`.

La première vient du fait que ce qui est après le point d'interrogation est normalement passé en paramètre à l'interpréteur. Ainsi, cette URL demande à PHP d'exécuter le fichier nommé `fichier`. En faisant passer le nom de fichier `/etc/passwd`, on demanderait à PHP d'exécuter le fichier de mots de passe des systèmes Unix et associés (qui, ne contenant pas de bloc de code PHP, serait affiché tel quel comme un fichier HTML).

PHP, utilisé en mode CGI, refuse d'interpréter les arguments en ligne de commande afin d'éviter ce problème.

La deuxième attaque se base sur le mode normal d'interprétation des URL par un serveur CGI : si l'URL `/dir/page` est un fichier PHP, le serveur redirige normalement en interne la requête vers `/cgi-bin/dir/page` afin de le faire traiter. Avant la redirection, le serveur web vérifie les contrôles d'accès (authentification ou restriction d'IP par exemple). Utiliser l'adresse directe saute ces contrôles d'accès et permet à l'attaquant d'afficher le contenu de pages restreintes ou interdites.

Afin de résoudre ce problème, PHP utilisé avec Apache permet de vérifier qu'il s'agit bien d'une redirection faite par le serveur et non d'une demande directe du visiteur : Apache envoie un paramètre CGI non standard qui indique la redirection. PHP, à son tour, vérifie la présence de ce paramètre avant d'exécuter le contenu. Pour activer cette fonctionnalité, il faut compiler l'interpréteur PHP avec l'option `--enable-force-cgi-redirect`. Si possible, il est recommandé de toujours activer cette option avec PHP en mode CGI sur serveur Apache.

L'alternative à l'emplacement de PHP dans les répertoires CGI est de se passer des redirections Apache. Vous pouvez placer l'interpréteur dans un répertoire plus habituel (par

exemple `/usr/local/bin`) et exécuter les scripts PHP comme les scripts Perl ou *shell* : en ajoutant `#!/usr/local/bin/php` en première ligne de chaque script appelé par l'utilisateur.

### Droits d'accès

En mode CGI, l'interpréteur est exécuté par le serveur web dans un processus indépendant. Ce processus utilise par défaut les droits utilisateur du processus du serveur web. Sur un environnement multi-utilisateur, ce comportement peut permettre à différents utilisateurs de lire et modifier les fichiers des autres. Sur un environnement mono-utilisateur, les droits peuvent autoriser le script à agir sur tous les fichiers auxquels a droit le serveur web, ce qui est probablement plus que nécessaire.

Afin de restreindre les droits d'accès, il est possible d'utiliser des programmes qui changent les droits d'accès du processus juste avant exécution, pour lui donner des droits plus restreints ou plus spécifiques. Sur le serveur web Apache, ce système s'appelle `suexec` ; nous vous laissons vous reporter à la documentation Apache pour son installation. La mise en place d'un tel système est généralement recommandée : les possibilités de dommages en cas de faille dans un script PHP en seront d'autant réduites.

### Configuration locale

En mode CGI, l'interpréteur lit le fichier de configuration `php.ini` global. Mais en supplément, il vérifie la présence d'un fichier de même nom dans le répertoire courant. Si tel est le cas, il le charge et modifie les valeurs de configuration pour donner priorité au fichier local.

Une telle fonctionnalité est très utile pour maintenir des scripts dans des configurations différentes, mais implique que n'importe qui pouvant poser un tel fichier sera capable de modifier la configuration. Il sera ainsi possible de modifier des options de configuration liées à la sécurité ou d'ajouter des scripts à la directive `auto_prepend_file` pour faire exécuter du code malveillant.

Veillez toujours vérifier les droits d'écriture sur les répertoires utilisés et utiliser au maximum les restrictions de droits d'accès pour qu'un changement ait le moins d'impact possible.

La version de l'interpréteur dédiée à la ligne de commande a la même fonctionnalité.

### Module Apache

Quand PHP tourne en module sur un serveur Apache 1.3 (le cas d'installation le plus fréquent), les scripts sont exécutés avec les droits d'accès du serveur web (généralement `nobody`). Ce fonctionnement peut se révéler dangereux si plusieurs utilisateurs sollicitent votre serveur (auquel cas ils se partagent les mêmes droits d'accès). Si c'est votre cas, il peut être important de spécifier une restriction de type `safe_mode` ou `open_basedir` afin de limiter les possibilités de PHP (voir plus bas et dans le chapitre *Configuration* pour l'explication de ces directives).

## Safe\_mode et restrictions

PHP contient différentes directives permettant d'activer des restrictions d'accès. Ces options rendent impossible l'accès à certaines ressources du système et à certaines fonctionnalités du langage. Elles sont généralement utilisées sur les configurations où plusieurs utilisateurs partagent les droits d'accès du serveur web (afin de limiter quelqu'un à ses propres fichiers par exemple) ou pour limiter des problèmes potentiels en prévision de failles de sécurité (l'attaquant aura alors moins de possibilités pour nuire). Nous vous recommandons de toujours activer ces options par défaut, même si vous contrôlez entièrement le serveur ; il sera toujours temps de diminuer les restrictions au cas par cas quand vous aurez besoin d'une fonctionnalité qui a été limitée.

### Safe\_mode

La directive `safe_mode` prend comme valeur un booléen. Activée, elle permet d'interdire certaines actions. Ainsi, il est possible d'interdire l'exécution de programmes externes, ou de désactiver certaines fonctions et classes. L'avantage principal est de vérifier la concordance entre le propriétaire du script et le propriétaire des fichiers auxquels PHP tente d'accéder.

L'interpréteur limite donc l'utilisateur à la manipulation de ses propres fichiers et pas tous ceux auxquels il a accès normalement. Par exemple, l'utilisateur Pierre se verra refuser l'accès aux fichiers d'Éric, même si le serveur web devrait être normalement capable de les lire.

### Open\_basedir

La directive `open_basedir` prend en paramètre une série de répertoires. Quand elle n'est pas vide, PHP n'autorise la lecture et l'écriture des fichiers que s'ils sont contenus dans un de ces répertoires ou des sous-répertoires.

Il est ainsi possible de définir une sorte d'architecture virtuelle dont PHP ne pourra pas sortir pour ce qui est de l'accès aux fichiers.

### Sources et fichiers externes

PHP a une fonctionnalité très pratique, qui est de pouvoir accéder à des ressources autres que des fichiers locaux de manière transparente. Il est ainsi possible de lire une page web d'un autre serveur comme si c'était un fichier local. L'étendue des possibilités est décrite dans le chapitre sur la gestion des flux.

Cette fonctionnalité ne présente aucun risque en elle-même, mais peut, si votre applicatif comporte une vulnérabilité, ouvrir des possibilités supplémentaires à l'attaquant. Par exemple, on peut imaginer un code fictif contenant la ligne `include($_GET['fichier'])` où le programmeur a oublié de contrôler le contenu de la donnée fournie par l'utilisateur. Si la directive de configuration `allow_url_fopen` est activée (c'est le cas par défaut), alors l'attaquant pourrait fournir l'adresse sur Internet d'un fichier texte contenant du code PHP. Ce code serait alors exécuté par votre application, avec les droits associés.

Cet exemple n'est pas si fictif, car c'est une vulnérabilité qui est assez fréquente. Il s'agit d'une des failles de sécurité les plus communes dans les applications Open Source en PHP. Il est vrai que le problème ne vient pas de la fonctionnalité, mais du programmeur qui n'a pas vérifié ce qu'a envoyé le visiteur avant de l'utiliser. Pourtant, si l'utilisation transparente des fichiers distants avait été désactivée, l'attaquant aurait eu beaucoup plus de mal à exploiter la faille.

Si vous n'utilisez pas cette fonctionnalité ou si vous pouvez vous en passer facilement, vous avez tout intérêt à la désactiver. Votre application ne sera pas plus sûre, mais l'exploitation des vulnérabilités qui pourraient exister sera potentiellement plus complexe.

### Modules et processus externes

Ces deux jeux de directives ont toutefois une portée limitée. En effet, chaque module ou fonction doit explicitement vérifier s'il a le droit d'accéder à une ressource avant de l'utiliser. Les fonctions standards de PHP ainsi que celles de l'essentiel des modules distribués avec PHP se comportent de cette manière.

L'orientation modulaire de PHP vous permet d'en ajouter simplement. Aussi, il est tout à fait possible que vous en récupériez un sur Internet, même d'une source fiable, qui pourrait ne pas vérifier ses droits. Il convient donc de faire attention si vous employez des modules peu courants.

Les fonctions complexes des modules peuvent comporter le même problème : si un nom de fichier peut être fourni de manière détournée, il ne sera pas soumis au contrôle d'accès. De même, si vous laissez la possibilité à PHP d'exécuter un programme externe, rien ne pourra contrôler les accès dudit programme.

## Échappement automatique

Avant PHP 4.1, il était d'usage d'activer la directive `magic_quotes_gpc`, qui préparait automatiquement les valeurs reçues dans la requête HTTP à une utilisation dans une requête SQL (les guillemets sont alors préfixés d'une barre oblique inverse par exemple). Cette possibilité est considérée comme étant une sécurité pour le développeur, car il n'a pas besoin de penser à faire cette conversion lui-même pour être protégé contre les attaques dites d'injection SQL (voir plus bas pour la définition de ce terme).

Actuellement, ce comportement est remis en cause pour des questions de performance (la conversion effectuée ne se révélera pas forcément nécessaire), mais aussi de sécurité et de facilité de programmation. En effet, l'activation de la directive `magic_quotes` implique de toujours savoir d'où vient la donnée qu'on utilise : si elle vient directement de la requête HTTP, elle est déjà convertie, et si elle vient d'une source interne (fichier, calcul, base de données), elle ne l'est pas. Ce défaut rend les abstractions impossibles (on doit à tout moment savoir comment a été reçue la donnée qu'on va traiter) et entraîne parfois l'oubli de la conversion pour les données qui ne viennent pas de la requête HTTP (le

programmeur étant habitué à une conversion automatique). Nous vous recommandons de ne pas activer cette fonctionnalité, mais de bien toujours penser à utiliser la fonction `addslashes()` (ou fonction équivalente) pour vos requêtes SQL.

Le choix des hébergeurs diffère. Beaucoup continuent à activer cette directive pour compatibilité avec les anciennes habitudes et anciens scripts. Avec PHP 5, il est possible que plus d'hébergeurs désactivent la fonctionnalité et cassent la prise en charge des scripts PHP 4. Pour être sûr de votre script, il vous faudra lire la configuration et agir en conséquence. Pour vous aider à le faire automatiquement, vous pouvez utiliser la fonction `get_magic_quotes_gpc()` qui renvoie la valeur de la directive (vrai si activée, faux sinon).

## Variables globales

Une autre fonctionnalité d'aide au développeur autrefois mise en avant est la directive `register_globals`. Activée, elle enregistre une variable globale pour chaque donnée reçue dans la requête HTTP. Par exemple, si le paramètre `?id=5` est présent dans l'adresse de la requête, la variable `$id` avec la valeur 5 sera créée ; si `register_globals` est désactivée, `$_GET['id']` et `$_REQUEST['id']` seront utilisés.

Si cette méthode peut sembler pratique, elle a un gros défaut : elle mélange automatiquement les variables du développeur et celles du visiteur distant. Il est alors possible, pour un programmeur oubliant d'initialiser une variable, d'utiliser une donnée fournie par le visiteur sans le savoir.

Énormément de failles de sécurité venaient de cette fonctionnalité quand elle était activée. Ce problème a poussé l'équipe de PHP à la désactiver par défaut, au risque de casser la compatibilité avec d'anciens scripts. Si vous n'avez pas absolument besoin de l'activer, nous vous recommandons fortement de ne pas le faire.

## Sessions et identifiants

Les sessions contiennent généralement des données importantes, par exemple le fait que le visiteur soit identifié ou non et ses droits d'accès. Connaître l'identifiant de session de quelqu'un permet plus ou moins facilement de l'utiliser et donc d'utiliser l'application web comme si on était la victime, avec les mêmes informations et droits d'accès. Il convient donc de protéger au maximum ces identifiants afin que personne d'autre que le visiteur concerné ne puisse les lire.

### Répertoire de stockage

Les sessions sont par défaut stockées sous forme de fichiers dans un répertoire commun, avec comme nom celui de la session suivi de l'identifiant. Cela peut être un problème si le répertoire utilisé est lisible par d'autres utilisateurs sur votre serveur. Ils seront alors capables de récupérer tous les identifiants de sessions en cours.



Malheureusement, PHP utilise par défaut le seul répertoire qu'il sait accessible en écriture pour ce genre d'information : le répertoire temporaire du système (/tmp sous un Unix par exemple). Ce répertoire est normalement lisible par tous les utilisateurs de la machine, y compris les utilisateurs sans pouvoir. Il est important de changer cette configuration afin que chaque personne bénéficie de son propre répertoire de stockage pour les sessions.

Pour définir un nouveau répertoire, il est possible d'utiliser la fonction `session_save_path()` pendant l'exécution, mais aussi de spécifier un nouveau chemin comme paramètre de la directive `session.save_path` dans la configuration.

Pour les installations de PHP faites en tant que module Apache, il est fréquent d'utiliser la possibilité de modifier la configuration PHP dans le fichier de configuration d'Apache. Il est ainsi possible de spécifier un répertoire de stockage par hôte virtuel (et donc par utilisateur) avec des lignes du type :

```
php_value session.save_path repertoire_destination
```

### Réécriture des liens

Une dernière fonctionnalité désactivable ayant un rapport avec la sécurité est la réécriture des liens par le moteur de session.

Le principe de session nécessite de faire envoyer un identifiant à chaque requête par l'utilisateur. Cet identifiant est normalement envoyé et reçu via les *cookies*. Cependant, certains utilisateurs désactivent les *cookies*. Pour résoudre ce problème, PHP implémente une méthode alternative de transmission de cet identifiant : modifier automatiquement tous les liens de la page afin d'y insérer l'identifiant (vous verrez alors des ajouts du type `?PHPSESSID=xxxxx` à vos liens).

L'activation de cette fonctionnalité peut, dans certains cas, impliquer un risque de sécurité pour l'utilisateur (ou pour vous si vous utilisez les sessions dans la partie d'administration de votre application). En effet, les navigateurs transmettent l'adresse de la dernière page visitée à chaque requête. Si vous vous rendez à partir de votre application sur un site externe contrôlé par un administrateur mal intentionné, ce dernier pourrait relire l'identifiant utilisé sur vos pages et plus ou moins facilement utiliser votre session (ce qui implique généralement obtenir les mêmes droits que vous sur votre application).

La désactivation de cette fonctionnalité implique que vos sessions ne marcheront pas pour les visiteurs n'acceptant pas les *cookies* (ce qui est de plus en plus rare ; tous les navigateurs, depuis de nombreuses années, savent les utiliser). Si vous pouvez vous le permettre, nous vous conseillons alors de ne pas activer la directive de configuration `session.use_trans_sid`.

### Vol et fixation de session

La fixation de session repose sur la possibilité pour l'attaquant de définir lui-même l'identifiant de session de la victime (au lieu qu'il soit défini par votre site). Une procédure très

simple est de passer un identifiant de session dans l'URL, comme si vous aviez activé `session.use_trans_sid` (que ce soit le cas ou pas). Vous pouvez restreindre ces possibilités en activant la directive `session.use_only_cookies`. Dans ce cas, les identifiants de session sont lus uniquement à partir des *cookies*, jamais à partir des éléments GET ou POST des requêtes HTTP.

Vous n'empêcherez pas toute fixation de session, mais vous les rendrez plus complexes à mettre en œuvre. Pour plus de renseignements sur les fixations de session, vous pouvez regarder plus loin dans la section « Protection contre la fixation de session ».

## **Mises à jour du logiciel**

La mise à jour des applications elles-mêmes, comme PHP et Apache, n'est pas non plus à négliger, même si ces applications n'ont historiquement que peu de problèmes. Le conseil de suivre les mises à jour peut sembler superflu, mais on voit énormément de serveurs Internet avec des logiciels présentant des failles datant de plusieurs mois ou plus. Dans le cadre du Web, les exploits découverts sont utilisés dans les quelques jours qui suivent. Il est donc important de se tenir au courant en permanence afin de mettre à jour dès qu'une faille de sécurité est découverte. Il existe, entre autres, une liste de diffusion par courrier électronique dédiée aux annonces PHP : vous y trouverez toutes les nouvelles versions et les correctifs. Vous pouvez vous y inscrire en vous rendant sur le site officiel de PHP à l'adresse [http://www.php.net/mailling\\_lists.php](http://www.php.net/mailling_lists.php).

Le temps passé à la mise à jour et aux tests en vue de mettre en place ces mises à jour est souvent faible au regard du risque d'exploitation des failles.

## **Stockage des données et fichiers**

En dehors de la configuration des logiciels (système d'exploitation, serveur web, etc.) et du moteur PHP, il est possible d'ajouter quelques protections sur l'accès aux fichiers. En effectuant quatre opérations simples, on évite la plupart des problèmes.

### **En dehors de la hiérarchie web**

Les serveurs web sont autorisés à servir au visiteur tous les fichiers contenus dans certains répertoires. Un fichier qui se trouve en dehors de ces répertoires et sous-répertoires ne peut théoriquement pas être appelé directement via une requête sur votre serveur web.

Ce fonctionnement peut être exploité pour protéger un peu plus vos données et fichiers. Il vous suffit de stocker par défaut dans un répertoire qui n'est pas atteignable par le Web. Normalement, seuls les fichiers qui doivent être directement appelés par HTTP ont une raison de se retrouver dans les répertoires accessibles. Tous les autres ne le devraient pas.

Ceci concerne les fichiers de données, les fichiers de cache, les fichiers temporaires, mais aussi les fichiers PHP qui sont utilisés uniquement en les incluant dans des fichiers maîtres (les définitions de classes et de fonctions entre autres).

En plaçant vos fichiers ainsi à l'extérieur, vous vous mettez à l'abri d'une éventuelle mauvaise configuration de vos permissions d'accès web ou de votre serveur HTTP.

### Avec des droits d'accès restreints

Il est inutile de laisser tous vos fichiers et répertoires accessibles en écriture. Il est probable que l'essentiel de vos scripts PHP ne soit utilisé qu'en lecture. Seuls les répertoires contenant les fichiers temporaires et les caches devraient être accessibles en écriture. Il est une bonne idée d'empêcher l'écriture sur tous les autres répertoires et fichiers. Ainsi, si votre applicatif contient une vulnérabilité, l'attaquant ne pourra pas écrire de nouveaux scripts ni modifier les actuels. Il aura alors plus de mal à s'ouvrir des accès supplémentaires ou à installer une porte dérobée.

#### Note

Un utilisateur a toujours le droit de modifier les permissions sur ses propres fichiers. Si vous voulez imposer la lecture seule sur certains répertoires et fichiers, il faut qu'ils appartiennent à un utilisateur différent de celui qui exécute les scripts.

### Interdiction d'accès pour le serveur web

Dans le même ordre d'idées, il peut être utile de créer des permissions spéciales pour votre serveur web. Par exemple, vous pouvez préfixer vos fichiers avec une certaine chaîne et demander à votre serveur web de refuser l'accès direct vers les fichiers ainsi préfixés (par exemple, avec la configuration par défaut d'Apache, les fichiers commençant par `.ht` sont interdits de lecture directe). Avec le serveur Apache, il est aussi possible de créer des fichiers de configuration locale qui interdisent l'accès direct (ces fichiers sont nommés `.htaccess` par défaut).

L'utilisation de ces protections n'est pas forcément contraire à l'idée de placer les fichiers en dehors de la hiérarchie du serveur web : vous obtenez ainsi une assurance, même s'il y a une erreur dans la configuration des répertoires accessibles.

On peut aussi les mettre dans un répertoire dédié qui est interdit par Apache avec une directive `deny from all`.

### Extensions des fichiers

Une habitude de nombreux programmeurs PHP est de mettre `.inc` comme extension pour les scripts PHP qui sont inclus dans d'autres. Nous déconseillons fortement cette procédure.

En donnant l'extension `.php` à tous vos scripts sans exception, vous vous assurez qu'en cas de mauvaise configuration de votre serveur web, vos fichiers seront toujours exécutés, mais jamais visualisés. Pour certains, la différence n'est pas intéressante, mais pour les scripts contenant des mots de passe, des clés de chiffrement ou des informations confidentielles, la non-visualisation est importante.

**Astuce**

Le fait d'avoir une extension en `.php` ne vous empêche pas de nommer vos fichiers de façon à reconnaître leur utilité, par exemple `.inc.php` ou `.class.php`.

## Sécurité de l'application

L'étape de configuration est la plus simple à mettre en œuvre côté sécurité. Ce qui reste à faire est ce qui est le plus vulnérable : le code de votre application. En effet, les services Internet sont actuellement mis en difficulté plus souvent à cause d'un code imparfait que de problèmes sur le système, la configuration ou les logiciels utilisés.

La simplicité de PHP fait souvent oublier les détails de l'implémentation ; il est pourtant important de respecter quelques habitudes afin de limiter les risques. Cette partie va vous montrer les erreurs les plus courantes et leur exploitation, pour vous permettre de les garder en tête pendant le développement.

### *Vérification des entrées utilisateur*

L'erreur la plus fréquente rencontrée dans les applications, dédiées au Web ou non, est une absence de vérification des informations envoyées par l'utilisateur. PHP ne fait pas exception à la règle.

Pour donner un exemple concret, on peut détailler le problème d'inclusion.

**Utiliser l'extension Filter**

Une partie importante des problèmes de vérification des entrées utilisateur peut être résolue en n'utilisant plus directement les tableaux `$_GET`, `$_POST`, `$_COOKIE` et `$_REQUEST`. Vous pouvez alors passer par l'extension Filter avec des filtres appropriées. Vous évitez ainsi les problèmes de transtypage, de cohérence, et vous pouvez vérifier les critères simples automatiquement (valeur minimum d'un nombre, l'absence de partie décimale, etc.).

Si l'utilisation de Filter ne pourra pas vous garantir une sécurité totale ou couvrir toutes les vérifications à faire sur les entrées utilisateurs, elle vous offre tout de même des facilités non négligeables. Nous vous recommandons fortement de considérer l'extension Filter comme une étape obligatoire. Rendez-vous au chapitre 8 pour avoir plus d'informations.

## Paramètre dans une liste prévisible

Beaucoup de sites ont un script central, qui en appelle d'autres en fonction de paramètres fournis par l'utilisateur. Essayons de décrire ce fonctionnement.

Si on reçoit un URI de type `/index.php?module=forum`, on charge le fichier `forum.php` afin d'afficher un cadre avec les derniers messages du forum sur la page d'accueil. La manière la plus directe qui vient à l'esprit est la ligne suivante :

```
include( $_REQUEST['module'] . '.php' );
```

Cette formulation paraît simple, mais elle comporte une vulnérabilité évidente. En effet, si, à la place de `forum`, je fournis l'adresse d'un fichier texte disponible sur mon site et qui contient du PHP, l'application exécute mon script. J'ai alors possibilité de fournir n'importe quel code au moteur, et d'accéder aux données avec les mêmes droits que ceux de l'application vulnérable.

On peut se dire qu'en désactivant les possibilités d'accès aux fichiers à distance, l'application retrouve de sa sécurité, mais il n'en est rien. Il existe énormément de méthodes pour arriver à insérer du code sur le serveur et donc d'y faire appel via la faille précédente.

On peut par exemple citer les sessions (si vous stockez une information de l'utilisateur dans les sessions, il peut très bien vous y faire mettre du code PHP), les journaux de *log* (il est possible de transmettre des caractères dans les URL, et donc de les voir apparaître dans les *logs* du serveur web), les fichiers temporaires (si vous transmettez un fichier avec votre requête, PHP le sauvegarde automatiquement en local dans un répertoire temporaire), etc.

De plus, même si vous vérifiez bien toutes les méthodes précédentes, il reste toujours la possibilité à l'attaquant de lire un fichier non exécutable contenant des données privées. Un exemple significatif est la lecture du fichier de mots de passe en demandant `/etc/passwd`.

Pour éviter d'inclure des fichiers non souhaités, il est théoriquement possible de vérifier le texte fourni par l'utilisateur pour le disqualifier s'il contient des motifs interdits : deux points consécutifs significatifs d'une remontée dans la hiérarchie des répertoires, une barre oblique en premier caractère pour une adresse absolue, le caractère deux-points pour une adresse préfixée par un protocole, etc. Théoriquement seulement, parce qu'en pratique, il est trop facile d'oublier une vérification et donc de laisser une porte ouverte. En pratique, la seule méthode sûre est de dresser une liste des paramètres autorisés et de vérifier la concordance :

```
$autorise = array( 'forum', 'actu', 'articles' );

if (in_array($_REQUEST['module'], $autorise)) {
    include( $_REQUEST['module'] . '.php' );
} else {
    // Renvoyer une erreur
}
```

## Paramètre avec des critères connus

Le cas précédent est un cas d'école souvent pris en exemple, mais la plupart des cas réels sont souvent moins simples à traiter. En effet, si pour quelques informations comme les codes postaux on peut établir une liste des valeurs autorisées, il est fréquent de ne pas pouvoir le faire, voire de ne pas savoir ce qui peut être reçu.

Pourtant, il est toujours possible de définir quelques critères sur la donnée à traiter. Par exemple, on sait qu'un âge est un nombre positif de deux ou trois chiffres inférieur à 200. Pour un nom de ville, c'est une suite de moins de 75 caractères alphabétiques qui peut contenir des accents et des tirets mais aucun autre caractère spécial.

En filtrant l'âge, vous évitez que quelqu'un puisse saisir un nombre négatif et déclencher des actions non prévues, par exemple entraîner un prix négatif pour un service dépendant de l'âge. En filtrant le nom de la ville, vous évitez que quelqu'un puisse saturer votre système de fichiers avec des données de plusieurs méga-octets ou que quelqu'un envoie des caractères spéciaux qui poseront problème à votre système.

Une donnée a toujours des critères définissables, au minimum une taille maximale, le plus souvent un type et une liste de caractères autorisés. Il est même assez fréquemment possible d'utiliser une expression régulière pour faire une vérification complexe. Dans tous les cas, une vérification doit être faite et cette vérification doit être la plus stricte possible.

## Cas par défaut dans une vérification

Si vous autorisez cinq types de valeurs, une manière naturelle est d'en énumérer quatre pour essayer de voir si la valeur correspond et sinon de considérer que la valeur est du cinquième type. Cette méthode est particulièrement utilisée avec la simple suite `if {} else {}`. Pour une valeur qui est soit 0, soit 1, il est courant de faire :

```
if ($valeur == 1) {  
    echo "la valeur est 1" ;  
} else {  
    echo "la valeur est 0" ;  
}
```

Cette manière de procéder est à éviter. Même si vous pensez être sûr que la valeur est bien d'un des cas possibles, vous prenez un risque si la fonction de vérification en amont se révèle défectueuse. Il est fortement conseillé de prévoir les cas théoriquement impossibles, et de lancer une erreur si étrangement ils surviennent :

```
if ($valeur == 1) {  
    echo "la valeur est 1" ;  
} elseif ($valeur == 0) {  
    echo "la valeur est 0" ;  
} else {  
    // Lancement d'une erreur  
}
```

Ce type de sélection est à faire la plupart du temps, quand c'est possible : le cas par défaut ne doit alors pas être une des alternatives, mais le cas impossible. Si jamais un problème survient, vous pourrez vous en rendre compte et intercepter l'erreur.

### Concordance et transtypage

PHP est un langage faiblement typé. Il n'est normalement jamais nécessaire de s'occuper du type d'une valeur, car les conversions éventuelles sont faites automatiquement. Les données provenant de l'utilisateur ou d'une source externe sont une exception.

En effet, si les conversions automatiques de PHP sont un avantage, elles sont aussi un point faible. Il existe des erreurs fréquentes.

#### Chaînes composées de zéros

La première erreur est due à la chaîne de caractères "00". La chaîne ne contient que des chiffres et passe donc les vérifications excluant les chaînes de caractères non numériques. Un zéro classique est converti par PHP en nombre mais un double zéro reste une chaîne de caractères ; la valeur s'évalue donc à TRUE, contrairement au nombre zéro. En revanche, quand on utilise la valeur dans une opération numérique ou à la place d'un nombre, la chaîne est évaluée comme le chiffre zéro. On se trouve donc devant une valeur vraie, qui ne contient que des chiffres et qui pourtant vaut zéro. L'erreur principale dans ces interprétations vient du fait qu'une valeur s'évaluant à TRUE est non nulle.

```
$valeur = "00" ;  
// On vérifie qu'il s'agit bien uniquement de valeurs numériques :  
if ( !is_numeric($valeur) ) die("pas un nombre") ;  
if ( $valeur ) echo 'action si non nul <br>' ;  
if ( $valeur == 0 ) echo 'action si nul <br>' ;  
/* La valeur "00" exécutera les actions prévues,  
   pour les valeur nulles ET pour les valeurs non nulles.  
   Ce n'est probablement pas ce qui était prévu et risquera  
   de provoquer des incohérences */
```

#### Chaînes commençant par des chiffres

La deuxième erreur vient aussi de la conversion entre les nombres et les textes. En effet, avec PHP, additionner le chiffre 3 à la chaîne de caractères "5 est la version de PHP" donne la valeur 8. Une chaîne de caractères non numériques ne s'évalue pas toujours à zéro. Imaginons alors le cas d'un magasin en ligne. Il permet de saisir la quantité de produit qu'on veut avant d'acheter. Je demande, en tant que client, deux fois le même produit, avec la première fois le nombre 3 et la deuxième fois le texte "3Z". Le programme vérifie que mes quantités sont supérieures à zéro, ce qui est vrai pour les deux, avant de les additionner pour déclencher l'envoi de 6 produits. Dans la facture en revanche, étant donné que j'ai fait deux entrées, il faut au magasin facturer deux fois trois produits et non six ; le programme insère deux lignes dans la table de commande de la base de données, pour produire la facture plus tard. Le problème est que lors de l'insertion, mon texte "3Z" est refusé par la base de données car non numérique. Seule une ligne

sera effectivement inscrite dans la table, et quand la facture sera établie, je n'aurai que trois produits à payer alors qu'on m'en a envoyé six.

### Tableaux reçus par HTTP

La dernière erreur liée aux types de données est moins connue. Il s'agit principalement d'une assertion faite par le programmeur qui se révèle fausse. En effet, nombreux sont les programmeurs qui vérifient si une donnée reçue est un texte et, dans le cas contraire, la traitent comme une valeur numérique, ou inversement. Ces vérifications oublient que le visiteur a moyen de soumettre des tableaux à PHP via les requêtes HTTP. S'il fournit une valeur sous un nom se terminant par des crochets (par exemple `nom[]`), alors PHP crée un tableau pour contenir la valeur.

### Éviter les erreurs

Dans les trois cas précédents il est possible d'éviter les potentiels problèmes à l'aide de quelques lignes :

- toujours vérifier le type d'une variable et ne pas simplement faire des comparaisons de valeurs ;
- avant d'utiliser une valeur qui doit avoir un certain type, il faut toujours la convertir vers le type en question.

#### Note sur les comparaisons

Il est possible pour les tests d'égalité de vérifier l'égalité du type en plus de la valeur en ajoutant un signe égal. Ainsi, `3=="3"` est vrai mais `3==="3"` est faux. Il est fortement conseillé de faire ces tests complets quand c'est possible ; ils sont de plus très légèrement plus rapides que les tests normaux car il n'y a pas d'essai de conversion avant le test.

## Éviter les principales attaques

L'utilisation directe des données externes juste après d'éventuelles vérifications entraîne certaines des vulnérabilités parmi les plus connues et les plus simples à mettre en œuvre : l'injection SQL et le *Cross Site Scripting*.

Dans les deux cas, l'erreur est la même. Selon l'usage auquel est destiné le texte reçu, certains caractères peuvent avoir des significations spéciales. Si le programmeur ne veut pas que celui qui contrôle le texte puisse contrôler les actions que déclenchent ces caractères, il lui faut les transformer.

### Injection SQL

Pour des applications contrôlées par base de données, il est fréquent d'intégrer des éléments venant de l'utilisateur ou d'une source externe dans des requêtes SQL. Le procédé d'injection SQL consiste à modifier le comportement de la requête en insérant des caractères de contrôle.



Pour illustrer cette faille de sécurité, on peut prendre l'exemple d'une procédure d'identification. On demande à l'utilisateur un identifiant et un mot de passe avant de vérifier dans notre base de données que les deux correspondent à ce qu'on a. La requête SQL pourrait être du type de la suivante :

```
$sql = "SELECT nom FROM utilisateurs
WHERE identifiant = '$_REQUEST['identifiant'].'"
AND passe = '$_REQUEST['passe'].'" ;
```

Si l'utilisateur n'y pense pas, tout marche correctement. En revanche, s'il fournit "pierre" -- " comme identifiant utilisateur, la requête SQL devient la suivante :

```
SELECT nom FROM utilisateurs WHERE idenfiant = 'pierre' -- ' AND passe = ''
```

La suite de deux tirets marquant un début de commentaire, l'utilisateur sera authentifié en tant que Pierre sans avoir besoin de mot de passe.

Pendant plusieurs années, ces problèmes étaient rares dans les applications PHP, car, par défaut, la directive de configuration `magic_quotes_gpc` était activée. Les valeurs reçues par l'utilisateur étaient alors directement converties pour ajouter des barres obliques inverses devant les apostrophes. Ainsi, ces attaques étaient sans effet car la requête modifiée aurait donné :

```
SELECT nom FROM utilisateurs WHERE idenfiant = 'pierre\' -- \' AND passe = ''
```

Cela aurait échoué. Actuellement il est recommandé de désactiver cette directive (voir précédemment dans ce chapitre et dans celui sur la configuration pour plus de détails). Il est donc de la responsabilité du programmeur de faire appel à la fonction `addslashes()` ou similaire avant toute utilisation d'une chaîne à risque dans une requête SQL. Pour PDO il s'agit normalement de la méthode `quote()`.

#### Note

L'utilisation de requêtes paramétrées (disponibles entre autres avec PDO), vous permet de vous protéger contre les attaques par injection SQL de manière efficace avec un minimum de contraintes. Nous vous recommandons très fortement cette méthode plutôt que des échappements manuels à chaque opération. Vous retrouverez plus d'informations à ce sujet dans le chapitre dédié à l'utilisation de PDO.

## Cross Site Scripting

Le *Cross Site Scripting* est l'application de ce même principe à d'autres langages que le SQL, le plus souvent JavaScript, HTML ou CSS. Ainsi, dès qu'une donnée est renvoyée vers le navigateur, il est primordial de vérifier son contenu et de convertir les caractères spéciaux. En cas d'oubli ou de conversion imparfaite, l'attaquant pourra réussir à afficher du code HTML, JavaScript ou CSS dans la page.

Beaucoup de programmeurs pensent, avant d'être confrontés au problème, qu'il ne s'agit pas là d'une faille de sécurité puisque leurs propres données ne sont pas en jeu. C'est pour cela que nous avons insisté au début de ce chapitre sur les différents points de vue de la sécurité.

Imaginez qu'un attaquant insère du JavaScript (par exemple sur un forum). Il pourra alors récupérer les *cookies* présents sur les navigateurs de vos clients, avec les identifiants de session. Avec ces identifiants, il peut potentiellement usurper des identités, accéder à des zones d'administration ou protégées, etc. L'exploitation de cette faille peut se révéler, elle, un vrai problème de sécurité. Imaginez aussi quel impact sur la réputation de votre application aurait une page dégradée par du HTML injecté au milieu de votre page. On peut aussi penser à un code qui renverrait certains visiteurs chez le concurrent.

Ces deux concepts additionnés, le *Cross Site Scripting* amène presque toujours la question de la confiance. Auriez-vous confiance dans un site qui affiche une page bizarre ou exploitée par un visiteur ? Dans une application qui permet à un autre visiteur d'utiliser vos données ? Dans ce cas, feriez-vous affaire avec la société en question ? Une simple erreur n'entraînant aucune divulgation peut entraîner tout de même un gros impact sur la réputation et la confiance envers votre site.

Il est important de noter que, même si le code non converti n'est affiché qu'à destination de celui qui l'envoie, le problème existe. En effet, il serait facile à l'attaquant de mettre un lien qui atterrit sur votre site en déclenchant l'*exploit*. Par exemple, si je crée un code JavaScript qui récupère l'identifiant de session en *cookie* avant de l'envoyer sur un site tiers que je contrôle et si j'arrive à exploiter une faille pour faire afficher ce code sur un site, il me suffit de diffuser un lien contenant l'*exploit*. Les infortunés qui cliqueront enverront leur identifiant de session chez moi. Le visiteur voit et subit alors les effets sans en être réellement à l'origine.

Pour du HTML ou XML, vous pouvez utiliser les fonctions `strip_tags()` ou `htmlspecialchars()` afin d'éviter que le texte ne soit interprété.

### Cross Site Request Forgery

Le *Cross Site Request Forgery* n'est pas vraiment impacté par la présence de PHP. Les solutions ne sont généralement pas techniques ni propres à PHP.

Comme le *Cross Site Scripting*, l'objet de cette attaque est de profiter d'une victime pour lui faire faire quelque chose de non prévu. En constatant l'impossibilité de faire exécuter du code au navigateur (parce que vous avez évité les failles précédentes), notre attaquant n'a plus qu'un seul outil : les liens HTML. Il va donc essayer de convaincre sa victime de cliquer sur un lien HTML. Généralement il va réussir.

Imaginons que je navigue sur mon webmail favori. Sur ce webmail, je peux effacer des e-mails en allant sur la page <http://webmail.example.org/delete.php> avec un tableau d'identifiants en paramètres. Le webmail reçoit la requête et efface les données correspondantes. L'attaquant peut alors tout simplement mettre en place un lien vers cette page en passant les identifiants 1 à 100 en paramètre. S'il arrive à me faire cliquer sur le lien, c'est finalement moi-même qui vais demander à mon webmail d'effacer mes 100 premiers e-mails. Comme je suis identifié sur mon webmail, la commande va être acceptée et l'attaquant va bien rire en me voyant restaurer mes sauvegardes (avez-vous pensé à faire des sauvegardes ?).

Les authentifications classiques comme le cookie, la session ou l'identification HTTP ne sont d'aucun recours : vu que c'est mon navigateur qui fait la requête, c'est mon authentification qui sera utilisée.

La seule protection simple et efficace est d'empêcher l'attaquant de pouvoir créer un lien correct. L'idée pourrait être d'insérer un jeton dans l'URL par exemple. Ce jeton doit être unique, secret et modifié à chaque connexion. Dans ce cas, l'attaquant ne connaîtra pas le jeton à utiliser et ne pourra pas le prédire. Le lien qu'il pourra générer ne sera pas valide et le webmail refusera la requête.

Ce système de jeton unique est à créer en plus du système d'authentification classique. Il est toutefois pénible à mettre en place et peut être réservé aux opérations sensibles ou destructrices (effacement ou changement de mot de passe, par exemple).

### Exécution de commandes

Sur certains projets il est nécessaire d'exécuter des programmes externes sur le serveur. Dans ce cas, l'utilisation de paramètres ou de données venant de l'utilisateur peut se révéler très dangereuse.

Par exemple, si on essaie de lire un répertoire avec la commande `ls`, un code possible serait :

```
$rep = $_REQUEST['repertoire'] ;  
system("ls -l $rep") ;
```

Ces lignes comportent pourtant une vulnérabilité importante. Si on passe `"/tmp ; rm -f *` comme paramètre, alors le script effacera tous les fichiers du répertoire courant, ce qui n'était pas prévu.

Pour éviter ces désagréments, il existe deux fonctions préparant les données client à une utilisation dans une commande système. La fonction `escapeshellcmd()` échappe tous les caractères qui pourraient servir à exécuter des commandes arbitraires. On peut utiliser alors la valeur retournée dans une commande sans aucun risque. La fonction `escapeshellarg()`, elle, prépare la valeur à être utilisée en argument dans un programme. Elle entoure la chaîne par des apostrophes et échappe celles qui sont contenues à l'intérieur. Le résultat est alors interprété comme un argument unique, peu importe la présence d'espaces.

#### Attention

Ces fonctions ne font que délimiter les arguments de façon qu'ils n'aient pas une signification particulière lors du lancement du programme externe. Il est de votre responsabilité de vérifier qu'ils correspondent bien à ce que votre programme sait recevoir et à ce que vous voulez lui fournir. Vérifiez en particulier que le programme n'interprétera pas les arguments fournis pour faire des actions que vous ne voudriez pas utiliser. N'utilisez des arguments utilisateur dans des programmes que si vous maîtrisez totalement le fonctionnement du programme face à ses arguments.

## Autres types de données et importance

Nous avons détaillé les conversions vers le SQL, le HTML et les exécutions système. Ce sont les cas les plus courants, mais ils ne sont pas exhaustifs. Quel que soit le format que vous destinez aux données reçues, vous devrez presque toujours les convertir ou les filtrer avant de les envoyer. Même le format texte est à filtrer la plupart du temps pour retirer les caractères spéciaux ou au moins le caractère ayant pour valeur ASCII zéro. Utiliser une valeur sans la convertir est probablement l'erreur applicative la plus courante après le manque de vérification. C'est en tout cas une des plus faciles à exploiter.

## Emplacement des contrôles

Les vulnérabilités dans les applications ne viennent pas uniquement d'oublis de vérification, de filtrage et de conversion. Assez fréquemment le développeur a pensé à vérifier les accès et contrôler les données mais gère mal le fonctionnement de sa propre application. Il place alors des contrôles à des endroits non pertinents pour laisser vides de vérification les endroits où c'est nécessaire.

### Stockage du résultat chez l'utilisateur

La forme la plus évidente de mauvais emplacement des contrôles d'accès est le stockage de résultat chez l'utilisateur. Lors d'une authentification, après vérification de l'identifiant et du mot de passe, il faut stocker le fait que l'utilisateur est authentifié. Un néophyte aura peut-être le réflexe d'utiliser un *cookie* pour cela. Le problème de sécurité posé par une telle idée est assez évident : l'utilisateur pourra modifier lui-même son *cookie*, et donc faire croire à l'application qu'il a déjà été authentifié alors que ce n'est pas le cas.

Un tel problème paraît simple à éviter, pourtant on en voit des similaires assez souvent, mais un peu plus durs à déceler.

### Chiffrement avant stockage chez le client

Pour éviter ce problème, notre débutant aura souvent comme premier réflexe de penser au chiffrement. La solution qui paraît bonne est de chiffrer le résultat avant de l'envoyer dans un *cookie*. L'attaquant ne pourra alors pas savoir ce qu'il contient.

La solution est mauvaise car elle ne répond en fait pas au vrai problème. Le problème n'était pas que le visiteur pouvait modifier le *cookie*, mais que le *cookie* n'était pas un bon endroit pour stocker ce type d'information. En effet, la vulnérabilité est toujours présente, bien que plus complexe à exploiter. L'attaquant aura maintenant pour but de récolter un *cookie* réel chez quelqu'un (par exemple grâce à un *Cross Site Scripting*). Il ne pourra pas le relire, mais il n'en a pas besoin. Il lui suffit de se créer un *cookie* identique pour obtenir les mêmes droits d'accès. Le chiffrement n'a offert aucune protection parce que si le résultat stocké est une chaîne incompréhensible, il est toujours stocké chez le client et il suffit toujours pour se faire croire authentifié.

### Exemple avec un formulaire en plusieurs étapes

Il est fréquent d'avoir des saisies de formulaire en plusieurs étapes : adresse de facturation, adresses de livraison, informations sur les personnes, etc. Dans ce cas-là, de nombreux développeurs vérifient les données saisies et les réinjectent dans le formulaire suivant. Ils assurent ainsi la sauvegarde de l'information jusqu'à la soumission finale sans avoir besoin d'une session ou d'un procédé similaire.

Malheureusement, assez souvent chaque donnée est vérifiée la première fois qu'elle apparaît et uniquement à ce moment. Les deux raisons invoquées sont qu'il est inutile de vérifier la même donnée plusieurs fois et qu'il doit forcément y avoir une vérification à l'apparition d'une valeur pour pouvoir la refuser avant de passer à l'étape suivante.

Le raisonnement paraît sensé, mais alors rien n'empêche l'utilisateur de modifier les données cachées dans le formulaire au moment de la dernière soumission. La vérification n'aura servi à rien, car il est possible de modifier une donnée vérifiée ou de faire comme si une donnée avait été vérifiée.

### Les règles à respecter

Si vous voulez avoir confiance dans un résultat, ne stockez jamais le résultat chez l'utilisateur ; stockez les paramètres du calcul. Il vous restera à refaire le calcul en interne à chaque fois que vous aurez besoin du résultat. Si les données sources peuvent être modifiées, les contrôles doivent être faits à chaque fois qu'on utilise un résultat, pas seulement la première fois.

Si une vérification ou un calcul prend du temps, il vous faudra mettre en place un système de cache pour retenir ce résultat, mais en aucun cas une information fournie par l'utilisateur ne devra être utilisée directement. Utiliser les sessions à cet effet pourrait être une bonne idée pour la plupart des applications.

### Contrôles JavaScript

Toujours dans la détermination de l'emplacement des contrôles et vérifications, il est important de faire attention à la différenciation entre les exécutions chez le client et celles sur le serveur.

Des contrôles sur les données de formulaires envoyées par un client peuvent être faits en JavaScript. C'est même assez souvent une bonne solution, car il est possible de signaler l'erreur à l'utilisateur sans avoir besoin de recharger la page. On peut par exemple lui faire remarquer que son code postal contient un chiffre de trop.

Pourtant ces contrôles en JavaScript ne doivent être pris que comme une aide pour le visiteur, et nullement une aide pour le programmeur. Ils ne seront par exemple pas utilisés si le JavaScript est désactivé, et un attaquant pourra de toute façon très facilement soumettre des données sans passer par le JavaScript.

Il est donc important de refaire toutes les vérifications et tous les filtrages sur le serveur, et de ne pas donner foi aux résultats trouvés par JavaScript. Le langage JavaScript est

donné en exemple, mais le principe est extensible à tout logiciel qui s'exécute sur le poste client et non sur le poste serveur : *applet* Java, Flash, VBScript, *plug-in*, etc.

### Mauvaises assertions

Une mauvaise assertion est une hypothèse faite par le développeur qui se révèle fausse alors qu'elle aurait dû toujours être vraie. Ces mauvaises assertions sont relativement courantes. En général, elles surviennent quand un visiteur fait quelque chose d'imprévu.

Par exemple, PHP permet facilement d'afficher ou non des liens et des formulaires suivant les autorisations qu'a le visiteur. Il est facile de penser que si quelqu'un arrive sur une page ou soumet le formulaire, c'est qu'il est autorisé à le faire, parce que sinon il n'aurait pas eu le lien ou le formulaire vierge. C'est un exemple simple de mauvaise assertion. Il n'est pas impossible que quelqu'un arrive sur le lien sans y avoir été invité. Il peut s'agir de quelqu'un qui avait les droits et qui ne les a plus, mais qui a sauvegardé les pages d'origine, ou d'un administrateur qui a vu l'adresse des pages et qui a choisi de s'en servir, ou encore simplement de quelqu'un qui a essayé pour voir si cela fonctionnait.

L'exemple plus haut est relativement simple (même s'il n'est malheureusement pas si rare de le rencontrer), mais il doit vous encourager à vérifier toutes vos assertions. Si à un quelconque moment, vous faites une déduction ou une hypothèse, alors vous avez un comportement à risque. Toutes vos assertions doivent découler d'un calcul ou de données concrètes. De plus, les données et calculs en question doivent venir du serveur ou prendre en compte la possibilité qu'a l'utilisateur d'avoir fourni autre chose que prévu.

### Protection par le secret

La protection d'une ressource par le secret est souvent employée. Par secret on entend par exemple de garder secret l'URL d'un outil d'administration sans y mettre de procédure d'authentification.

Toute sécurité basée sur le secret est amenée un jour ou l'autre à s'effondrer. Dans ce type de protection il faut au moins utiliser un mot de passe identificateur.

Il est donc tout à fait déconseillé de ne pas mettre d'authentification sur un outil d'administration en se disant que seuls les administrateurs connaissent l'adresse de la page. Tôt ou tard, un de vos administrateurs notera l'adresse quelque part où elle pourra être lue. L'historique de votre navigateur pourrait être lu à cause d'une faille de sécurité sur votre poste personnel. L'administrateur d'un routeur ou d'une passerelle entre un administrateur et votre serveur pourrait lire cette adresse, etc.

La protection par le secret est toujours faible car les moyens de divulguer involontairement un secret sont très nombreux. Ne vous reposez jamais sur le fait que quelqu'un connaisse ou ignore certaines choses.

## Gérer les erreurs

Il est important d'avoir une politique de gestion des erreurs, de savoir si vous devez les afficher, de penser à faire un fichier de journalisation et à le lire régulièrement, voire d'implémenter régulièrement des vérifications de cohérences dans votre code. Consultez le chapitre 19 dédié à ce sujet pour plus d'informations.

## Sécuriser les sessions

Stocker les données temporaires sur l'utilisateur en session comporte généralement peu de risques par rapport aux autres solutions (si les sessions sont bien configurées, voir plus haut dans la sécurité de la configuration). Pourtant, les sessions ont un point faible important : leur identifiant. Quiconque connaît cet identifiant peu se faire passer pour un autre utilisateur.

### Création manuelle des identifiants

Les identifiants sont calculés par PHP avec un mélange de dates et de code aléatoire. Ils sont pratiquement impossibles à prédire et leur longueur les met à l'abri d'une recherche incrémentale. PHP vous offre, via la fonction `session_id()`, la possibilité de définir vous-même les identifiants de session. Nous vous mettons fortement en garde contre cette possibilité. Il est très improbable que vous ayez besoin de cette fonctionnalité (à vrai dire, l'utilité de définir soi-même les identifiants reste à établir), mais si vous le faites sans une maîtrise complète du sujet, vous ferez à coup sûr une fonction moins aléatoire que la fonction interne de PHP et qui sera plus prédictible. Il sera alors possible ou plus facile pour un attaquant d'usurper un identifiant existant (et donc une session existante).

### Protection contre les vols de sessions

Quiconque connaissant ou devinant un identifiant de session peut par défaut l'utiliser. Il existe toutefois une protection supplémentaire possible et facile à mettre en œuvre.

À la création de la session, stockez des informations relatives au client, notamment son adresse IP publique, et son adresse IP privée s'il passe par un proxy qui l'envoie. À chaque ouverture de session, vérifiez que les informations sont bien identiques à celles qu'il a fournies lors de la création. Si ce n'est pas le cas, détruisez la session et les données contenues avant de refuser l'accès : c'est que quelqu'un essaie d'usurper la session.

Il est important de noter que cette méthode n'est en rien une protection absolue. Il est tout à fait possible de simuler des informations comme l'adresse IP. Malgré tout, la procédure est plus complexe et l'attaquant préférera peut-être s'occuper d'un site où cette protection n'est pas mise en place.

De plus, pour passer outre, il devra se rendre compte de toutes les informations vérifiées, et toutes les fournir. Comme à chaque essai, la session usurpée est perdue, il devra trouver beaucoup d'identifiants avant d'arriver à en usurper un réellement, s'il y arrive.

Enfin, le fait de stocker l'adresse IP limite de fait les utilisations qu'il peut faire de ce vol de session. Si l'attaquant peut masquer son adresse réelle et en fournir une fausse (procédé nommé *IP spoofing*), il ne peut pas recevoir les réponses du serveur web. L'attaquant peut tout au plus envoyer des données, mais pas en recevoir.

**Note**

Il existe toutefois certaines sociétés qui ont plusieurs proxies et où deux requêtes d'un même utilisateur peuvent venir tour à tour de plusieurs adresses IP. Une variante plus sage est d'autoriser les requêtes venant de la même IP ou d'une IP proche (bloc identique).

**Protection contre la fixation de session**

Dans les chapitres sur les sessions et les *cookies*, nous vous avons suggéré de toujours supprimer la session existante et d'en recréer une autre quand un utilisateur s'identifie. Il est temps de vous expliquer pourquoi.

La fixation de session est un vol de session qui marche de manière inversée. En considérant que je suis un visiteur comme un autre du site, au lieu d'essayer de voler mon identifiant de session, l'attaquant va essayer de m'en attribuer un lui-même pour que je l'utilise par la suite. Dans les deux cas, il connaîtra l'identifiant que j'utilise et pourra utiliser ma session.

Attribuer un identifiant de session à un tiers est en fait assez simple. Si la directive de configuration `session.use_only_cookies` du fichier `php.ini` est désactivée, il suffit de faire suivre un lien à la victime. Si le lien fini par `?PHPSESSID=xxx` où `xxx` est un identifiant, la victime aura désormais cet identifiant sur le site tant qu'elle n'en partira pas. Une autre possibilité est d'exploiter une faille de *Cross Site Scripting* pour insérer un *cookie* chez la victime.

Une fois la session insérée, la victime peut s'identifier et continuer sa visite comme si de rien n'était. Si c'est le cas, l'attaquant bénéficie alors d'une session authentifiée valide à sa disposition (il en connaît l'identifiant car c'est lui qui l'a donnée à la victime).

Le palliatif est de calculer un nouvel identifiant de session quand l'utilisateur s'identifie. Ainsi, l'attaquant bénéficiera au mieux d'une session anonyme, sans privilège, ce qui ne lui servira probablement pas. Pour nous aider, on a la fonction `session_regenerate_id()` ; son rôle est de changer l'identifiant tout en gardant le contenu actuel de la session. Il vous suffit d'y faire appel juste avant de stocker l'identifiant utilisateur ou l'authentification dans la session. Pour faire simple, vous pouvez y faire appel inconditionnellement juste après `session_start()` dans le script qui gère le formulaire de *login*.

**Chiffrement et sécurité**

Le chiffrement est souvent pris comme une solution à tous les problèmes de sécurité. Si le procédé n'amène aucune vulnérabilité, il nous paraît bon de rappeler quelques principes afin que l'outil soit utilisé à bon escient.



## Chiffrement et hachage

Il existe au moins trois procédés qu'on désigne communément par le terme « chiffrement » : le chiffrement avec clé unique, le chiffrement à clé privée et clé publique, et le hachage.

### Hachage

Cette dernière méthode est en réalité du chiffrement selon la définition la plus stricte : la donnée est brouillée pour la rendre illisible. Personne ne peut la relire par la suite, pas même celui qui a opéré la conversion la première fois. On parle aussi de chiffrement à sens unique, pour montrer qu'il est impossible de décoder le résultat. Du fait qu'il n'y a pas besoin qu'une donnée soit décodable, les procédés de hachage peuvent se permettre d'avoir un résultat d'une longueur fixe. Ainsi, une donnée de 650 Mo aura un résultat de 32 caractères après hachage par la méthode MD5.

On peut considérer le procédé de hachage comme l'aboutissement de la sécurité puisque nul n'a de méthode de décodage.

### Chiffrement avec clé

Les deux autres méthodes diffèrent uniquement par le système de clé. Dans la première, il existe une clé unique, qui sert aussi bien à coder qu'à décoder. C'est le procédé utilisé quand celui qui crypte et décrypte est la même personne, ou deux personnes partageant les mêmes droits.

Le système à clé privée permet, lui, de séparer la personne pouvant chiffrer les données de celle qui les décrypte. Ainsi, si je me réserve une clé de décryptage privée et que je diffuse une clé de chiffrement publique correspondante, tout le monde pourra chiffrer des documents, mais je serai le seul à pouvoir les relire. C'est par exemple le procédé utilisé pour les signatures (dans ce cas, c'est le chiffrement qui est privé et le décryptage qui est public).

La sûreté d'un chiffrement à clés dépend de deux critères : la longueur de la clé et sa divulgation. Plus une clé est importante, plus il sera long de décoder les données sans l'avoir. À partir d'une certaine taille, la durée de décodage pour quelqu'un qui n'a pas la clé se chiffre en siècles ; on peut donc considérer que ce n'est pas un point faible.

Le réel problème de ces chiffrements est la sécurité de la clé elle-même. Il est très important de se souvenir que vos données cryptées ont une sécurité limitée par celle de la clé elle-même. Ceci implique un concept très simple : si vous stockez la clé au même endroit que vos données, il sera inutile de faire un chiffrement car celui qui accédera aux données pourra aussi lire la clé et donc les décoder. On dit que la sécurité d'un système est celle de son maillon le plus faible.

### Chiffrement et mots de passe

Les mots de passe sont les données qui sont en premier soumises à chiffrement. Ne pas les laisser en clair est d'une grande importance. Le chiffrement empêche les failles de divulgation. Il nous reste ensuite à choisir si on utilise un hachage ou un système décodable.

De nombreux applicatifs utilisent des procédés décodables pour les mots de passe. Nous vous déconseillons fortement un tel système.

En effet, si quelqu'un accède aux mots de passe, il est tout à fait possible qu'il puisse accéder aussi à vos clés de codage. Vos mots de passe ne sont alors potentiellement pas en sécurité et, à la première vulnérabilité trouvée, ils risquent d'être divulgués. Le hachage n'a pas ce défaut et doit donc être préféré. Les systèmes d'exploitation actuels utilisent d'ailleurs tous ce type de chiffrement pour leurs mots de passe, prenez exemple.

La raison qui pousse souvent les développeurs à choisir la mauvaise solution est la possibilité de rendre son mot de passe à un utilisateur s'il l'a oublié. Ce n'est en fait pas nécessaire puisqu'il suffit d'en calculer un nouveau et de le lui donner lors de la procédure de récupération. Il ne peut de toute façon pas préférer l'ancien puisqu'il l'a oublié. À lui de le changer si le nouveau ne lui convient pas.

Il est de plus important de ne jamais redonner un ancien mot de passe en clair (même dans une procédure de récupération) parce que la plupart des gens utilisent un unique mot de passe pour toutes leurs ressources. Certains même utilisent leur numéro de carte bancaire. Ils ne vous pardonneraient pas d'avoir donné ce mot de passe à un ami ou un voisin qui a utilisé l'ordinateur personnel et qui a lancé une procédure de récupération en profitant de la boîte aux lettres électronique.

### Mot de passe crypté dans un cookie

Il est relativement fréquent de voir des applications envoyant deux *cookies* : l'identifiant de l'utilisateur et le mot de passe. Elles peuvent ainsi vérifier l'authentification à chaque accès. Ces applicatifs cryptent le mot de passe en pensant qu'ainsi personne ne pourra le voir et donc le réutiliser pour accéder à un compte frauduleusement.

C'est une mauvaise réflexion et un effet pervers des possibilités de chiffrement : elles donnent un sentiment de sécurité. En effet, l'attaquant n'a pas besoin de connaître le mot de passe pour avoir accès au compte : il lui suffit de noter la chaîne cryptée et de recréer chez lui un *cookie* identique. Le mot de passe crypté est le bon et l'attaquant passe les barrières de vérification.

Dans une telle méthode, on peut considérer que le mot de passe est en réalité la chaîne cryptée et que la procédure d'authentification par formulaire n'est qu'une fonction de conversion d'un mot mnémotechnique en mot de passe. Il y a donc bien suivant cette logique un mot de passe en clair dans le *cookie*. Le chiffrement n'a amené aucune sécurité, juste une impression.

## Bonnes habitudes

Nous avons décrit les paramètres de configuration auxquels faire attention et les problèmes de sécurité qui peuvent intervenir à cause d'une mauvaise utilisation de PHP. Il reste à parler de tout ce qui n'est pas technique.

## Vérifiez vos résultats

Nous vous avons recommandé plus haut de vérifier toutes les données fournies par l'utilisateur : forme, type, valeur, etc. Cette vérification suffit théoriquement, mais elle ne vous met pas à l'abri. Il vous suffit d'oublier une vérification pour rendre votre applicatif vulnérable.

Dans cette optique, nous vous conseillons de faire régulièrement des vérifications de la cohérence des résultats que vous obtenez. Par exemple, lors d'un calcul de prix, vérifiez après calcul que le prix est bien positif, de l'ordre de grandeur attendu et avec la précision attendue. Si vous vendez des maisons, un prix inférieur à 10 000 € est signe d'un problème. De même, un prix avec des chiffres après la virgule alors que vous ne devriez pas en utiliser est révélateur du fait que quelqu'un cherche à exploiter une faille.

Dès que vous avez un critère, même peu précis, permettant de vérifier la cohérence d'un de vos résultats, mettez-le en œuvre. Il est même possible, si certains résultats sont possibles mais très rares, de les signaler dans un journal de *logs* ou d'envoyer un e-mail à l'administrateur quand ils surviennent. Ainsi, vous pouvez vérifier s'ils arrivent trop souvent ou dans des conditions spéciales, et agir en conséquence.

### Vérifiez vos fichiers de logs et statistiques

Dans la même optique que la vérification de résultats, il est nécessaire de regarder régulièrement les statistiques de votre application.

Dans ces statistiques, vous pouvez porter une attention toute particulière aux statistiques de vente. Par exemple, un client qui fait plusieurs commandes coup sur coup est à surveiller et il peut être une bonne idée d'y regarder de plus près.

De même, le fichier de *log* du serveur web contenant la liste des pages appelées est une très bonne source pour des vérifications. Si, sur une courte période, une page interne est appelée fréquemment par la même IP, c'est potentiellement quelqu'un qui essaye de l'exploiter, ou y réussit. Si vous trouvez des adresses de pages avec des paramètres (données après le point d'interrogation) non prévus ou extrêmement longs, c'est probablement aussi l'exploitation d'une vulnérabilité, ou au moins un essai.

Vous pouvez aussi regarder les adresses IP dans vos statistiques. Si plusieurs clients ont la même adresse IP, ce peut être normal (proxy d'entreprise ou de fournisseur d'accès Internet par exemple), mais ce peut être aussi un visiteur malveillant qui a réussi à faire des commandes sans les payer ou qui exploite une faille de votre application. Si un trafic anormalement élevé vient d'une même adresse IP, ce peut être un moteur de recherche comme un pirate.

L'analyse de vos statistiques et fichiers de *logs* doit être régulière et faite à la main. Des outils peuvent vous aider à récupérer rapidement les parties étranges ou inhabituelles pour vous permettre de les étudier à la main, mais ils ne peuvent que compléter et faciliter une analyse directe par un humain, pas la remplacer. En effet, une personne peut d'un coup d'œil voir certaines choses qu'un programme ne verra pas s'il n'est pas prévu pour cela.

## ***Ne croyez pas l'utilisateur***

L'essentiel de la sécurité repose en réalité sur un point important : toujours se méfier de l'utilisateur et ne jamais lui faire confiance. Volontairement ou pas, s'il y a quelque chose à ne pas faire, un jour ce sera fait. Considérez toujours qu'un utilisateur cherche à exploiter ce que vous lui autorisez et trouvera ce que vous ne voulez pas qu'il trouve.

En conséquence, il est important de toujours prévoir le pire. Il vous faut toujours vérifier ses données, les filtrer, les convertir, vérifier son identité, etc. Cela doit intervenir que vous lui fassiez confiance ou pas, que ce soit pour une application en intranet ou non.

Il est fréquent de voir des applications intranet avec une sécurité faible ou inexistante. Ce manque de sécurité peut se révéler dommageable sur le long terme. On peut par exemple imaginer le cas d'un employé en conflit avec l'entreprise et qui, avant de partir, chercherait à nuire (ce qui malheureusement arrive), ou celui d'une vulnérabilité sur la machine servant de passerelle vers Internet qui permettrait à un extérieur d'utiliser votre application (ce qui arrive encore plus souvent). Il est même possible que, dans le futur, quelqu'un décide de mettre votre application accessible via Internet pour y accéder à distance sans savoir que la sécurité n'y est pas suffisante. Tous ces cas sont tout à fait réels et difficilement prévisibles.

Même si, au moment de la conception, vous avez une totale confiance en vos utilisateurs, concevez la sécurité comme si ce n'était pas le cas. Cela ne le sera pas forcément dans le futur, et il est toujours possible qu'un autre problème permette à un extérieur d'accéder à l'intranet. Faire confiance pourrait alors avoir des conséquences graves.

### **Restreignez les accès**

Toujours dans cette optique de ne pas faire confiance à l'utilisateur, il est important d'avoir des contrôles d'accès au plus strict. N'autorisez quelqu'un à faire quelque chose que s'il en a besoin.

En particulier, ne laissez pas sans nécessité des répertoires accessibles en écriture, un PHP sans le `safe_mode`, l'accès en écriture à la base de données, etc. Il sera toujours temps d'ajouter une autorisation par la suite. Faites de même pour vous : n'autorisez votre compte et vos scripts à faire que ce qui est indispensable.

Outre la confiance en l'utilisateur, cette démarche permet de restreindre les conséquences d'une vulnérabilité dans votre application. Imaginons qu'une faille soit présente : si votre script n'est autorisé qu'à lire la base de données, mais pas à y écrire (ce qui est réservé au script d'administration), vous évitez toute possibilité de corruption et d'effacement des données. De même, si vous avez plusieurs modules, il peut être utile de leur affecter des droits et des espaces séparés afin que la vulnérabilité de l'un d'eux ne puisse pas affecter les autres.

La première chose qu'essaie un pirate quand il a trouvé une faille, c'est de voir si cette faille ne peut pas l'amener à exploiter quelque chose de plus important encore. En bloquant les accès non nécessaires, vous limitez cette escalade de privilèges.

## N'exagérez pas

Malgré tout ce que ce chapitre a pu vous conseiller et vous laisser penser, n'exagérez pas. La sécurité est un point important, mais il est aussi facile de se laisser entraîner à faire des choses qui ne sont pas utiles et qui donnent trop de complexité à l'application.

Ainsi, un site marchand sécurisait à outrance les *popups*. Quand une page devait avoir un lien vers une *popup*, on créait un jeton aléatoire. Le jeton, l'adresse de la *popup* et le numéro de session de l'utilisateur étaient alors insérés en base de données. Le lien créé vers la *popup* était fait avec la syntaxe : `popup/xxxxxxx`, où les `x` représentent le jeton aléatoire. Quand le visiteur accédait à cette adresse, le script `popup.php` récupérait la valeur du jeton et les informations correspondantes en base de données. Si l'utilisateur demandant la page était le même que celui en base de données, alors on faisait un `include()` pour accéder à l'adresse réelle de la *popup*. Ce procédé était outrageusement complexe, côté maintenance et côté performance. Il était de plus très perturbant pour l'utilisateur, car il n'y avait aucune adresse réelle dans la barre d'adresse : juste une chaîne aléatoire ne voulant rien dire. En réalité, ce procédé ne sert pourtant à rien : il aurait été aussi simple que le script réel gérant la *popup* vérifie si l'utilisateur y a bien accès. Plus simple, plus facile à coder, plus facile à maintenir, et tout aussi sûr. La sécurité n'a nullement été augmentée, au contraire, car un code plus complexe et plus difficile à maintenir contient potentiellement plus d'erreurs.

À l'image de cet exemple (réel), n'exagérez pas. Contrôlez les accès et toutes les données utilisateurs partout où vous le pouvez, mais n'utilisez pas des procédés tordus et complexes ; ils se retourneraient contre vous tôt ou tard et sont rarement plus sûrs.

## Faites faire un audit externe

Une fois votre application finalisée et sécurisée, il est peut-être utile de s'assurer qu'on n'a rien oublié. Pour cela, la meilleure manière est de faire relire le code par une personne (ou un groupe) extérieure à celui qui a conçu l'applicatif.

En effet, en étant dans l'équipe de développement, on ne voit généralement pas facilement les comportements imprévus et incorrects. Un regard extérieur n'aura pas ce problème et détectera plus facilement les problèmes de sécurité.

Certains programmeurs tirent profit de cette vue extérieure en codant en deux groupes. Chaque groupe opère sur une partie indépendante du programme et quand des tâches unitaires sont faites, les travaux sont échangés pour faire une relecture et une vérification du code de l'autre.

### Pour en savoir plus

*Sécurité DMPT & MySQL*, D. Seguy, P. Gamech :  
612114, Eyrolles 2007.

# Outils de développement PHP

---

Le mode de développement peut être très variable selon vous-même, selon le contexte dans lequel vous vous trouvez et le programme que vous créez. Généralement les puristes préfèrent utiliser le strict minimum et un simple éditeur de texte avec, éventuellement, une coloration syntaxique leur suffit. D'autres préféreront un outil avec plus de fonctionnalités, on parle alors d'IDE (*Integrated Development Environment*). Enfin, une troisième catégorie s'éloignera encore plus du code et utilisera des outils dits de RAD (*Rapid Application Development*) pour générer tout ou partie du code. Nous allons ici faire un tour d'horizon des principaux outils du marché en nous arrêtant sur certains d'entre eux que nous jugeons particulièrement intéressants.

## Éditeurs de texte & IDE

PHP est un langage extrêmement populaire et il existe donc de nombreux outils pour travailler avec lui. Parmi les solutions, on peut citer en vrac, pour les éditeurs de textes :

- NEdit (<http://www.nedit.org/>, Unix) ;
- Vim, Emacs et leurs dérivés (disponibles aussi sous Windows mais classiques et surtout utilisés dans le monde Unix) ;
- Jext et jEdit (<http://www.jext.org> et <http://jedit.org>, multi-plates-formes) ;
- EditPlus (<http://www.editplus.com/>, Windows) ;
- ConTEXT (<http://www.fixedsys.com/context/>, Windows) ;
- UltraEdit (<http://www.ultraedit.com/>, Windows).

De nombreux outils permettent d'aller plus loin et offrent des fonctionnalités plus poussées. Il s'agit des IDE (*Interface Development Environment*) dont voici une liste non exhaustive :

- PHPEdit (<http://www.phpedit.net/>, Windows) ;
- Eclipse (<http://www.eclipse.org/>, multi-plates-formes) ;
- Zend Studio ([http://www.zend.com/products/zend\\_studio/](http://www.zend.com/products/zend_studio/), multi-plates-formes) ;
- PhpED (<http://www.nusphere.com/>, multi-plates-formes) ;
- Komodo (<http://www.activestate.com/Products/Komodo/>, multi-plates-formes) ;
- KDevelop 3 (<http://www.kdevelop.org/>, Unix).

Enfin, les principaux éditeurs HTML ont eux aussi prit le train du PHP et proposent de plus en plus des extensions dédiées PHP :

- Dreamweaver Mx (<http://www.dreamweaver.com/>, Windows) ;
- Quanta (<http://quanta.sourceforge.net/>, Unix) ;
- BBEdit (pour Macintosh).

Nous n'avons pas ici la prétention de présenter tous ces outils mais nous allons détailler ceux qui nous semblent les plus représentatifs dans leur genre.

## UltraEdit

UltraEdit est un éditeur de code très rapide prisé par de nombreux développeurs. Il ne fonctionne que sous Windows et remplace avantageusement Notepad pour le développement PHP mais également pour d'autres langages.

Dans la même catégorie on retrouve aussi NEdit (un éditeur des plates-formes Unix) ou EditPlus (pour Microsoft Windows)

Voici quelques fonctionnalités qui ont fait le succès d'UltraEdit :

- pas de limitation par rapport à la taille des fichiers ;
- très rapide ;
- coloration syntaxique possible :

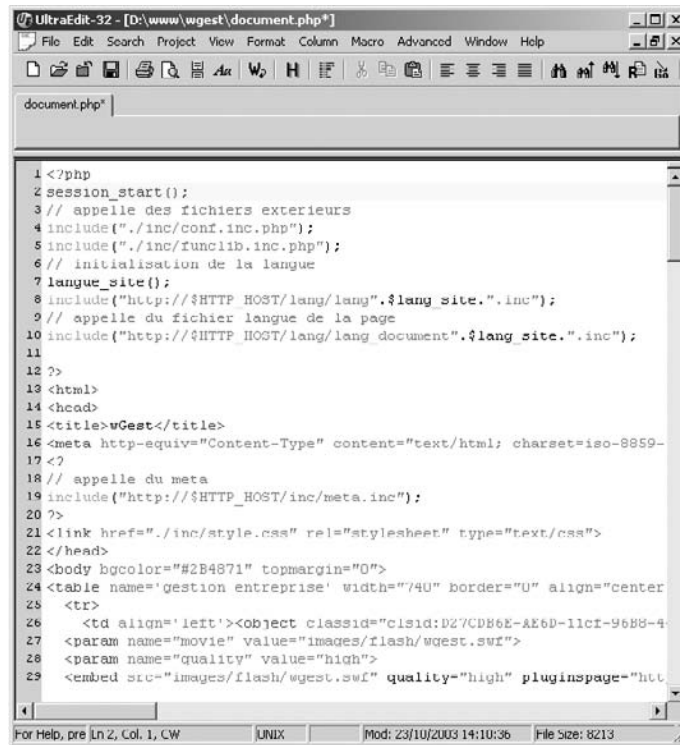
la coloration du code PHP n'est pas fournie par défaut dans toutes les versions. Pour la définir, il faut se rendre sur le site du logiciel (<http://www.ultraedit.com/>), cliquer sur Wordfiles puis sélectionner PHP. Enregistrez le texte et copiez-le dans le fichier wordfile.txt présent dans le répertoire où a été installé UltraEdit. Ensuite vos fichiers dont l'extension est .php seront colorés.

- complétion automatique :

pour ajouter la complétion des fonctions PHP il faut vous rendre sur le site du logiciel (<http://www.ultraedit.com/>), cliquer sur Wordfiles et vous rendre dans la section Autocomplete Files. Choisissez PHP et enregistrez le fichier dans le répertoire d'UltraEdit.

Allez ensuite dans le menu de configuration et indiquez dans le champ Auto Complete File le chemin d'accès au fichier que vous venez de sauvegarder (voir figure 28-2).

Figure 28-1  
Présentation globale  
d'UltraEdit

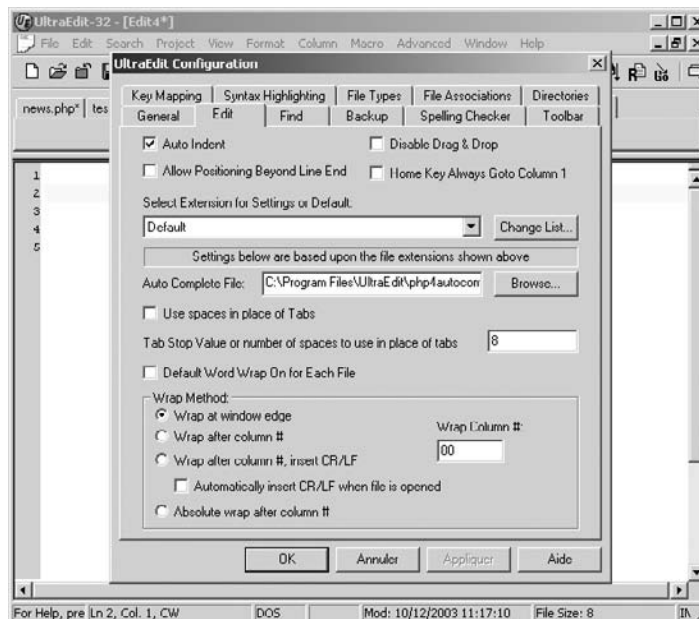


```
UltraEdit-32 - [D:\www\wgest\document.php*]
File Edit Search Project View Format Column Macro Advanced Window Help
document.php*

1 <?php
2 session_start();
3 // appelle des fichiers externes
4 include("../inc/conf.inc.php");
5 include("../inc/funclib.inc.php");
6 // initialisation de la langue
7 langue_site();
8 include("http://$HTTP_HOST/lang/lang".$lang_site.".inc");
9 // appelle du fichier langue de la page
10 include("http://$HTTP_HOST/lang/lang_document".$lang_site.".inc");
11
12 ?>
13 <html>
14 <head>
15 <title>wGest</title>
16 <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
17 <?
18 // appelle du meta
19 include("http://$HTTP_HOST/inc/meta.inc");
20 ?>
21 <link href="../inc/style.css" rel="stylesheet" type="text/css">
22 </head>
23 <body bgcolor="#2B4871" topmargin="0">
24 <table name='gestion entreprise' width=""/40" border="0" align="center
25 <tr>
26 <td align='left'><object classid="clsid:D27CDB6E-AE6D-11C1-96B8-4
27 <param name="movie" value="images/flash/wgest.swf">
28 <param name="quality" value="high">
29 <embed src="images/flash/wgest.swf" quality="high" pluginspage="htt

For Help, pre Ln 2, Col. 1, CW UNIX Mod: 23/10/2003 14:10:36 File Size: 8213
```

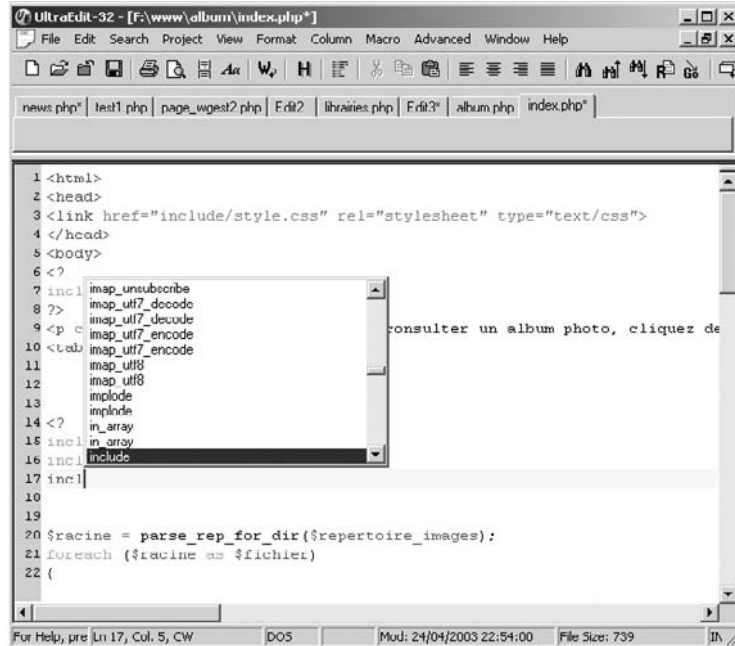
Figure 28-2  
Installation de la  
complétion sous  
UltraEdit





Pour utiliser cette fonctionnalité, il vous suffit alors d'appuyer simultanément sur les touches Ctrl et Espace pour que s'affiche la liste des fonctions ressemblantes.

**Figure 28-3**  
*Auto-complétion  
pour UltraEdit*



- possibilité d'ajouter des modules ;
- protection des fichiers :  
par défaut UltraEdit génère des sauvegardes de sécurité en enregistrant vos fichiers sous l'extension .bak. C'est un danger potentiel car vous pouvez être amené à les déposer sur votre serveur web sans faire attention à leur extension, or ils peuvent être lus... Il faut donc spécifier « Pas de sauvegarde » dans Avancé > Configuration > Sauvegarde de sécurité. (voir figure 28-4)

De plus UltraEdit offre de nombreuses configurations (jeux de caractères, terminaison de lignes, etc.) couvrant l'essentiel des besoins.

## PHPEdit

PHPEdit est un environnement de développement pour PHP ne fonctionnant que sous Windows mais l'équipe de développement annonce une version Linux. Il propose de nombreuses fonctionnalités qui sont pour certaines introuvables dans les autres outils. Nous allons détailler ici les principales en basant nos tests sur la version 0.8. Autre point important, le projet PHPEdit est géré par le PHPEdit group, composé principalement de français. (voir figure 28-5)

Figure 28-4  
Les backup  
d'UltraEdit

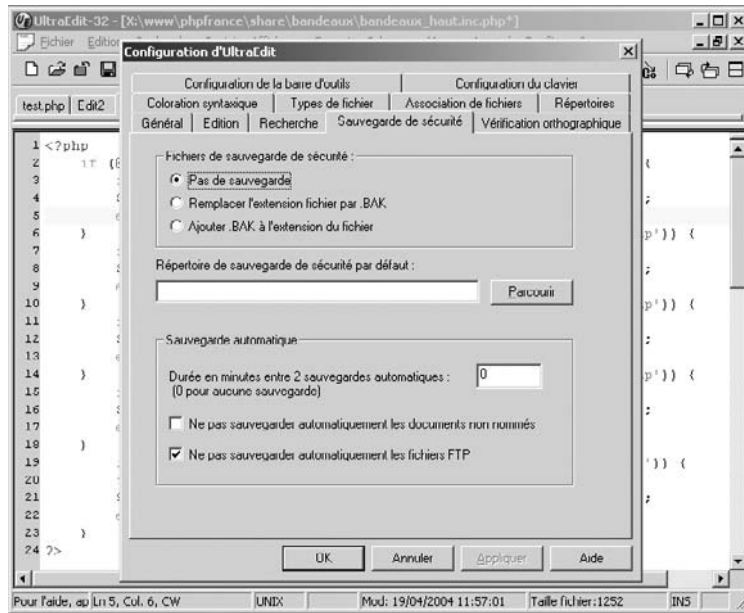
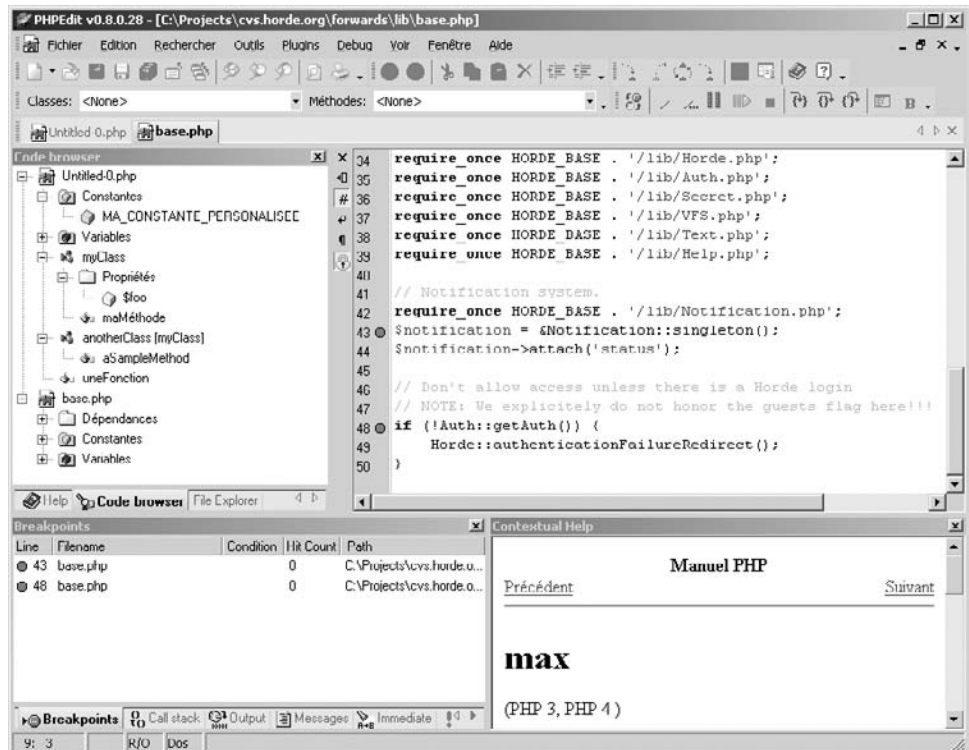


Figure 28-5  
Interface  
principale  
de PHPEdit



PHPEdit a été primé par le *PHP Magazin* comme étant le meilleur IDE.

#### Remarque

L'équivalent Unix le plus proche est probablement KDevelop dans sa troisième version.

Pour commencer, voici à quoi ressemble l'interface principale. Comme vous pouvez le constater, elle est beaucoup plus riche que celles des éditeurs de textes améliorés et relativement agréable.

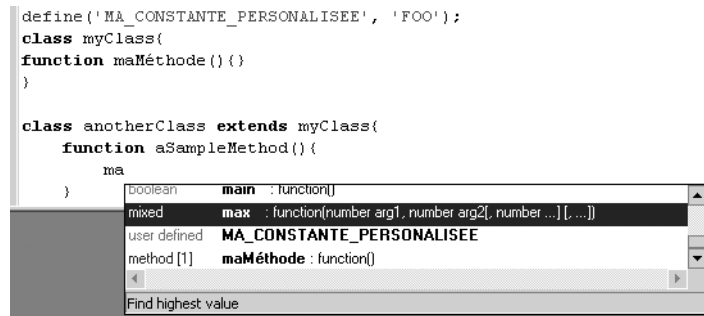
PHPEdit intègre les fonctions classiques et indispensables à tout IDE qui se respecte.

### Code Insight

Vous propose une liste déroulante des éléments correspondant à la fois au contexte et à votre application. Le contexte signifie que la liste ne comportera pas les mêmes éléments si vous vous trouvez à l'intérieur d'une classe ou dans une fonction ; dans le premier cas vous allez voir les éléments spécifiques à la classe. Cette liste se base sur le contenu de votre application en vous proposant les constantes, fonctions, classes, attributs, méthodes et variables que vous avez déclarés, comme vous pouvez le constater sur l'exemple suivant :

Figure 28-6

Utilisation  
de Code Insight  
pour PHPEdit



### Code Hint

Vous propose une aide contextuelle sur les paramètres de la fonction PHP que vous utilisez. Exemple :

Figure 28-7

Code Hint  
de PHPEdit

```

class anotherClass extends myClass{
    f mixed max(number arg1, number arg2[, number ...] [, ...])
        max (| ▲ ▼ ▲)
}
    
```

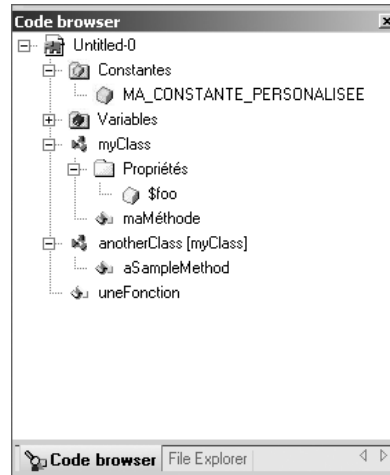
### L'explorateur de code

Il vous permet d'avoir une vision synthétique des éléments contenus dans les fichiers de votre application. De plus, il propose les constantes, variables, classes, opérations et attributs.

L'affichage de chaque type d'éléments peut être configuré pour correspondre à vos attentes. En double-cliquant sur un élément vous déplacerez votre curseur sur sa définition. Dans le cas des dépendances (`require`, `include`, `require_once`, `include_once`), en double-cliquant sur l'élément vous ouvrirez le fichier en question.

Figure 28-8

Explorateur de code  
de PHPEdit



## QuickMark

Outre ces fonctions classiques, il y a bien entendu la coloration syntaxique et quelques autres plus qui devraient vous faire apprécier cet outil. Cependant, il y a aussi des fonctionnalités plus poussées.

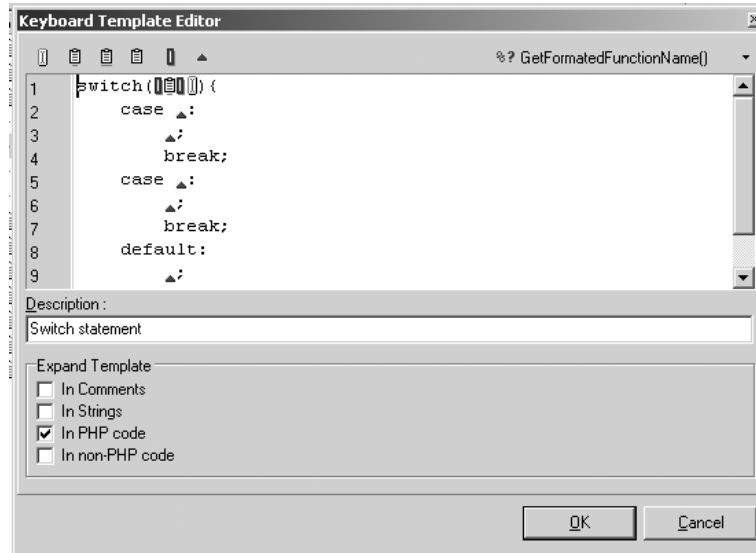
En effet, PHPEdit propose deux outils pour vous permettre de naviguer dans votre code : les signets (*bookmarks*) et des QuickMark. Les QuickMark sont beaucoup plus volatiles et vous permettent de naviguer très rapidement dans le document. À l'inverse des signets, ils sont à usage unique et une fois que vous les avez utilisés, ils disparaissent. Ils sont donc très pratiques pour revenir rapidement à une partie de code après avoir copié ou copié un autre morceau.

## Macros

PHPEdit intègre un système de macro (Keyboard Template) très puissant vous permettant de déclencher le remplacement d'un morceau de code par un autre plus long. Par exemple en écrivant `switch`, l'IDE va écrire pour vous un prototype de `switch` comme montré dans la figure 28-9. Ces macros peuvent interagir avec le presse-papiers, définir l'emplacement du curseur après le remplacement, positionner des QuickMark, faire appel à des fonctions prédéfinies (`date`, fonction courante, utilisateur connecté...) vous offrant un réel outil d'optimisation de votre développement.

Figure 28-9

Éditeur de macros  
de PHPEdit



### Générateur d'aide

PHPEdit propose également un générateur d'aide intégré, dans le style de phpDocumentor (<http://www.phpdoc.org>). Cela vous permet de générer une documentation de l'API de votre application très rapidement et simplement pour la fournir avec l'application à votre client ou à vos collaborateurs.

### Formatage de code

Une autre fonction très appréciable est le module de formatage du code (phpCodeBeautififier) permettant d'unifier les différents styles de codage entre les développeurs d'un projet. De nombreuses options vous permettent de l'ajuster à votre propre convention de codage.

#### Remarque

Le framework PEAR propose une norme de codage. Il est possible de configurer PHPEdit pour s'y conformer.

### Débogueur

Enfin le débogueur vous permet d'investiguer les problèmes de votre application.

Il vous permet :

- de poser des points d'arrêt ;
- d'évaluer le contenu de certaines variables ;
- de les modifier en cours d'exécution ;

- de consulter la liste des variables locales et globales et leurs valeurs.

Vous pouvez également consulter la liste des erreurs générée par votre application et voir la pile d'appels au fur et à mesure de l'avancée de l'exécution de votre application. Pour contrôler cette exécution vous avez accès aux fonctions de pas-à-pas simple, en entrant dans la fonction ou jusqu'à la sortie de la fonction. Ce module est basé sur DBG qui doit être configuré sur votre serveur PHP. Le programme d'installation vous propose d'installer une version de PHP locale avec DBG préconfiguré pour vous permettre un débogage dès l'installation.

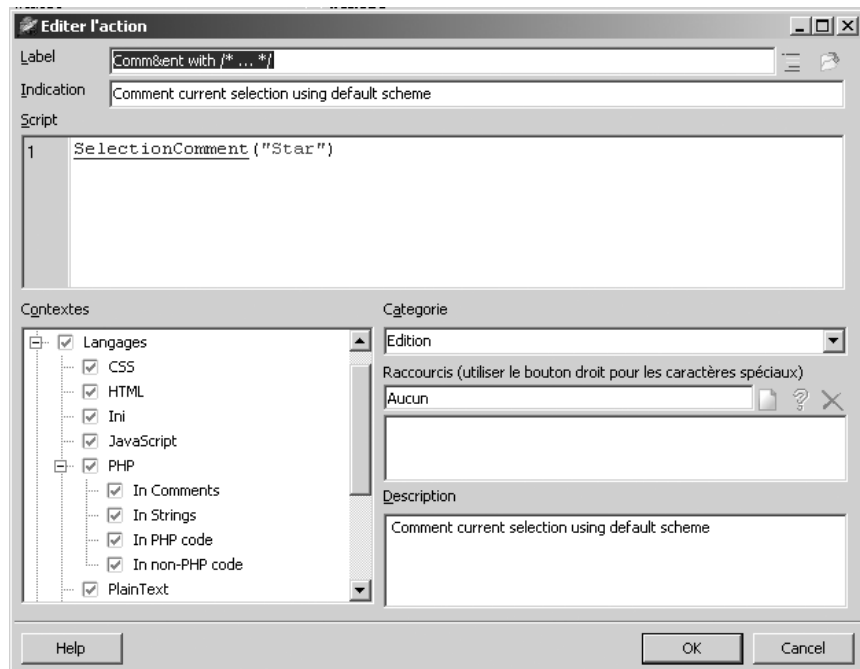
### Conseil

Si vous n'avez jamais utilisé de débogueur avec PHP, testez-le, c'est une petite merveille.

### Interface de modules

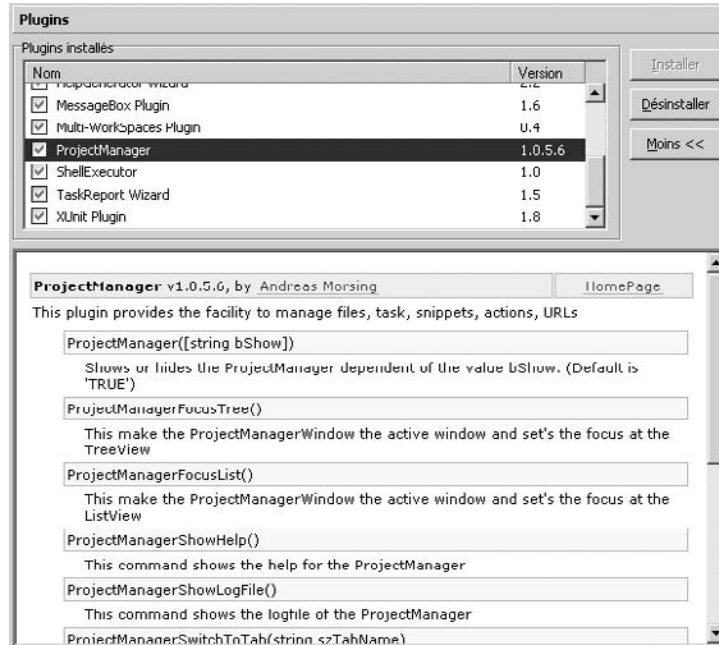
Ce qui est très confortable lors de l'utilisation de cet outil, c'est sa capacité de personnalisation : tous les panneaux sont aménageables à souhait pour vous permettre d'arranger l'interface comme bon vous semble, mais ce n'est pas tout ; toutes les fonctionnalités de l'application sont mises à disposition au travers de commandes pour lesquelles l'utilisateur peut lui-même configurer les raccourcis clavier. Toutes les barres d'outils et les menus sont configurables comme Office. Voici le dialogue d'édition d'une action, tout est paramétrable : nom, icône, traitement, contexte, raccourcis...

**Figure 28-10**  
*Éditeur d'actions  
de PHPEdit*



Pour vous permettre de le personnaliser encore plus, l'outil offre une interface de plug-ins. Comme vous pouvez le constater sur la capture d'écran suivante, plusieurs plug-ins ont déjà été développés et sont disponibles sur le site : <http://www.phpedit.net/products/PHPEdit/exchange/>.

**Figure 28-11**  
Éditeur de plug-ins  
de PHPEdit



PHPEdit est un environnement de développement puissant, français et reconnu de façon internationale grâce aux nombreuses fonctions qu'il intègre. Il n'a pas grand-chose à envier aux autres applications du même type, disponibles sur le marché. On peut lui reprocher un manque de qualité et une version stable datant de plus de deux ans, mais ces problèmes sont en cours de traitement par un gel des évolutions de l'application pour la publication d'une nouvelle version stable. Pour plus de détails sur l'application et la tester, rendez-vous sur le site <http://www.phpedit.net/>

## Eclipse

Eclipse est un environnement de développement particulièrement prisé par les utilisateurs de Java. Eclipse a été mis à la disposition de la communauté par IBM et depuis de nombreux add-ons ont été développés. Celui qui nous intéresse est l'add-on PHPEclipse qui permet de disposer de fonctionnalités avancées dans cet environnement.

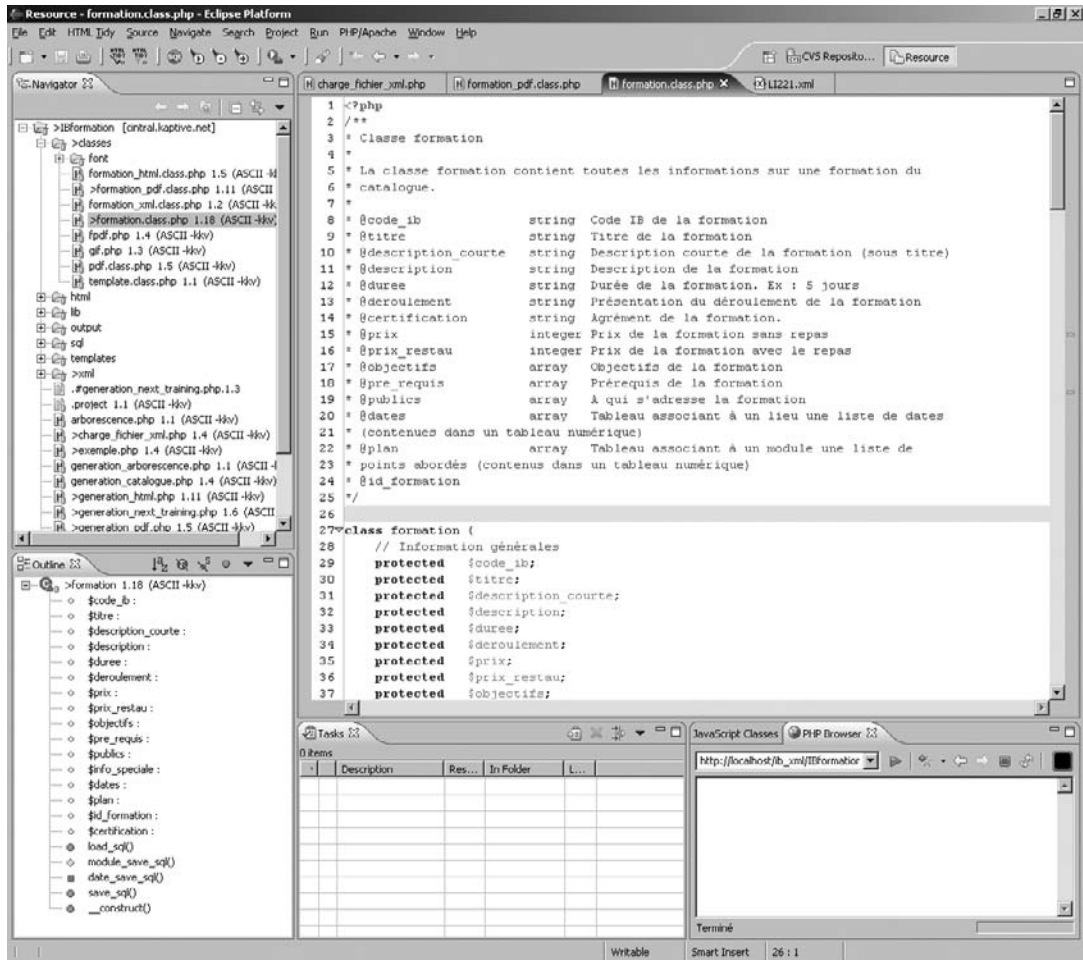
Vous y trouverez la classique colorisation syntaxique, les différents outils de recherche, un explorateur de code, une aide en ligne contextuelle, un débogueur, des possibilités de connexion FTP ou SFTP, des aides à la frappe et à la complétion automatique ainsi que tout ce que fait habituellement un IDE.

**Note**

Installer et configurer l'ensemble de votre plate-forme de développement peut être un peu pénible. Qu'à cela ne tienne, utilisez les paquets PHP pré-installés d'easyeclipse.

**Le débogueur**

Le débogueur est un outil permettant de gagner du temps lors de vos développements. Il vous est possible de mettre un débogueur (dbg) avec PHPEclipse.

**Figure 28-12**

Vue globale d'Eclipse avec le module PHP



## Travailler en équipe avec Eclipse

Eclipse vous permet de travailler avec CVS qui est un outil de gestion de version. La facilité d'utilisation du module CVS est un atout non négligeable. Un module pour Subversion et la plupart des autres outils de versionnement sont aussi disponibles.

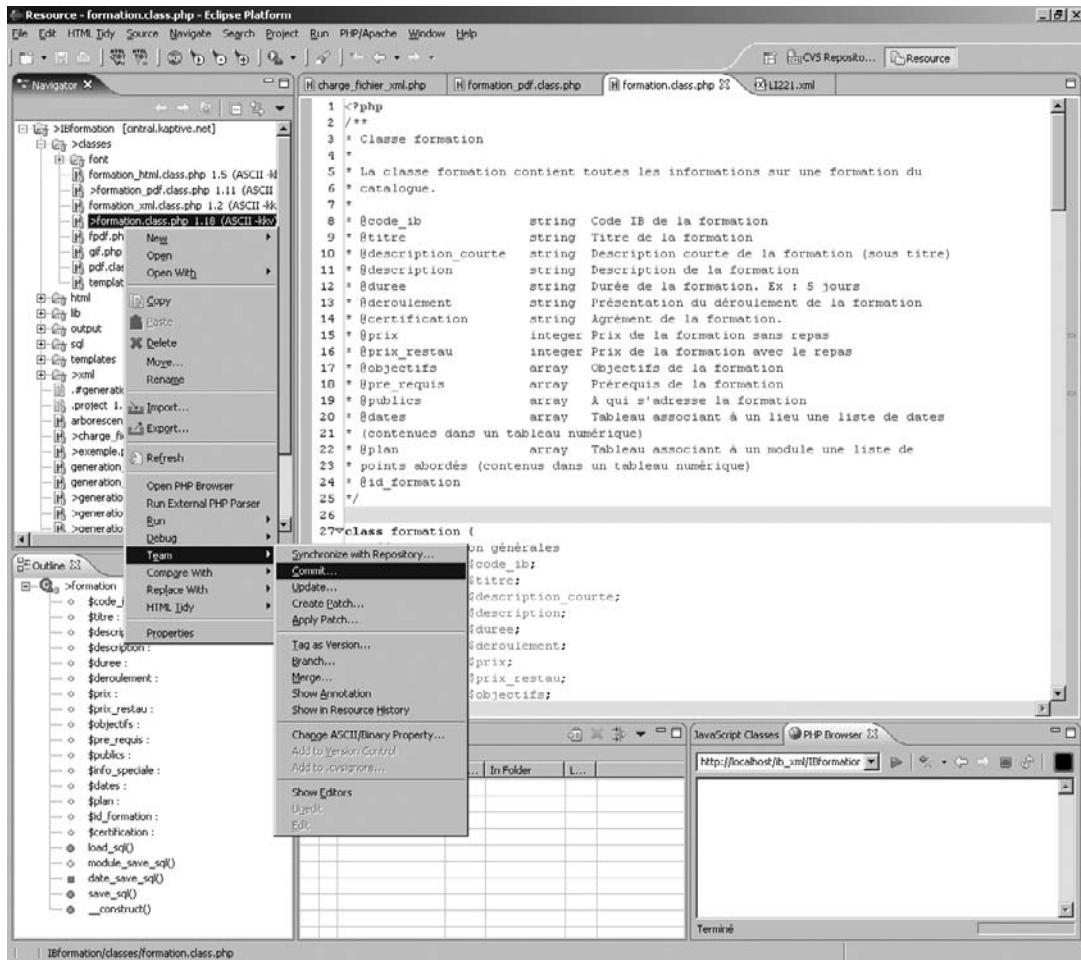


Figure 28-13

La gestion utilisateur de CVS

## Le Zend Studio

Le Zend Studio est la solution commerciale de développement PHP proposée par la société Zend. À ce jour, développé en tant que programme complet, le Zend Studio est voué à migrer en un module d'Eclipse.

### Rapidité

Contrairement aux éditeurs simples tels que VI, Emacs voire UtraEdit, le Zend studio est un environnement de développement complet, il est donc plus lent à démarrer. Il faut éviter de l'ouvrir et de le fermer comme on pourrait le faire avec des programmes plus légers. Un ordinateur disposant d'au moins 256 Mo de mémoire vive est requis, 512 Mo à 1 Go sont conseillés.

Comme vous pouvez le voir dans la figure 28-14, le Zend Studio dispose d'une interface complète.

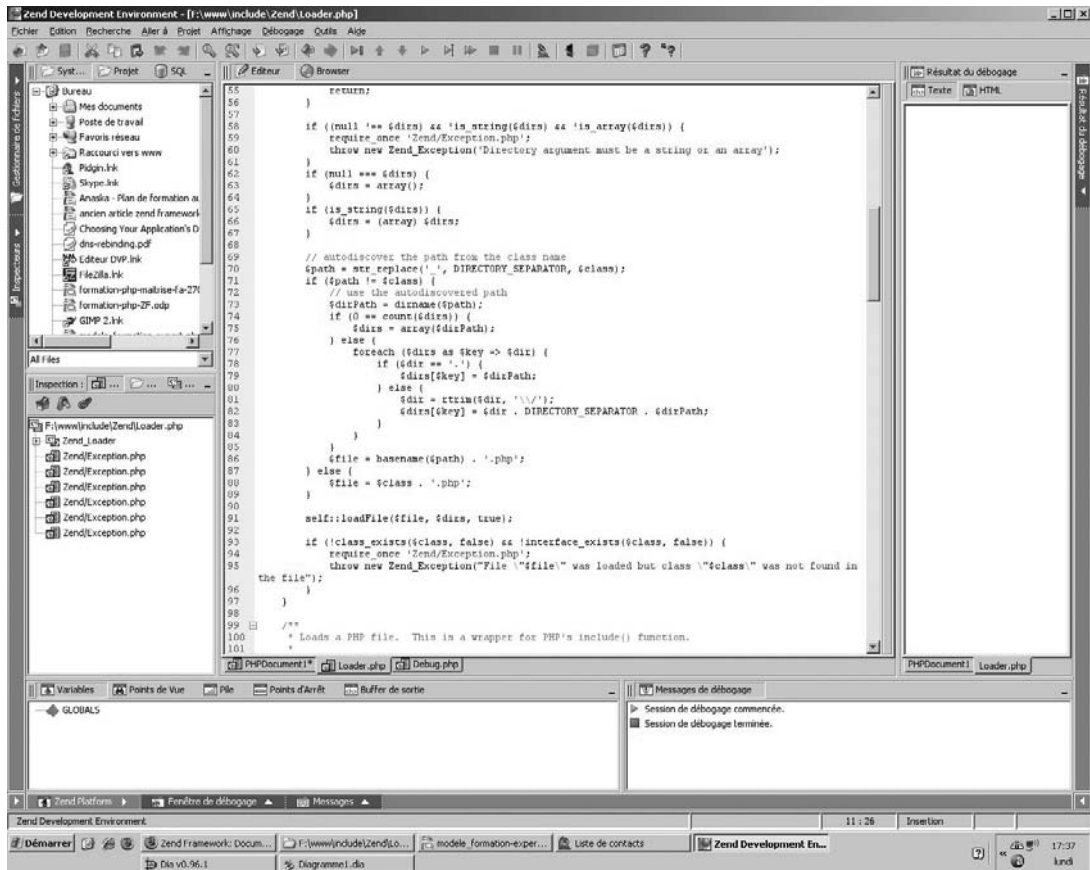


Figure 28-14

Vue globale du Zend Studio

Il fournit de nombreuses fonctionnalités intéressantes pour des développements pointus. Parmi ces fonctionnalités nous pouvons compter les suivantes.

## Le débogueur interne

Le débogueur est un outil permettant de gagner du temps lors de vos développements. Il vous permet de poser des points d'arrêts, d'ajouter des points de vue, de définir l'url faisant appel au programme, etc.

## Un analyseur de code

L'analyseur de code est un outil permettant de mettre en exergue les différentes erreurs de code que vous pourriez avoir faites. Vous pourriez par exemple voir que telle ou telle partie de votre code n'est pas accessible, voir que vous utilisez une variable qui n'a jamais été initialisée, etc.

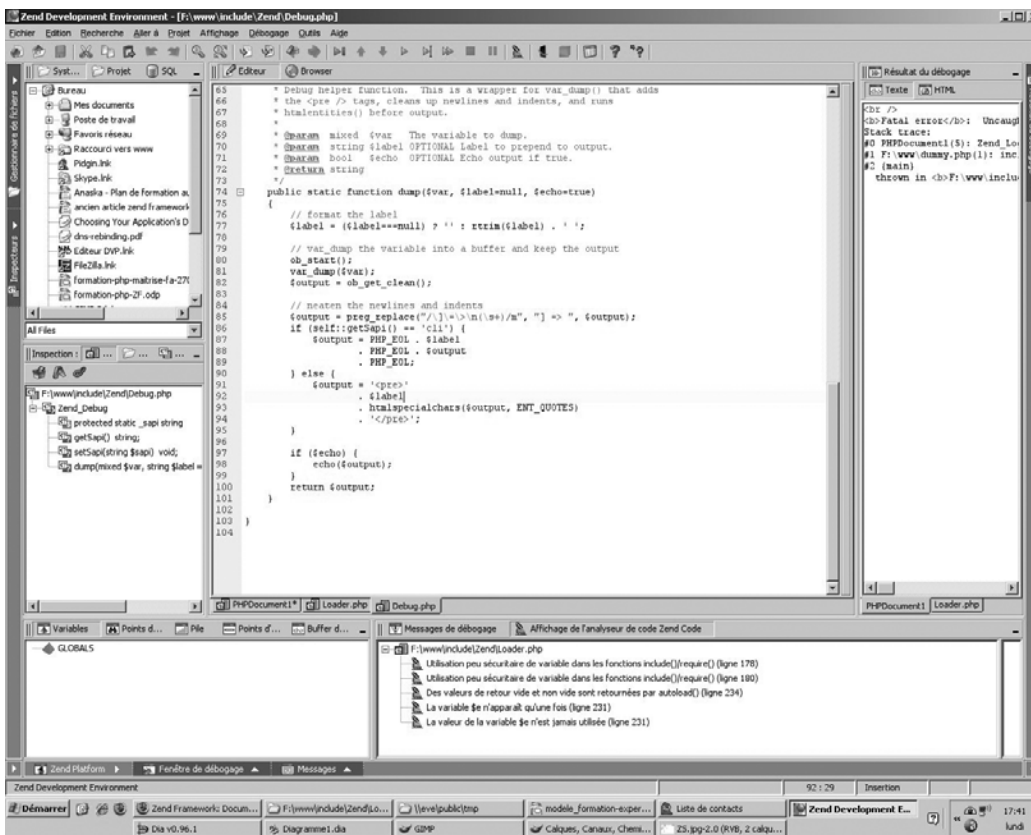


Figure 28-15

*L'analyseur de code du Zend Studio*

Cet outil est un plus important dans l'utilisation du Zend Studio car il vous permet de vous auto-former, et dans le cas d'une analyse de code existant, il vous permettra de voir les éventuels problèmes.

## Reconnaissance des dernières syntaxes

Les équipes de Zend travaillent au développement du langage PHP et particulièrement sur son cœur : le Zend Engine. Ils sont donc particulièrement au courant des évolutions de PHP, ce qui leur permet d'avoir toujours un pied d'avance dans les syntaxes.

## Travailler en équipe avec le Zend Studio

Le Zend Studio vous permet de travailler avec CVS qui est un outil de gestion de version. Couplé au gestionnaire de projet, cela vous donne un outil complet.

## Complétion de code

Bien sûr, le Zend Studio permet aussi de gérer la complétion de code. Ainsi toutes les fonctions commençant par le mot que vous avez tapé vous sont proposées dans un menu déroulant. Le système améliore les solutions proposées.

Un point spécifique au Zend Studio est qu'il propose en complétion non seulement les fonctions natives de PHP mais également les fonctions définies par l'utilisateur, des variables utilisateur ou des méthodes et propriétés lors de la programmation orientée objet.

# Les outils de modélisation/RAD

## *Macromedia Dreamweaver MX*

Dreamweaver est l'un des produits phares de la société Macromedia. Il permet de gérer dans son ensemble la création de sites web. Après s'être longtemps cantonné aux outils propriétaires, Macromedia, avec les versions MX, a implémenté des fonctionnalités permettant d'exploiter une partie de la puissance du couple PHP/MySQL. L'objectif étant de développer tout un site web sans avoir besoin d'écrire la moindre ligne de code. Ce mode de réalisation ravira les graphistes qui pourront ainsi créer des applications dynamiques simples. D'un autre côté les « hard-codeurs » n'y trouveront pas leur bonheur mais peut-être un léger gain de temps sur les réalisations les plus simples qui leur sont demandées.

Nous allons ici faire une présentation succincte de l'utilisation de Dreamweaver pour générer du PHP mais nous invitons les graphistes qui le souhaitent à consulter l'ouvrage *PHP/MySQL avec Dreamweaver MX 2004* de Jean-Marie Defrance dans la même collection.

## Les fonctions de base

Les fonctionnalités les plus simples sont la gestion de l'instruction echo et la mise en place de commentaires.

Viennent ensuite des outils un peu plus poussés tels que le couple `if() {}else{}` et l'inclusion de fichiers.

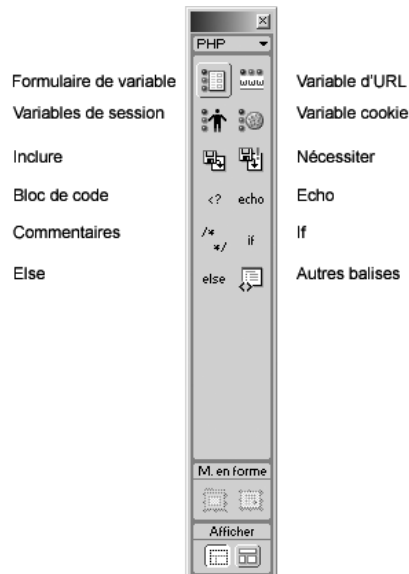
Outre ces quelques outils, les principales fonctionnalités correspondent aux données recueillies par formulaire, par cookie et par session. Il est ainsi possible d'accéder aux données envoyées par la méthode POST ou par la méthode GET. La syntaxe utilisée est :

```
<?php
$HTTP_POST_VARS[toto];
$HTTP_GET_VARS[identifiant];
?>
```

Cette syntaxe a l'avantage d'être compatible avec les anciennes versions de PHP. Ce sont en effet ces noms qui étaient utilisés à la place de \$\_GET et \$\_POST dans les premières versions de PHP 4. Cette syntaxe reste toutefois compatible avec PHP 5, la seule différence est que ces variables sont des globales classiques (si vous voulez les utiliser dans une fonction, il faudra les déclarer comme globales explicitement).

**Figure 28-16**

*La barre d'outils  
PHP de  
Dreamweaver*



De la même façon, il est possible de récupérer la valeur d'un cookie et d'une variable de session avec \$HTTP\_COOKIE\_VARS ou \$HTTP\_SESSION\_VARS.

**Note**

Avec la version Dreamweaver MX 2004 on passe en notation superglobale type. C'est-à-dire que vous n'aurez plus \$HTTP\_GET\_VARS[] mais \$\_GET[].

Bien que cet ensemble de fonctions soit relativement limité, associé aux autres, telle que la gestion MySQL, il vous permettra de gérer effectivement, sans toucher le code, des développements simples.

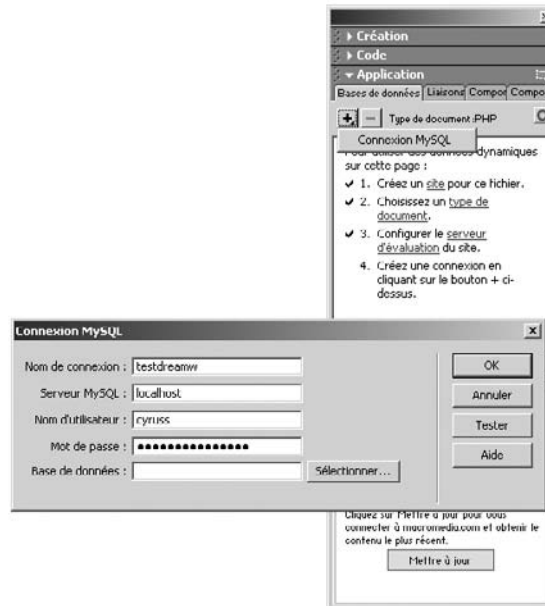
## Utiliser MySQL

Utiliser MySQL avec PHP relève du jeu d'enfant. La première étape consiste à définir les paramètres de votre base de données.

Allez dans l'espace fenêtre \bases de données. Si ce n'est pas encore fait, créez un site et configurez votre serveur d'évaluation. Cliquez ensuite sur l'icône + et sélectionnez Connexion MySQL. Il ne vous reste plus qu'à remplir les champs et à sélectionner votre base de données comme indiqué dans la figure 28-17.

Figure 28-17

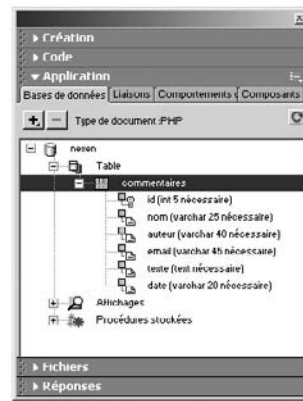
*Paramètres de connexion MySQL*



Une fenêtre vous permet alors de visualiser vos différentes tables (voir figure 28-18).

Figure 28-18

*Les tables et les champs de votre base de données*



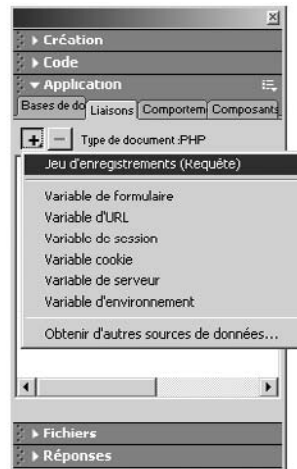
À cette étape, Dreamweaver a automatiquement créé pour vous un fichier de configuration et de connexions dans le répertoire *Connections*. Ce fichier contient les informations que vous avez définies :

```
<?php
# FileName="Connection_php_mysql.htm"
# Type="MYSQL"
# HTTP="true"
$hostname_testdreamw = "localhost";
$database_testdreamw = "testos";
$username_testdreamw = "cyruss";
$password_testdreamw = "xxxxxx";
$testdreamw = mysql_pconnect($hostname_testdreamw, $username_testdreamw,
    $password_testdreamw) or die(mysql_error());
?>
```

Maintenant que la connexion est initialisée nous allons pouvoir définir les données que l'on veut afficher. Pour cela, on clique sur l'icône + présente dans l'onglet *Liaison*, et on choisit *Jeu d'enregistrements* (voir figure 28-19).

Figure 28-19

*Créer une requête*

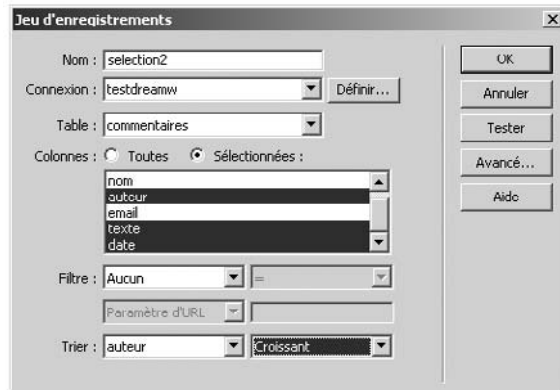


Vous aurez alors à définir les paramètres de votre requête. Comme indiqué dans la figure 28-20, il est possible de sélectionner certains champs de la table, de filtrer ou trier les données.

Il vous suffit alors de déplacer vos champs à l'endroit où vous souhaitez qu'ils apparaissent (figure 28-21). L'onglet *Comportement* vous permet également de disposer de fonctionnalités supplémentaires :

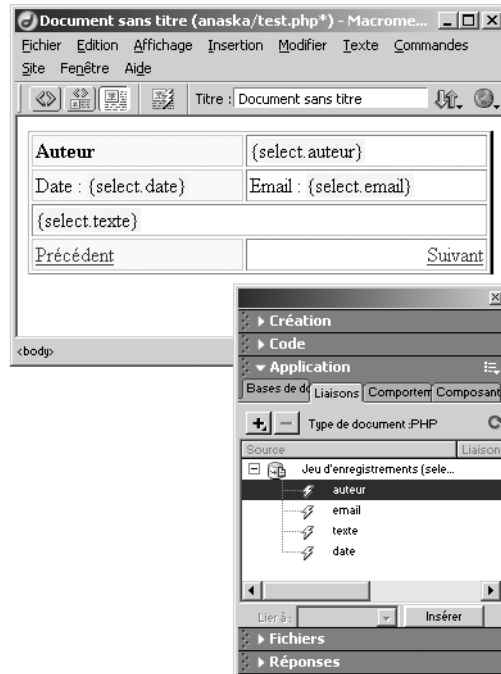
- page suivante/précédente ;
- nombre d'enregistrements ;

Figure 28–20  
*Créer une requête*



- mise à jour des enregistrements ;
- suppression ;
- etc.

Figure 28–21  
*Déplacement des champs de données*

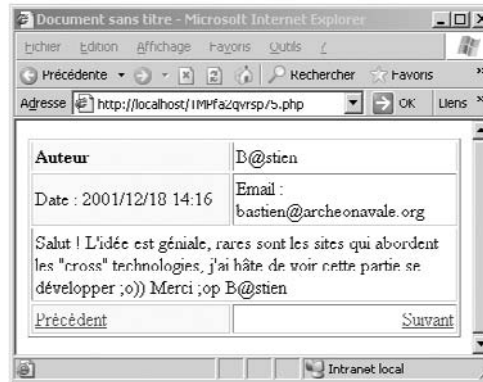


Comme toujours, en utilisant la touche F12 vous accédez au résultat de votre composition (voir figure 28-22).



Figure 28–22

Résultat de la composition sous Dreamweaver



## Conclusion

Dreamweaver est un outil qui permet aux graphistes et aux néophytes en PHP de développer facilement des sites dynamiques simples. À cet égard, c'est un outil de RAD car il induit des gains de temps importants. Cependant, dans sa version 2002 il reste de nombreuses limitations quant à l'utilisation de PHP en tant que plate-forme et pour développer des applications poussées. La version MX 2004 apporte quelques fonctionnalités supplémentaires permettant notamment de gérer l'authentification d'un utilisateur grâce aux sessions. L'outil est donc sur la bonne voie mais il reste du chemin à faire pour qu'il soit adapté aux développeurs.

## WaterProof ::UML

WaterProof::UML est un logiciel commercial de modélisation UML dédié à PHP. Il vous propose de travailler sur une vue graphique de votre application en utilisant les diagrammes de classes du standard UML, ceci avec efficacité et en toute simplicité. Pour ce faire, il propose deux approches complémentaires.

Le logiciel est développé par des membres du PHPedit Group. Le logiciel est payant mais reste dans des domaines raisonnables (aux alentours de 50 €), de plus ses auteurs (français) offrent des licences dans le cadre de projets Open Source.

### Approche nouveau projet

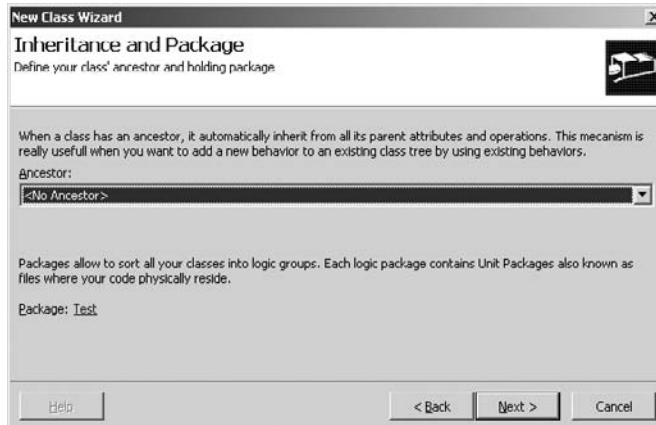
Vous pouvez l'utiliser sur un nouveau projet en commençant par modéliser votre application en définissant vos interfaces, classes, attributs, opérations, attributs et liens d'héritage. Au fur et à mesure que votre diagramme se complète, vous avez en temps réel des aperçus du code correspondant à vos éléments. Vous pouvez générer, quand vous le souhaitez, tout le code correspondant à votre application; WaterProof::UML est très rapide, la génération prend quelques secondes pour des applications de très grande taille.

Voici par exemple l'assistant vous guidant lors du processus de création d'une classe :

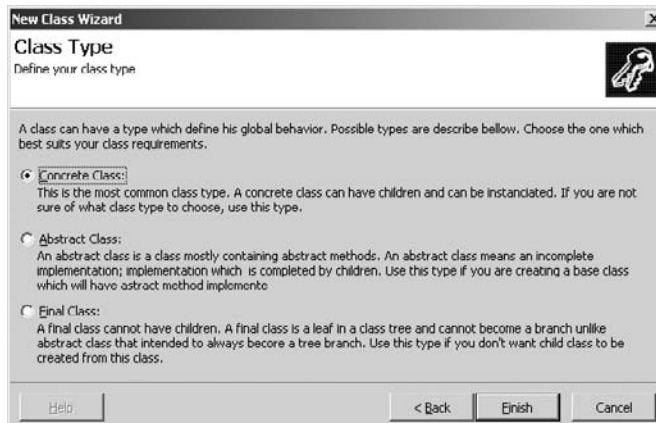
**Figure 28-23**  
*Création de classes*  
*sous*  
*WaterProof::UML*



**Figure 28-24**  
*Création de classe*  
*WaterProof::UML,*  
*page 2*



**Figure 28-25**  
*Création de classe*  
*WaterProof::UML,*  
*page 3*



Et voici l'interface de l'application avec la classe que l'on vient de créer.

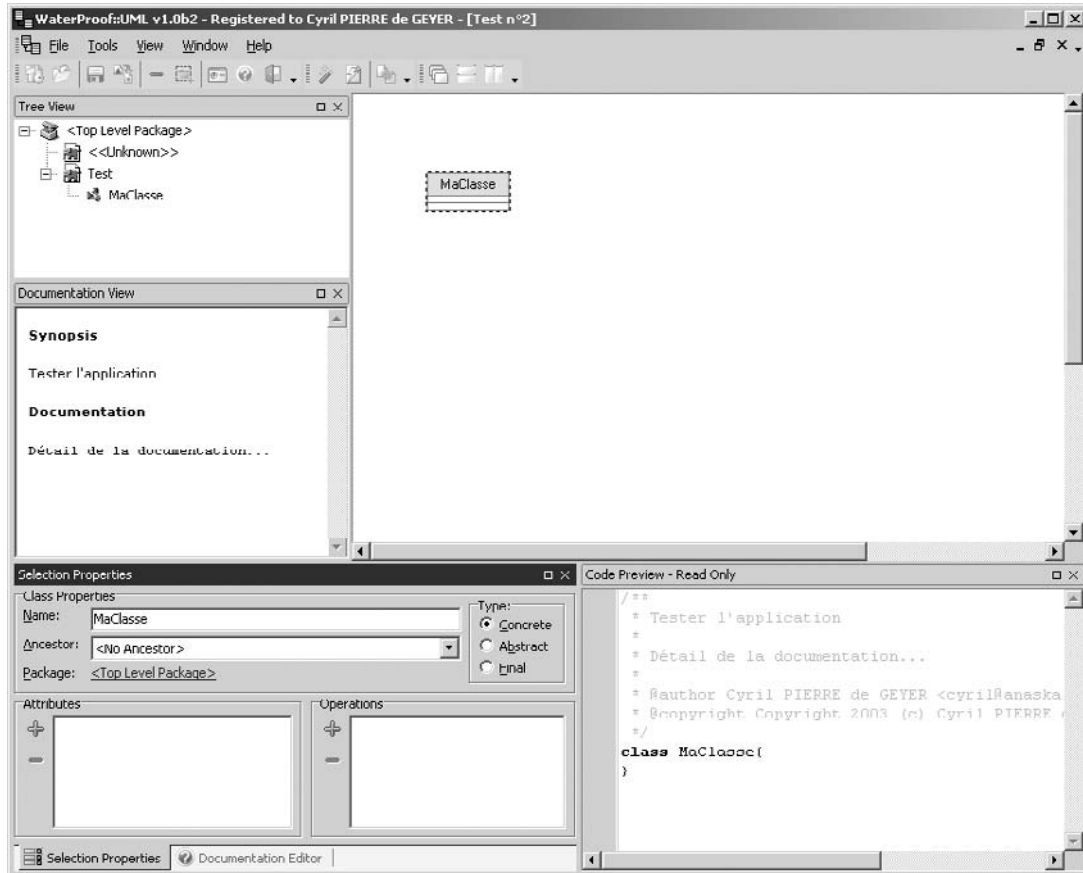


Figure 28-26

*Classe générée par WaterProof::UML*

Comme vous pouvez le constater, la documentation est visible en un coup d'œil et un éditeur spécifique des propriétés de la classe est disponible pour vous permettre d'ajouter opérations et attributs. La partie Code Preview (en bas à droite de la capture d'écran) vous affiche en permanence le code de l'élément sélectionné ; en temps réel reflétant les modifications que vous effectuez sur votre modèle. L'arbre de navigation en haut à gauche permet de naviguer très rapidement dans les différents éléments de votre application.

## Approche rétro-ingénierie

### Remarque

Par rétro-ingénierie il faut comprendre *reverse engineering*.

WaterProof::UML vous propose également une fonctionnalité inédite de rétro-ingénierie de code PHP; c'est la seule application sur le marché à le proposer. Cela vous permet de reconstruire le diagramme de classe correspondant au code de l'application, ce qui s'avère très pratique quand vous devez travailler sur une application que vous ne connaissez pas ou sur laquelle vous avez travaillé plusieurs mois auparavant. En effet, vous allez pouvoir explorer toutes les classes de l'application de façon visuelle en consultant très facilement la documentation de chacun des éléments présents. Là encore, il est très rapide, par exemple l'analyse du code de la librairie graphique JPGGraph (700 Ko de fichiers PHP) prend moins de 10 secondes.

### PHP 4/PHP 5

Waterproof ::UML supporte à la fois PHP 4 et PHP 5, c'est-à-dire que vous pouvez non seulement générer du code directement en PHP 4 et PHP 5 en changeant une simple option, mais également recréer un diagramme à partir d'une application sans vous soucier de la version dans laquelle elle a été écrite.

Voici un exemple d'utilisation du module de rétro-ingénierie; il suffit lors de la création d'un projet de préciser que du code existe déjà. En quelques secondes, l'importation sera réalisée et grâce au module de mise en page automatique, toutes vos classes seront organisées pour vous permettre de naviguer dans le diagramme plus facilement.

Figure 28-27

Création d'un nouveau projet

New Project Wizard

Welcome to the New Project Wizard

This wizard will assist you during the required steps to create a new project

Specify here base parameters of your project. This data will be used during your project life. You can edit those data in the project settings dialog.

Project Name:  
Test de WaterProof UML

Author: Cyril PIERRE de GEYER Email: cyril@anaska.com

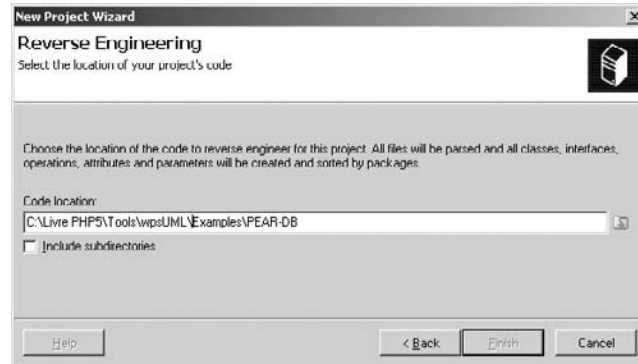
Copyright: Copyright 2003 (c) Cyril PIERRE de GEYER <cyril@anaska.com>

I have php code I want to reverse engineer for this project

Help < Back Next > Cancel

Figure 28-28

Reverse engineering  
sous  
WaterProof::UML



Et voilà donc le résultat du reverse engineering pour PEAR::DB.

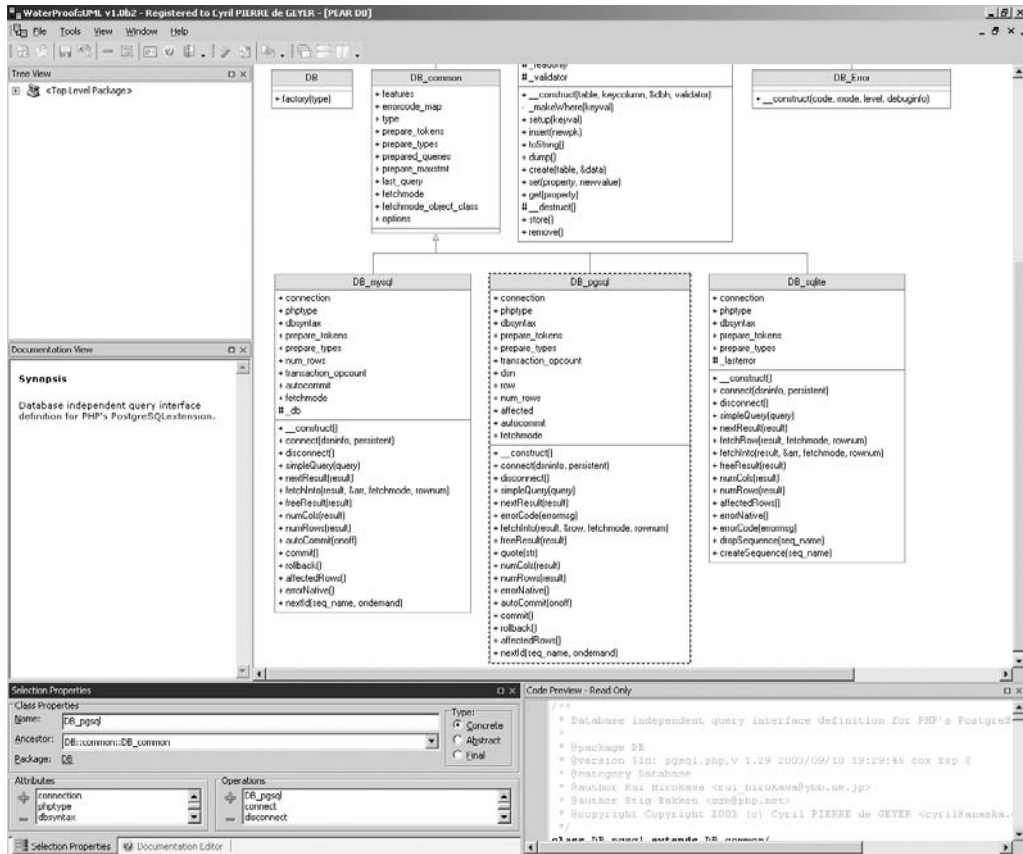


Figure 28-29

Schéma UML de PEAR::DB

Pour vous faciliter la prise en main d'UML et de l'outil, des assistants vous guident lors des étapes clés de votre modélisation. Cet outil de modélisation est de surcroît ouvert, il supporte un export en XMI (le format standardisé d'échange de modèles UML) et supporte les tags `phpDocumentor` ; lors de la génération de code PHP4, le code comporte les tags `@access`, `@static`, `@final...` et sont omis lors de la génération de code PHP5 car ces informations sont gérées par le langage lui-même dans un soucis de clarté et d'efficacité.

Voici par exemple le code généré pour une méthode finale, abstraite et protégée en utilisant le mode PHP 4 :

**Figure 28-30**  
Génération de code  
PHP 4

A screenshot of a code preview window titled "Code Preview - Read Only". The window displays PHP code for a method named "MaMéthode". The code includes a multi-line comment block with the text "Documentation de ma méthode", followed by tags `@final`, `@static`, and `@access protected`. The function definition is `function MaMéthode(){` followed by a single-line comment `// code for MaClasse.MaMéthode here ...` and a closing brace `}`.

```
Code Preview - Read Only
/**
 * Documentation de ma méthode
 *
 * @final
 * @static
 * @access protected
 */
function MaMéthode(){
    // code for MaClasse.MaMéthode here ...
}
```

Et le résultat en changeant simplement le mode pour PHP 5 :

**Figure 28-31**  
Génération de code  
PHP 5

A screenshot of a code preview window titled "Code Preview - Read Only". The window displays PHP code for a method named "MaMéthode". The code includes a multi-line comment block with the text "Documentation de ma méthode", followed by tags `@final`, `@static`, and `@access protected`. The function definition is `final protected static function MaMéthode(){` followed by a single-line comment `// code for MaClasse.MaMéthode here ...` and a closing brace `}`.

```
Code Preview - Read Only
/**
 * Documentation de ma méthode
 *
 * @final
 * @static
 * @access protected
 */
final protected static function MaMéthode(){
    // code for MaClasse.MaMéthode here ...
}
```

Pour résumer, WaterProof::UML est un outil pratique et simple d'utilisation qui vous permettra de gagner du temps lors des premières phases de vos projets et lors du cycle de vie pour toujours avoir une vision haut niveau de l'application pour vous permettre d'être plus efficace en vous concentrant sur les points où vous apportez de la valeur ajoutée.

Pour plus d'informations consultez le site : <http://www.waterproof-software.com/uml/>

## UML2PHP5

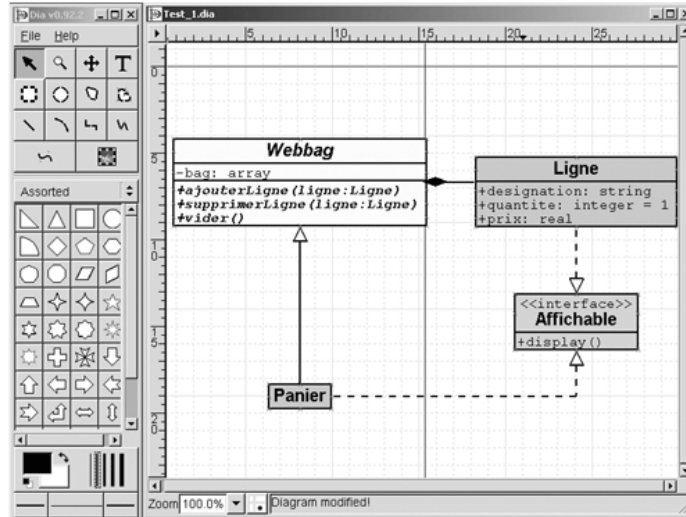
Une autre solution de modélisation UML est UML2PHP5. Celle-ci est particulièrement adaptée aux utilisateurs de DIA (logiciel de modélisation de diagrammes) et aux utilisateurs de Linux (DIA fonctionne sous Linux et Windows). UML2PHP5 permet de générer le code PHP 5 correspondant aux diagrammes UML dessinées.

Vous pouvez aller sur le site <http://uml2php5.zpmag.com/>

La licence de ces deux logiciels est la GPL, ce qui signifie que vous pouvez les utiliser librement. Seule la documentation de UML2PHP5 est payante, mais le tarif reste raisonnable (plus ou moins de 10 €). UML2PHP5 est écrit par un membre de la communauté PHP francophone : KDO.

Figure 28-32

*Dia sous Windows XP*



### Création d'un nouveau projet

La première étape consiste à créer vos classes et interfaces, liens d'héritage, etc. Vous pouvez aussi associer des commentaires aux classes, attributs, méthodes et paramètres de méthodes. Globalement vous modélisez votre application en UML avec DIA comme vous le feriez avec n'importe quel autre outil.

C'est une fois que vous avez terminé que vous allez utiliser un module (UML2PHP5) pour générer votre code PHP 5.

Il est à noter que le code généré contiendra automatiquement des balises de commentaires compatibles phpDoc : les balises @access, @var, @return, @param.

### Génération de code

La génération de code effectuée par UML2PHP5 est paramétrable par un fichier de configuration permettant de personnaliser le code généré. Ainsi, par exemple, on peut désactiver la génération automatique des balises et de commentaires pour phpDoc, choisir l'extension des fichiers de classe et d'interface, activer la génération automatique de *getters/setters* pour les attributs *private*, autoriser la création automatique des méthodes d'une implémentation etc.

Le plug-in sait prendre en compte les spécificités de PHP 5 comme la définition de constantes dans le corps de la classe ou le *typage objet* pour les paramètres de méthodes.

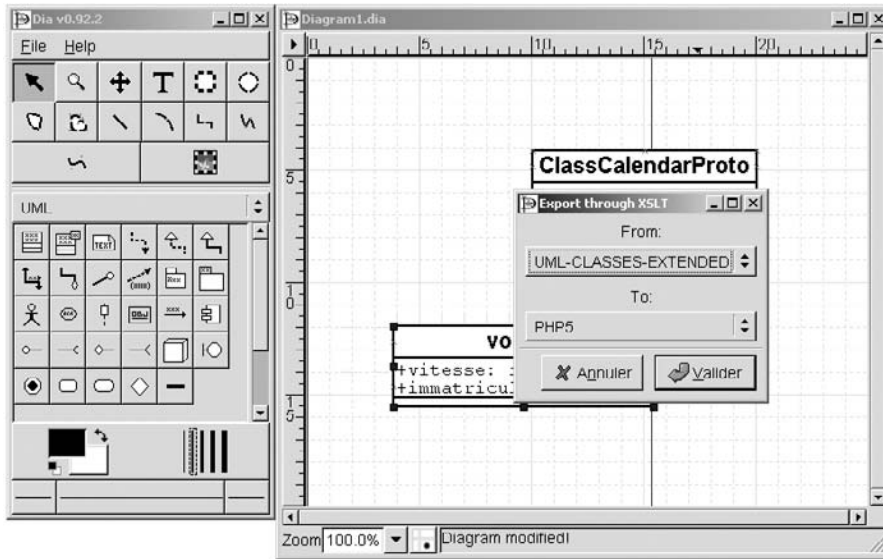


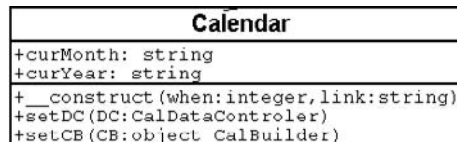
Figure 28-33

Export vers PHP 5 avec UML2PHP5

Voici un exemple de diagramme et le code généré par le plug-in.

Figure 28-34

Classe Calendar en  
UML



Les codes générés sont les suivants :

Fichier CalendarProto.interface.php

```
<?php
/**
 *
 * Code skeleton generated by dia-uml2php5 plugin
 * written by KDO kdo@zpmag.com
 */

interface CalendarProto {
    public function setDC(CalDataControler $DC);
    public function setBC(CalBuilder $BC);
}
?>
```



## Fichier Calendar.class.php

```
<?php
/**
 *
 * Code skeleton generated by dia-uml2php5 plugin
 * written by KDO kdo@zpmag.com
 */
require_once('CalendarProto.interface.php');

class Calendar implements CalendarProto {
    /**
     * current month (xx)
     * @var string
     * @access public
     */
    public $curMonth;

    /**
     * current year (xxxx)
     * @var string
     * @access public
     */
    public $curYear;

    /**
     * @access public
     * @param integer $when
     * @param string $link
     */
    public final function __construct($when, $link) {
    }

    /**
     * @access public
     * @param CalDataControler $DC
     */
    public final function setDC(CalDataControler $DC) {
    }

    /**
     * @access public
     * @param object CalBuilder $CB
     */
    public final function setCB(CalBuilder $CB) {
    }
}
?>
```

# Les frameworks

---

Un développement commence habituellement par la construction d'un cœur applicatif et d'une série de composants techniques. Ce cœur et ces composants serviront de base à toute l'application par la suite. On parle généralement de frameworks, littéralement des cadres de travail. Certains frameworks sont disponibles en Open Source. Les réutiliser vous permet de vous épargner un temps de développement important et de commencer immédiatement avec votre application au lieu de réinventer à chaque fois les mêmes briques techniques.

## Ce qu'est un framework

### *Un cadre de travail*

Les développeurs français utilisent couramment le terme anglais de framework. La traduction littérale donnée en introduction est toutefois très révélatrice du rôle de ces applications. Il s'agit bien de cadres de travail.

En utilisant un framework, vous vous imposez des conventions de nommage et d'organisation des fichiers. Vous adoptez aussi une série de bibliothèques pour gérer toutes les composantes techniques habituelles d'une application : abstraction de base de données, authentification, sessions. Enfin, vous adoptez des outils qui vous permettront d'utiliser tous ces composants. Vous pourrez réfléchir à ce qui fait votre application et sa spécificité. Tout le reste est déjà fait : le développement des briques techniques et les choix de structure.

Votre développement sera donc guidé, cadré. Vous aurez un environnement et des méthodes similaires à tous les autres développeurs qui utilisent le framework. Vous évitez aussi de vous poser des questions qui ont été résolues plusieurs fois par d'autres.

## ***La séparation MVC***

Le sigle MVC représente les trois termes suivants : Modèle, Vue et Contrôleur. Il s'agit d'une séparation de l'application en trois couches.

Le modèle se charge de faire l'interface avec la base de données et de représenter les données (généralement sous forme objet) et de définir les traitements propres à chaque métier. C'est là qu'on utilisera les abstractions de base de données par exemple.

La vue sert d'affichage. C'est cette partie qui génère le HTML, le PDF ou globalement le rendu envoyé au client. C'est ici qu'on utilisera XMLWriter et les systèmes de template.

Enfin, le contrôleur est le chef d'orchestre de l'application. C'est lui qui va faire l'interface avec l'utilisateur (recevoir et traiter les requêtes http), décider de l'action à entreprendre, déclencher des actions sur le modèle et envoyer des données dans la vue pour faire générer le HTML.

Quasiment tous les frameworks PHP modernes utilisent MVC ou une variation de cette structuration. Généralement la séparation entre les différentes couches applicatives est imposée par l'utilisation de plusieurs fichiers différents pour le modèle, la vue et le contrôleur. Le framework se charge ensuite d'ordonner le tout et d'appeler les fichiers quand cela est nécessaire.

## ***Les avantages d'un framework***

Les avantages d'un framework sont multiples. Le gain le plus visible au premier abord est un gain de temps. Le cœur de l'application est déjà développé. Vous vous épargnez au minimum plusieurs semaines d'étude, de choix de bibliothèques de code et de développement pour tout relier ensemble.

Ensuite, et surtout, vous assurez à votre application une sécurité et une fiabilité qu'elle n'aurait certainement pas eue sans cela. Quasiment tous les frameworks PHP sont développés publiquement et disponibles en Open Source. Le cœur de votre application a été enrichi et fiabilisé par des centaines ou des milliers d'installations, qui ont contribué en retours d'expérience ou en correctifs. Les bugs restants pourront être corrigés en installant des mises à jour publiques.

Enfin, vous bénéficiez d'une pérennité qu'il serait difficile d'égaliser. Le cœur de votre application et son fonctionnement interne sont documentés et soutenus par une communauté qui s'entraidera. Votre application ne sera pas dépendante d'un cœur applicatif privé, sous documenté et dont la connaissance risque de se perdre avec le départ des développeurs principaux de l'entreprise.

## Quelques frameworks disponibles en Open Source

Il existe des dizaines de frameworks PHP disponibles. Certains assez anciens, d'autres très récents. On peut en compter plus d'une douzaine qui ont une communauté active et qui sont relativement complets. Nous ne chercherons donc pas à tous les lister et nous nous attardons uniquement sur les plus en vue et les plus intéressants à notre avis.

Tous les frameworks présentés sont disponibles gratuitement et sous une licence libre.

### Copix et Jelix

Copix est un framework MVC qui propose une séparation de l'application en sept couches applicatives. Malgré cette forte structuration sur le papier, il s'en sort très bien. Il est même probablement l'un des frameworks les plus simples d'accès. Il ne propose toutefois que les composants les plus standards (abstraction de base de données, mapping objet-relationnel, sessions) et fait l'impasse sur les outils évolués pour générer du HTML (formulaires, Ajax), ainsi que sur la génération automatique de code.

Figure 29-1

Les logos  
de Copix et Jelix



Le développement de Copix est réalisé par une petite équipe française. Même si l'activité n'est pas très forte, une version 3 est proche de la sortie en juillet 2007. Sa pérennité est toutefois assurée car Copix est une des recommandations officielles des administrations françaises pour le choix d'un framework. Des ministères, des mairies et des conseils généraux utilisent donc Copix en interne. Vous trouverez des informations sur le site de Copix : <http://www.copix.org/>.

Jelix est un framework plus récent, réalisé par l'un des deux développeurs initiaux de Copix. Les concepts sont les mêmes mais le code a été réécrit entièrement pour proposer quelques fonctionnements spécifiques à PHP 5. Les développeurs sont très actifs et plusieurs modules supplémentaires complètent ce qui est disponible dans Copix (générateur de formulaires par exemple). Vous trouverez des informations sur le site de Jelix : <http://www.jelix.org/>.

## Symfony

Symfony est en 2007 le framework le plus en vue et le plus abouti pour PHP 5. Il propose une séparation MVC, des générateurs de code, un mapping objet relationnel, et une intégration d'Ajax.

Le gros avantage de ce framework est le peu de code à écrire. Les générateurs de code permettent d'obtenir des squelettes complets à modifier ou à utiliser et les fichiers de configuration sont rédigés dans une syntaxe très simple. La génération d'un premier back-office qui servira de base à l'application est une question d'heures tout au plus.

Le framework Symfony bénéficie d'une grande communauté et d'une forte activité. La documentation fournie est particulièrement complète. Une introduction au fonctionnement de Symfony est proposée à la fin de ce chapitre. Vos pouvez également consulter son site officiel : <http://www.symfony-project.com/>.

Figure 29-2

Le site officiel de Symfony

**symfony**    about    installation    documentation    community

symfony is an open-source PHP web framework

Based on the best practices of web development, thoroughly tried on several active websites, symfony aims to speed up the creation and maintenance of web applications, and to replace the repetitive coding tasks by power, control and pleasure.

Symfony provides a lot of features seamlessly integrated together, such as:

- simple templating and helpers
- cache management
- smart URLs
- scaffolding
- multilingualism and I18N support
- nhjert model and MVC separation
- Ajax support
- enterprise ready

Do you want to know more? Get started and join the growing symfony community now!

**discover**  
Read the [about page](#), [build your first project](#) or start browsing the [book](#).

**download**  
[Download and install](#) the .tgz archive of the latest stable release.

**interact**  
Discuss on our [mailing-lists](#), join us in the [#symfony](#) channel on the freenode IRC network or ask a question on our [forum](#).

easy ajax    admin generator    24 days with...    book

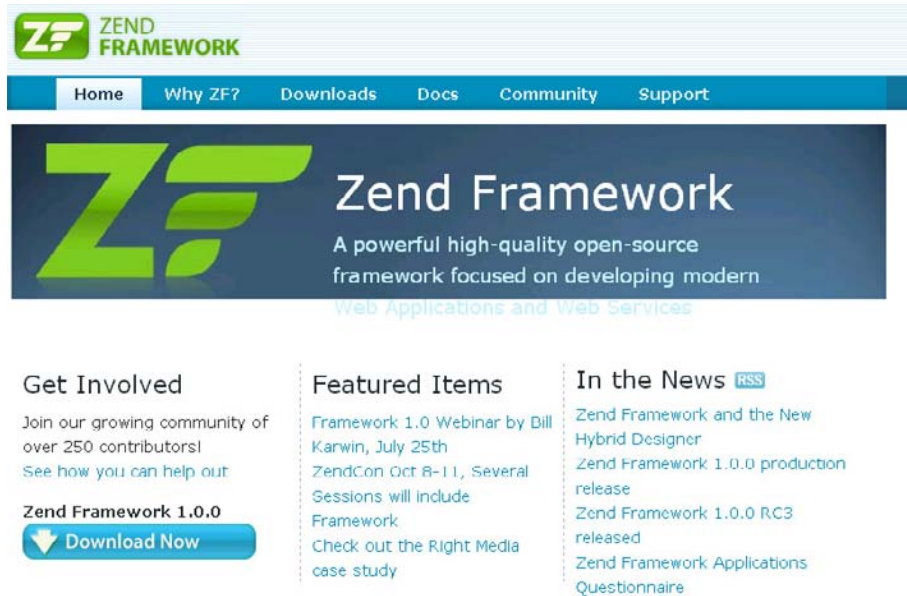
## Zend Framework

Le framework de Zend est le dernier arrivé. Malgré son nom, il s'agit davantage pour l'instant d'une série de bibliothèques de code (un peu comme PEAR) avec une proposition d'organisation qui constitue l'ébauche d'un framework.

Ces bibliothèques sont toutefois complètes et sous une licence très permissive. Le framework bénéficie aussi de la puissance de frappe de la société Zend et du support qu'elle peut offrir aux entreprises (moyennant finances). C'est d'ailleurs le gros point fort de cette solution.

Figure 29-3

Le site officiel du framework Zend



Les briques habituelles sont présentes. On trouve aussi des briques évoluées pour s'adresser au moteur de recherche Google ou générer du PDF. Malheureusement, des briques de base sont toujours absentes à ce jour comme la gestion des formulaires ou la génération automatique de code. Ce projet bénéficie néanmoins d'une forte activité et devrait s'enrichir rapidement. Allez faire un tour sur le site officiel de Zend (<http://framework.zend.com/>).

## Les autres

Nous manquons de place dans ce livre pour détailler suffisamment le fonctionnement de ces frameworks et vous en expliquer les différences principales. Toutefois, lors de votre choix, en plus des quelques uns déjà présentés, nous vous recommandons de considérer aussi les frameworks suivants :

- Prado : <http://www.pradosoft.com/>
- Code Igniter : <http://codeigniter.com/>
- CakePHP : <http://cakephp.org/>

## Courte introduction à Symfony

Afin de vous présenter le fonctionnement interne de Symfony et des frameworks MVC, nous vous présentons une application simple et minimale qui permet d'afficher une liste de petits textes avec un titre chacun.

### Installation

L'installation de Symfony peut se faire soit à partir d'une archive à décompacter, soit à partir de l'installateur de PEAR. Nous vous recommandons fortement cette dernière méthode.

En passant par l'installateur de PEAR, il nous faut d'abord informer PEAR de l'emplacement des paquet Symfony, puis ensuite en demander le téléchargement et l'installation. Toutes les dépendances sont alors téléchargées et installées :

```
pear channel-discover pear.symfony-project.com
pear install symfony/symfony
```

Vous pouvez tester l'installation en tapant la ligne de commande suivante :

```
symfony -V
```

Le numéro de la version de Symfony devrait apparaître en retour.

Pour le cadre de cet exemple, vous pouvez prendre directement une archive complète qui contient tout ce qui est nécessaire à Symfony et quelques fichiers d'exemples. Cette archive est disponible sur [http://www.symfony-project.com/get/sf\\_sandbox.tgz](http://www.symfony-project.com/get/sf_sandbox.tgz) et peut être décompactée directement dans le répertoire public de votre serveur web. Vous aurez par contre à exécuter toutes les instructions en ligne de commande directement depuis ce répertoire.

### Initialisation de l'application

Une fois les bibliothèques Symfony installées, il faut déployer un nouveau projet avec l'instruction suivante (myproject est le nom de votre projet) :

```
symfony init-project myproject
```

Dans le cadre de ce chapitre, en ayant pris le paquet sandbox, cette étape est déjà réalisée.

Il faut ensuite créer une ou plusieurs applications dans le projet. La commande est `init-app` (news est le nom de notre application exemple) :

```
symfony init-app news
```

#### Note

Sous Microsoft Windows, si votre exécutable PHP n'est pas trouvé, vous pouvez en changer l'adresse dans le fichier `symfony.bat`.

## Génération du modèle

Symfony génère les modèles et la base de données à partir d'une description dans un format nommé Yaml. Il s'agit d'un format texte hiérarchique plus simple que le XML.

Nous allons donc modifier la description de nos données dans le fichier `config/schema.yml`. Notre exemple étant très simple, nous n'aurons qu'une seule table avec quatre champs. Le code à insérer est le suivant (attention, les indentations sont faites avec deux espaces consécutifs, les tabulations sont à proscrire) :

```
propel:
  news_messages:
    _attributes: { phpName: Message }
    id:
    title:      varchar(255)
    body:      longvarchar
    created_at:
```

Ici nous avons déclaré une table `news_messages` qui contient nos messages. Chaque message a un identifiant unique, un titre, un contenu, et une date de création. La classe générée par Symfony s'appellera `Message`.

Nous allons ensuite demander à Symfony de générer notre classe de modèle (la classe qui gère les données et l'accès à la base pour nous), le SQL du schéma, et l'insertion de ce SQL dans la base de données :

```
symfony propel-build-model
symfony propel-build-sql
symfony propel-insert-sql
```

### Note

Dans notre sandbox, une base de données SQLite est déjà configurée et fonctionnelle. Si vous le souhaitez, vous pouvez changer cette base de données en éditant les fichiers `propel.ini` et `databases.yml` dans le répertoire de configuration.

Vous devriez avoir désormais deux classes respectivement dans les fichiers `lib/model/Message.php` et `lib/model/MessagePeer.php`.

## Premier contrôleur

Nous allons désormais pouvoir réaliser deux actions. La première consiste à réceptionner les demandes de création de nouveaux messages. La seconde sert à recevoir les requêtes sur la page d'accueil et déclencher l'affichage des dix derniers messages.

Commençons déjà par l'action qui crée les nouveaux messages. Il va nous falloir :

- réceptionner le titre et le corps du message à partir d'un formulaire ;
- demander un nouveau message ;



- remplir le contenu du message avec les paramètres récupérés ;
- sauvegarder le message ;
- définir un message de confirmation pour l'utilisateur ;
- enfin, renvoyer l'utilisateur vers la page principale qui liste les derniers ajouts.

Nous allons utiliser un fichier nommé `addAction.class.php` dans un nouveau répertoire `apps/news/modules/messages/actions` :

```
<?php
class addAction extends sfAction {
    public function execute() {
        // on récupère les données de formulaire
        $title = $this->getRequestParameter('title');
        $body = $this->getRequestParameter('body');
        // si les données sont bien présentes
        if (strlen($title) && strlen($body)) {
            // on crée un nouveau message
            $message = new Message();
            // on initialise son contenu
            $message->setTitle($title);
            $message->setBody($body);
            // on le sauvegarde dans son nouvel état
            $message->save() ;
            // on définit le message de confirmation
            // il sera affiché sur la prochaine page
            $this->setFlash('info', "Message bien enregistré");
        } else {
            // message d'erreur, les données sont incomplètes
            $this->setFlash('info', "Erreur, message non enregistré");
        }
        // on redirige vers la page d'index des messages
        return $this->redirect('messages');
    }
}
```

La seconde action se charge uniquement d'aller chercher les dix messages les plus récents et de les enregistrer pour qu'ils soient affichés plus tard par la vue. Ce fichier s'appellera `indexAction.class.php`, dans le même répertoire que le précédent :

```
<?php
class indexAction extends sfAction {
    public function execute() {
        // on définit une série d'options
        $search = new Criteria() ;
        // on liste par date, descendant
        $search->addDescendingOrderByColumn(MessagePeer::CREATED_AT);
    }
}
```

```
// 10 messages maximum
$search->setLimit(10) ;
// on exécute la requête et on enregistre
$this->messages = MessagePeer::doSelect( $search );
}
}
```

La sélection se fait en deux étapes. Tout d'abord on définit les options de la requête (nombre de résultats, tri). Ensuite, on exécute la requête à l'aide d'une classe Peer qui permet l'accès aux données.

Dans les deux contrôleurs, les classes Message et MessagePeer sont des classes automatiquement générées par Symfony lors de l'étape précédente. Le reste est fourni nativement par Symfony.

## Lien avec la vue

Nous avons établi la structure de nos données et les actions à réaliser sur ces données. Il ne nous reste plus qu'à développer l'interface HTML : la vue.

Nous n'avons qu'une seule vue, celle qui permet d'afficher les dix derniers messages. Le formulaire pour ajouter un nouveau message sera inséré à la fin de cette même page.

La composition générale de la page est déjà gérée dans un fichier nommé `layout.php`. Nous n'avons à écrire que le contenu principal lui-même. Notre fichier va s'appeler `indexSuccess.php` dans le nouveau répertoire `apps/news/modules/messages/templates` :

```
<h1>Ajouter un nouveau message</h1>
<?php echo form_tag('messages/add') ?>
<p>
  <label for="title">Titre:</label>
  <input type="text" name="title" id="title">
</p>
<p><label for="body">Contenu:</label></p>
<textarea name="body" id="body"></textarea>
<p><?php echo submit_tag('Insérer') ?></p>
</form>

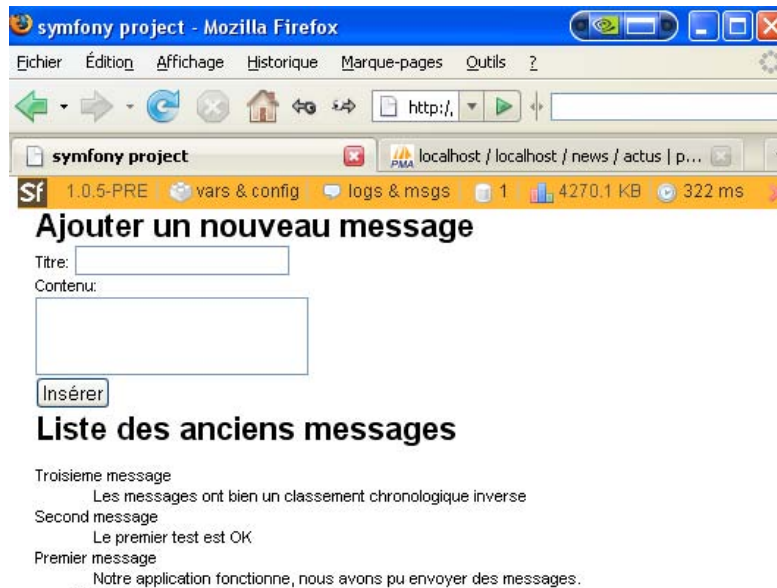
<h1>Liste des anciens messages</h2>
<d1>
<?php foreach($messages as $msg) { ?>
  <dt>
    <?php echo htmlspecialchars($msg->getTitle()); ?>
  </dt>
  <dd>
    <?php echo nl2br(htmlspecialchars($msg->getBody())); ?>
  </dd>
<?php } ?>
</d1>
```

## Le test

Toutes les pages passent par un script PHP central dans le répertoire `/web`. Le fichier à appeler pour notre mini-application s'appelle `news_dev.php` (où `news` est le nom de notre application). Ce script gère l'accès en mode développement. Vous en trouverez aussi un `news.php` pour l'environnement de production.

Avec ce script, nous souhaitons accéder au module `messages`. En considérant qu'on agit sur notre poste local et que la sandbox symfony a été installée directement dans le répertoire `web` public, l'adresse pour tester notre application est `http://localhost/web/news_dev.php/messages`.

Figure 29-4  
Test de notre  
application Symfony



## Quelques points non abordés

Ces quelques pages d'introduction à Symfony ne font qu'effleurer les possibilités. Avant de lire la documentation vous pouvez regarder le système de route, le *scaffolding* et le système métadonnées.

### Routes

Le système des routes fait la liaison entre l'URL qui est demandée par le navigateur et l'action effectivement exécutée. Les routes par défaut définissent des pages de type `/module/action/`, où l'action à exécuter est index par défaut. Dans notre cas, le module est `messages`. Ce fichier peut être lu dans le répertoire `apps/news/config` sous le nom `routing.yml`.

## Scaffolding

Le scaffolding est la possibilité pour Symfony de générer automatiquement le code nécessaire pour faire des créations, recherches, mises à jour et suppressions sur un modèle. On parle fréquemment de CRUD (Create, Read, Update, Delete). Vous aurez alors des écrans opérationnels pour les fonctionnalités de base, il vous suffira de les modifier. On peut générer un CRUD automatique avec la commande suivante :

```
■ symfony propel-generate-crud news messages Message
```

Ici news est le nom de l'application, messages le nom du module et Message le nom du modèle.

## Métadonnées

Enfin, si vous regardez le fichier `layout.php` dans `apps/news/templates`, vous verrez qu'il ne contient pas certaines informations comme le titre des pages ou les informations liées aux moteurs de recherche. Ces métadonnées sont stockées en configuration. Elles peuvent être définies de manière générale pour toutes les pages, ou de manière précise page à page. Vous pouvez modifier ces informations dans le fichier `apps/news/config/view.yml`.

## Documentation

Cette section sur Symfony n'a pour but que de vous mettre l'eau à la bouche et de vous montrer les fonctionnements mis en œuvre dans les frameworks MVC.

Nous vous encourageons donc fortement à aller lire et mettre en application le premier tutoriel à l'adresse [http://www.symfony-project.com/tutorial/my\\_first\\_project.html](http://www.symfony-project.com/tutorial/my_first_project.html), ainsi que le reste de la documentation et des vidéos sur <http://www.symfony-project.com/content/documentation.html>.



# Annexe

## Ressources en ligne

Une des grandes forces de PHP réside dans la multitude et la diversité des applications existantes. La difficulté pour l'utilisateur non averti va donc consister à trouver dans cette profusion ce dont il a besoin. Nous vous proposons ici deux types d'outils : des bibliothèques et des frameworks, ainsi que des logiciels complets développés en PHP. Cette liste n'a pas pour but d'être exhaustive mais de vous orienter dans vos recherches.

### *Bibliothèques*

#### Images

Il existe plusieurs bibliothèques et classes objet qui implémentent une interface simple pour fabriquer des graphiques. Il n'est donc pas nécessaire de faire appel aux fonctions bas niveau de la bibliothèque GD.

#### JpGraph

Une de ces bibliothèques est la JpGraph que nous avons présenté au chapitre 24 et qui concerne le traitement des images. C'est la référence en matière de génération d'images. De plus, elle est distribuée sous licence libre QPL (*QT public license*) mais une licence commerciale est aussi disponible.

URL : <http://www.aditus.nu/jpgraph/>

#### Artichow

Artichow est une bibliothèque permettant de créer simplement des graphiques avec PHP 5 et GD. Elle permet notamment de générer des courbes, des histogrammes, des camemberts, etc.

Artichow constitue une alternative au projet JPGraph, dont la licence QPL est restrictive. Artichow appartient dans le domaine public. Chacun est donc libre de la copier, modifier, publier ou distribuer, que ce soit pour une utilisation commerciale ou non.

URL : <http://www.artichow.org/>

## PHPPlot

PHPPlot est un très bon outil de génération de graphiques. Cette bibliothèque est disponible sous la même licence que PHP, ce qui lui permet d'être intégrée dans un applicatif propriétaire sans aucune contrainte de redistribution.

URL : <http://www.phplot.com>

## E-mails

La gestion des e-mails standards est très simple avec PHP. Cependant, si vous souhaitez une gestion plus poussée des e-mails, vous risquez de rencontrer un certain nombre de difficultés. Il existe de nombreuses bibliothèques qui permettent de gérer facilement les envois d'e-mails au format HTML, avec des pièces jointes, etc.

### Zend Mail

Le framework Zend contient un composant de gestion d'e-mails assez évolué qui peut être utilisé de manière indépendante. Il gère les envois par SMTP, les authentifications, les pièces jointes et le HTML.

URL : <http://framework.zend.com/manual/en/zend.mail.html>

### PEAR::MAIL

Le dépôt PEAR contient lui aussi un composant de gestion d'e-mails. La classe PEAR:MAIL gère l'envoi et la composition d'e-mails simples. Il faudra la combiner avec la classe Mail\_Mime de PEAR pour utiliser des pièces jointes.

URL : <http://pear.php.net/package/Mail>

### IMP

Contrairement aux deux précédentes, `IMP` n'est pas une bibliothèque permettant d'envoyer des e-mails mais un webmail complet en PHP pouvant rapatrier vos messages par POP3 ou IMAP4. C'est probablement le webmail le plus installé si on ne tient pas compte de ceux dédiés à un serveur mail donné.

URL : <http://www.horde.org/imp/>

## Formulaires

Plusieurs outils existent pour vous aider dans la gestion de vos formulaires.

### HTML\_QuickForm

Le package PEAR HTML\_quickform, dont le principal contributeur Bertrand Mansion est Français, permet de réaliser des formulaires complexes sans se soucier du code HTML.

Ce package offre un certain nombre de fonctionnalités dont la validation et le filtrage des saisies.

URL : [http://pear.php.net/package/HTML\\_QuickForm](http://pear.php.net/package/HTML_QuickForm)

## Benchmarks

Pour optimiser une application, il est nécessaire de pouvoir connaître les temps d'exécution des différentes parties de vos scripts. Pour du *profiling* plus pointu, orientez-vous vers xdebug.

### PEAR Benchmark

PEAR Benchmark est extrêmement simple et vous permettra de réaliser des benchmarks simples de vos scripts.

URL : <http://pear.php.net/package/Benchmark>

## Abstraction de bases de données

Que ce soit pour faciliter le changement de SGBD ou pour unifier les appels à des SGBD différents, on peut utiliser ce que l'on appelle l'abstraction de base de données. Cependant, il faut faire attention aux requêtes SQL que vous faites car il est souvent nécessaire de se limiter au jeu SQL commun aux SGBD envisagés.

### AdoDB

AdoDB est une couche d'abstraction de base de données avancée. Une méthode de classe permet de s'affranchir des différents dialectes et la requête sera syntaxiquement correcte avec MySQL, PostgreSQL ou SQL server (notamment la clause LIMIT spécifique à MySQL et pourtant si pratique pour la pagination). D'autres méthodes permettent de faciliter l'affichage. AdoDB est l'une des couches d'abstraction pour SGBD les plus performantes en PHP.

URL : <http://adodb.sourceforge.net/>

### PEAR:MDB2

PEAR:MDB2 est l'évolution des composants PEAR DB et PEAR Metabase. Le développement de ces deux derniers s'est donc arrêté. Vous y trouverez une abstraction de base de données complète, avec un support de quasiment toutes les bases du marché.

URL : <http://pear.php.net/>

## Templates

Les templates ont été présentés au chapitre 22. Ils servent à dissocier la logique métier de la logique d'affichage. Pour plus de détails ou un choix plus complet de bibliothèques, consultez ce chapitre.

### Smarty

Smarty est le moteur de templates le plus répandu dans le milieu PHP. Il fait un peu office de couteau suisse puisqu'il offre une solution adaptée à la majorité des cas et permet d'ajouter simplement des modules pour gérer le reste. Il dispose d'un système permettant de pré-interpréter les templates afin d'éviter une trop forte charge lors de l'exécution.

URL : <http://smarty.php.net/>



### PHPLib

La PHPLib est la bibliothèque la plus ancienne. Elle bénéficie donc d'une maturité importante. On y trouve une syntaxe objet qui allie simplicité et performances. Si vos besoins sont peu complexes, c'est probablement la solution qui prendra le moins de ressources et sera la plus agréable à maintenir.

URL : <http://phplib.sourceforge.net>

### PDF

Plusieurs bibliothèques existent pour manipuler des fichiers PDF.

#### FPDF

Développée par un Français (Olivier Plathey), cette bibliothèque est l'une des solutions les plus performantes d'édition de fichiers PDF. Son modèle objet permet la création facile d'extensions. Un certain nombre est d'ailleurs disponible sur le site de la bibliothèque.

URL : <http://www.fpdf.org757>

#### eZPDF

Une autre bibliothèque permettant de générer des fichiers PDF.

URL : <http://www.ros.co.nz/pdf/>

### Caches

Il existe plusieurs types de caches : le cache niveau opcode et le cache PHP. Pour le premier nous vous conseillons de regarder du côté de eaccelerator (ex turkmmcache) ou du côté des outils payants de Zend. Les solutions et outils au niveau PHP sont présentés plus en détail au chapitre 23.

#### JPCache

La bibliothèque JPCache est orientée vers les performances. Sa démarche est de donner une gestion complète des capacités de cache tout en nécessitant peu d'appels à des fonctions spécifiques.

URL : <http://www.jpccache.com/>

#### PearCache

Pear::Cache est une solution très générique, faite pour être personnalisable selon vos besoins. Il s'agit d'une bibliothèque relativement bas niveau, facilement extensible et spécialisable. Le but est d'avoir une batterie de classes dérivées pour des applications spécifiques (cache de la page résultat, d'une requête SQL, d'une image, etc.) qui se basent sur la classe générique.

Un bibliothèque dérivée, Pear::Cache\_Lite permet d'implémenter un cache de page simple avec de meilleures performances. Vous n'y trouverez cependant pas toutes les possibilités fonctionnelles de Pear::Cache.

URL : <http://pear.php.net/package/Cache>

## Documentation

Il existe des outils pour faciliter la création de la documentation technique de vos applications.

À partir de fichiers PHP correctement commentés suivant une norme établie vous pourrez générer des documentations complètes.

### PHPDocumentor

Un outil indispensable sur des projets d'envergure. PHPDocumentor et sa syntaxe sont utilisés sur la majorité des codes Open Source en PHP. On le retrouve par exemple pour la documentation technique de toutes les classes du dépôt Pear.

URL : <http://www.phpdoc.org/>

## Applications PHP

La force de PHP ne réside pas uniquement dans la multitude de bibliothèques et de classes disponibles. Il existe aussi de nombreux outils prêts à l'emploi qu'il vous suffit d'installer et éventuellement d'adapter à vos besoins.

### Blogs

Les blogs sont des logiciels qui permettent l'écriture d'articles courts. On peut s'en servir comme d'un carnet de voyage en ligne pour permettre à ses proches de suivre ses péripéties.

### DotClear

Un weblog simple et pratique. Il bénéficie d'un support Unicode, d'un jeu important de traductions de l'interface utilisateur et d'un nombre important de plug-ins (pour gérer de la galerie de photos jusqu'au forum).

URL : <http://www.dotclear.net/>

### Wordpress

Une référence dans le monde des blogs. Ses capacités sont très proches de Dotclear. Il est un peu moins courant en France mais est majoritaire dans les autres pays.

URL : <http://wordpress.org/>

### Forums

Les forums sont des espaces d'échanges dont le niveau de fonctionnalité peut être élevé (gestion de droits, système de push, affichage en arborescence, recherche, etc.). Il existe un nombre important de logiciels permettant d'offrir de tels services aussi avons-nous choisi de ne vous proposer que les deux principaux.

## PHPBB

PHPBB est le logiciel référence en matière de forum. Comme des dizaines de milliers de sites l'utilisent, il dispose d'une communauté très forte et active. Ses fonctionnalités sont poussées mais son administration autant que son utilisation sont simples. Attention cependant à se tenir au courant des évolutions de version car sa popularité en fait un outil dont les failles sont très recherchées.

URL : <http://www.phpbb.com/>

## Phorum

Autre dinosaure de la gestion de forum : Phorum. Ce logiciel est plus sobre que PHPBB mais offre de meilleures performances.

URL : <http://www.phorum.org/>

## PunBB

Arrivé plus récemment que les deux premiers, PunBB met en avant la sécurité et la légèreté du code plutôt que la multiplication des fonctionnalités.

URL : <http://www.punbb.fr/>

## Gestion de contenu

### SPIP

Un logiciel simple d'utilisation qui se prête particulièrement bien aux sites à orientation éditoriale. Développer un site de contenu simple sous SPIP est très rapide. Les possibilités de personnalisation ou de workflow sont toutefois limitées. Si les fonctionnalités vous suffisent, c'est probablement le premier logiciel à essayer.

URL : <http://www.spip.net/>

### Typo3

Ce CMS (*Content management system*) Open Source allemand jouit d'une excellente réputation dans le milieu professionnel. Ses possibilités sont plus importantes que celles de SPIP.

URL : <http://www.typo3.com/>

### ezPublish

Il s'agit probablement du CMS actuellement le plus complet en PHP. Le code est distribué sous licence Open Source mais il provient d'une entreprise norvégienne qui peut, sur demande, assurer le support ou des missions d'expertise.

URL : <http://www.ez.no/>

## Mambo

Un bon portail offrant de nombreuses possibilités de configuration. Mambo est distribué suivant la licence GPL (*GNU Public Licence*). Vous ne pourrez donc pas le redistribuer sous forme propriétaire.

URL : <http://www.mamboserver.com/>

## Xoops

Xoops est un logiciel de gestion de contenu dynamique. Son système d'extension permet de choisir les modules que l'on souhaite activer sur le site.

URL : <http://www.xoops.org/>

## Travail collaboratif

Dans une entreprise, le rôle d'un logiciel de travail collaboratif peut être comparé à celui d'un portail web de l'intranet. Il centralise plusieurs applications clientes telles que FTP, messagerie, agendas, bloc-notes, plannings, suivi de projets, etc.

## PHPGroupWare

phpGroupWare est composé de plusieurs modules dont un système de gestion de comptes et de configuration, un webmail, un calendrier partagé, un carnet d'adresses partagé, un gestionnaire de tâches, etc.

URL : <http://www.phpgroupware.org/>

## MoreGroupWare

Moregroupware est un logiciel de travail collaboratif en ligne (intranet/Internet). Il permet de gérer ses contacts et son agenda, mais également de consulter ses e-mails, de voir les nouvelles provenant de plusieurs sites, etc.

Sa vocation principale reste le travail collaboratif. Il vous permettra de gérer une équipe, des projets (gestion de temps, bug-tracker, etc.), etc.

URL : <http://www.phpdoc.org/>

## ERP

### Dolibarr

Dolibarr est un système libre de gestion d'entreprise. Outre les indispensables fonctionnalités de devis/facturation, il permet de générer des rapports, de gérer ses propositions commerciales, d'éditer des factures au format PDF, de gérer la relation client, etc.

URL : <http://www.dolibarr.com/>

## CRM

Le CRM (*Customer Relationship Management*, ou en français GRC, gestion de la relation client) vise à proposer des solutions technologiques permettant de renforcer la communication entre l'entreprise et ses clients afin d'améliorer la relation grâce à l'automatisation de certaines tâches.

### SugarCRM

SugarCRM est une solution de CRM global, intégrant notamment les modules de Gestion des forces de vente (SFA), d'Automatisation du marketing (EMA) et de Gestion du service client.

URL : <http://www.sugarcrm.com/>

### OBM

OBM est une application permettant, entre autres, la gestion des rendez-vous, des réunions, des contacts, de la comptabilité, des incidents, etc.

URL : [http://www.aliacom.fr/solutions/solution\\_soft/obm](http://www.aliacom.fr/solutions/solution_soft/obm)

## Boutiques en ligne

L'une des demandes les plus fortes sur Internet concerne la vente en ligne. Si votre projet n'est pas tentaculaire, laissez-vous tenter par les outils existants.

### OsCommerce

OsCommerce est un logiciel permettant de gérer tous les aspects d'une boutique en ligne. Il est particulièrement adapté à la vente de matériel.

URL : <http://www.oscommerce.com/>

### ZenCart

Un bon logiciel de e-commerce.

URL : <http://www.zencart.com/>

## E-learning

### Ganesha

Ganesha est une plate-forme de téléformation (*Learning Management System* ou LMS). Ce logiciel permet à un formateur ou un service de formation de mettre à la disposition d'un ou plusieurs groupes de stagiaires, un ou plusieurs modules de formation avec supports de cours, compléments, quiz et tests d'évaluation ainsi que des outils collaboratifs (webmail, forum, chat, partage de documents) et d'assurer un tutorat en ligne.

URL : <http://www.anemalab.org/ganesha/>

## Claroline

Claroline est un logiciel Open Source offrant un environnement de travail aux professeurs et élèves pour créer et gérer des cours via Internet. On dispose des principales fonctionnalités de ce type d'outil : calendrier partagé, *chat*, forums, rédaction de cours en ligne, création d'exercices, etc.

URL : <http://www.claroline.net/>

## Galerie d'images

### Gallery menalto

Gallery menalto est un logiciel de gestion de photos très complet.

URL : <http://gallery.menalto.com>

## Gestionnaire de bannières publicitaires

### PHPAdsnew

PHPAdsnew est un logiciel complet qui vous permet de gérer vos campagnes publicitaires d'affichage de bannières. Ses fonctionnalités sont très poussées et vous permettent notamment de créer des comptes à vos clients pour qu'ils gèrent eux-mêmes leurs bannières.

L'administrateur peut définir pour chaque utilisateur ou campagne des attributs basés sur des critères temporels ou géographiques (géolocalisation).

URL : <http://phpadsnew.com/>

## Moteurs de recherche

### PHPDig

PHPDig est un logiciel qui vous permet d'ajouter très facilement un moteur de recherche à votre site. Son utilisation se fait en deux étapes. La première consiste en l'indexation des pages sur lesquelles vous souhaitez pouvoir effectuer des recherches. Il ne reste ensuite plus qu'à insérer votre moteur de recherche sur votre site. Les résultats sont classés par pertinence.

URL : <http://www.phpdig.net/>

### Mnogosearch

Un outil indispensable sur des projets d'envergure. Il a l'avantage de bénéficier d'un module PHP pour gérer ses fonctions à partir de notre langage.

URL : <http://search.mnogo.ru/>



# Index

---

## Symboles

.Net 595

\$\_COOKIE[] 218

## A

accès

concurrents 318

statiques 283

accessor 288

adresse IP 206, 208, 348

affectation

*Voir* opérateurs 79

affichage 107

avec des masques 108

Afup (Association française des utilisateurs de PHP) 16

aléatoire 149

alerte de sécurité 510

Apache

configuration 33, 36

installation

sous Unix 39

sous Windows 32

sécurité 710

versions 30, 32

apostrophes 69

architecture 11

array

*Voir* tableaux 71

arrondi 148

ASCII 111

assertion 499

avoir un message explicatif 501

configuration 500

désactivation 500

description 499

erreur pendant l'évaluation 502

gestion personnalisée 502

utilisation 499

authentification 161, 164, 247

HTTP 203

autoincrément 463

autoload 285

## B

balises

d'ouverture 46

d'ouverture et de fermeture 54

HTML, suppression 118

base de données

*Voir* SGBD (Système de gestion de bases de données) 4

bases numériques 150

booléens 65

boucles 93, 95, 96

instructions d'arrêt 98

byte code 2, 13

## C

cache 239, 590

CakePHP 767

caractère 111

casse

*Voir* chaîne de caractères 124

CGI (Common Gateway Interface)

28, 33

sécurité 709

chaîne de caractères

accéder à un caractère précis 111

casse 124

échappement 712, 722

élagage 123

lister les mots 112

manipulations 122

position d'une sous-chaîne 113

protections et échappements

*Voir* protections et échappements 114

recherche 687, 697

d'une sous-chaîne 122

remplacement 699

remplacer un motif 122

remplissage 124

taille 111

test de présence 114

*Voir* expression régulière 685

chaînes de caractères 66

chargement automatique 285

chemin de recherche 49, 311

chiffrement 157

sécurité 729

classe 254

abstraite 280

accès privé 277

accès protégé 277

accès public 276

accès statique 283

chargement automatique 285

constructeur 269

déclaration 255

destructeur 271

finale 282

héritage 272

instanciation 257

méthodes parentes 275

parente 284

redéfinition 274

CLI (Command Line Interface) 3,

28, 31, 43, 58, 213

client-serveur 199



- clonage
    - Voir* objet 266
  - codage caractères 596
    - Voir* jeu de caractères 119
  - Code Igniter 767
  - code source 6
  - coloration syntaxique de code 147
  - commande shell 338
  - commentaires 54
  - comparaison
    - Voir* opérateurs 84
  - compatibilité 271
    - PHP 4 28, 266
    - PHP 5 490
  - compression
    - des pages 47
      - Web 376, 380
    - flux 352
  - concaténation
    - Voir* opérateurs 84
  - condition 88, 92
    - type d'objet 262
  - configuration 45
    - affichage 143
    - e-mail 388
    - fichier php.ini 30
    - modification 50
      - via* Apache 52
    - sécurité 708
    - sessions 238
  - constante 63, 256, 260
  - constructeur 269
  - contrôle de type 279
  - contrôleur 764, 769
  - conversions entre jeux de caractères 381
  - cookie 215, 217, 240, 241, 376
    - de session 234, 237, 239
    - envoi 217
    - expiration 221, 237
    - lecture 218
    - modification 220
    - restriction de portée 223
    - sécurité 225
    - suppression 220
  - copie 264
    - de fichier 320
  - Copix 765
  - coût 6
  - CRC32 158
  - cross site scripting 722
  - crypt 161
  - CSV (Comma Separated Values) 307
  - CVS (Current Version System) 9
- ## D
- date 151
  - date d'expiration 239
  - débogage 146, 259
  - débogueur 742, 745, 748
  - délai d'affichage 384
  - destructeur 271
  - DNS (Domain Name Server) 340
  - documentation 10, 23
  - DOM (Document Object Model) 549
    - chargement des données 551
    - copie d'un nœud 564
    - création d'un document 551, 566
    - création de nœud 562
    - description d'un nœud 553
    - effacer un nœud 565
    - erreurs 550
    - extensions 568
    - gestion des attributs 560
    - import SimpleXML 552
    - insertion d'un nœud 564
    - jeu de caractères 551
    - modification de l'arbre 564
    - navigation dans l'arbre 555
    - objet document 551
    - sauvegarde XML 553
    - structure générale 550
    - type de nœud 553
    - Xinclude 569
    - Xpath 567
  - double
    - Voir* nombres à virgule flottante 66
  - doublons dans un tableau
    - Voir* tableaux 141
  - Dreamweaver 749
  - droits d'accès 44, 47, 330, 716
- ## E
- échappement 67, 463
  - éditeur 735
  - égalité de type 85
  - e-mail 385
    - adresse de retour 391
    - anatomie 389
  - codage de transport 394
  - configuration 388
  - délimiteur MIME 397
  - destinataire en copie 392
  - entête 405
  - envoi en masse 408
  - envoyer 390, 409
  - expéditeur 391
  - HTML 395, 400, 409
  - image 400, 411
  - images 395
  - IMAP 388, 403
  - jeu de caractères 394, 411
  - lire 405
  - pièce jointe 397
  - plusieurs destinataires 390
  - POP 3 402
  - priorité 392
  - recevoir 402, 404
  - RFC 390
  - sécurité 406
  - spécifications 390
  - type de contenu 393
  - type MIME 393, 395
  - vérification 407
  - en-tête 597
    - HTTP 200, 216, 218, 233, 376
  - entités HTML et XML 116, 168
  - entrées utilisateur 717
  - erreur 56, 483, 598
    - affichage 48, 487, 492, 493
    - assertion
      - Voir* assertion 499
    - avertissement e-mail 494
    - compatibilité PHP 5 490
    - configuration 487
      - de production 488
      - pendant le développement 487
    - désactiver 491
    - description 485
    - enregistrer 485
    - exception
      - Voir* exception 503
    - fatale 499
    - filtrer 489
    - flux
      - Voir* flux 341
    - gestion 509
    - ignorer 487, 491
    - interception 484
    - journalisation 48, 487, 493

- lien vers la documentation 492
  - niveaux 48, 60, 488
  - réaction 486
  - sécurité 728
  - sous forme d'exception 508
  - erreur gestion personnalisée 498
  - Erreurs 452
  - espace disque 328
  - exception 503, 599
    - création 503
    - description 503
    - filtrage 505
    - gestionnaire 507
    - lancement 504
    - personnalisation 504
    - propagation 506
    - réception 504
    - renvoi 507
    - utilisation 508
  - exécution 2, 57, 164
    - à la réception d'un e-mail 406
    - de commandes 724
    - de programme 339
      - Voir* programme 337
    - en programme autonome 58
    - paramètres 347
    - statut des programmes 345
    - valeur de retour 341
  - expression régulière 685
    - assertion 693
    - capture 695
      - gloutonne 692
    - caractères
      - autorisés 689
      - interdits 689
    - classe de caractères 690
    - construction 688
    - début et fin de la chaîne 693
    - délimitation 686
      - des lignes 694
    - échappement 686, 702
    - fonction de rappel 697, 702
    - fonctionnement 703
    - modificateur 696
    - parenthèse
      - capturante 695
      - non capturante 695
    - performance 703
    - protection 686, 702
    - recherche 697
    - remplacement 699
    - répétitions 691
    - syntaxe 685
  - EXSLT 572
    - Voir* XSLT 572
  - eXtensible Markup Language
    - Voir* XML (eXtensible Markup Language) 237
  - extension
    - de fichier 716
    - Voir* module 45
- ## F
- fichier 303
    - accès concurrents 318
    - adresse 311, 329
    - changement de propriétaire 332
    - copie 320, 359
    - création 321
    - date de création 327
    - date de dernier accès 327
    - date de modification 327, 634
    - de configuration 305
    - déplacement 320
    - distant 312, 353
    - écriture 303, 310, 314
    - effacement 321
    - effacer 353
    - extension 716
    - fermeture 317
    - fin 316
    - fonctions d'accès rapide 304
    - lecture 303, 304, 313
    - local 352
    - mode d'ouverture 312
    - nom 329
    - ouverture 310
    - permissions 330, 332
    - pointeur 315
    - sécurité 333
    - taille 314, 328
    - tampon 317
    - temporaire 322
    - test d'existence 327
  - files 141
  - filter 184, 717
  - filtrage 184
  - filtre
    - de flux
      - Voir* flux 357
    - de sortie 380
    - de sortie utilisateur 382
  - filtrer les erreurs
    - Voir* erreur 489
  - fixation de session 246, 714, 729
  - float
    - Voir* nombres à virgule flottante 66
  - flux 337, 351
    - abstractions 352
    - compressé 352
    - connexion sécurisée 354
    - contexte 359
    - données non lues 357
    - entrée 341
    - entrée et sortie de PHP 354
    - erreur 341
    - expiration 356
    - fermeture 358
    - fichier distant 353
    - filtre 357, 361
      - personnalisé 363
    - fin de réception 356
    - lecture et écriture 356
    - métadonnée 352, 356
    - mode bloquant 358
    - ouverture 355
    - réseau 354, 358
    - sortie 341, 375
    - tampon 356
    - temps d'expiration 357
    - type 352, 355, 364
  - fonctions 99
    - appel 100
    - déclaration 99
    - nombre de paramètres indéfinis 104
    - retourner plusieurs valeurs 103
  - formulaires HTML 167
    - bouton radio 175
    - cases à cocher 174
    - champ de texte 171
    - définition 169
    - gestion du temps 195
    - image cliquable 192
    - liste déroulante 176
    - mage cliquable 179
    - méthode d'envoi 170
    - sécurité 196
    - sélections multiples 178, 191
    - taille des données 196
    - upload de fichier 179
    - zone de texte 173
  - frames 237

frameworks 763  
 FTP (File Transfer Protocol) 353  
 fusionner des tableaux  
   *Voir* tableaux 138

**G**

gabarits 601  
 garbage collector  
   *Voir* ramasse-miettes 239  
 GD 650  
 gestion  
   des erreurs 484  
   documentaire simple 334  
 GIF 651  
 graphique 673  
   en camembert 676  
 GTK 3

**H**

hachage 158, 730  
 hébergement 21  
 heredoc 69  
 héritage 272  
 historique 6  
 HTML 116, 168, 182, 184  
 HTTP (Hypertext Markup Language) 204, 579  
 HTTP (HyperText Transfer Protocol) 199, 208, 215, 239, 353, 598  
   cache  
     *Voir* système de cache 633  
   date de modification 634  
   définir la durée de vie d'une page 637  
   en-tête 654  
   proxy 635

**I**

iconv 120  
 IDE (Integrated Development Environment) 735  
 identification par adresse IP 207  
 image 649  
   affichage 654  
   arc de cercle 657  
   coordonnées 656  
   copie d'une zone 663  
   création 650  
   écrire du texte 659, 661  
   enregistrement 655  
   fusion 663

graphique 673  
 libération de la mémoire 653  
 miniature 680  
 palette 664  
 polices de caractères 660  
 rectangle 658  
 redimensionnement 663  
 redimensionner 680  
 référentiel 656  
 rotation 664  
 superposition 683  
 taille 665  
 tracer des formes 656  
 transparence 664, 665  
 utilisation des couleurs 653, 664  
 utiliser une image existante 652

**IMAP**

*Voir* e-mail 388  
 include\_path 311  
 inclusion 105  
 incrémentation  
   *Voir* opérateurs 83  
 injection SQL 721  
 installation  
   Apache  
     *Voir* Apache, installation 32  
   CGI (Common Gateway Interface) 44  
   module Apache 43  
   MySQL  
     *Voir* MySQL 36  
   PHP sous Unix 38, 41  
   PHP sous Windows 29

**integer**

*Voir* nombres entiers 65  
 interface 280, 281  
 interopérabilité 5  
 IRC (Internet Relay Chat) 18  
 itérateur 288

**J**

JavaScript 213, 726  
 Jelix 765  
 jeu de caractères 119, 376, 381, 394, 522, 542, 551  
 journal système 494, 495  
 JpGraph 668

**L**

LAMP (Linux Apache MySQL PHP) 27  
 licence 2, 6  
 liens 322  
   symboliques 330  
 ligne de commande 31  
   *Voir* CLI (Command Line Interface) 3  
 log 493

**M**

machine virtuelle 13  
 magazine 22  
 magic\_quotes 48, 182, 464, 722  
 magic\_quotes\_gpc 712  
 majuscule  
   *Voir* chaîne de caractères, casse 124  
 maximum 148  
 mbstring 120  
 MD5 159  
 mémoire, allocation maximale 48  
 migration 27  
 MIME  
   *Voir* e-mail 393  
 minimum 148  
 minuscule  
   *Voir* chaîne de caractères, casse 124  
 mode d'ouverture 312  
 modèle 764, 769  
 module 45  
   activation 46  
   Apache 29, 43  
   répertoire 45  
 modulo  
   *Voir* opérateurs 82  
 mot de passe 730  
 mutator 287  
 MVC (Modèle, Vue, Contrôleur) 764  
 MySQL 37, 38, 446  
   auto-incrément 430, 434  
   chaîne de caractères 428  
   clé primaire 429  
   comparatif 415  
   connexion persistante 451  
   coût 416  
   créer une base 424  
   créer une table 425

- dates et heures 427
  - fonctionnalités 417
  - installation
    - sous Unix 40
    - sous Windows 36
  - interface d'administration 422
  - nombres 426, 427
  - performances 415
  - présentation 415
  - types de champs 426
  - types de tables 418
  - utilisation avec Dreamweaver 751
  - mysql 37
- N**
- navigateur 211
  - négociation de contenu 211
  - news 406
  - niveaux d'erreur
    - Voir* erreur 488
  - nom de domaine 348
  - nombre
    - à virgule flottante 66
    - entier 65
  - nouveautés de PHP 5 10
- O**
- objet 253
    - affichage 261
    - attribut 254, 255, 257, 274
    - clonage 266
    - concept 254
    - constante 256, 260
    - débogage 259
    - égalité et identité 269
    - instance 255
    - instanciation 257
    - méthode 254, 256, 258, 274
    - méthodes et attributs finaux 282
    - test de type 262
  - ODBC 446
  - open\_basedir 333, 347
  - opérateurs 79
    - affectation 79
    - arithmétiques 81
    - bit à bit 86
    - combinés 83
    - comparaison 84
    - concaténation 84
    - incrémentation 83
    - logiques 86
    - modulo 82
    - priorités 87
  - Oracle 446
  - outil de développement 735
- P**
- page référente 211
  - paragraphes 125
  - parse error 56
  - passage par référence 265
  - PDO
    - connexions 450
      - persistante 451
      - multiples 454
    - dernier identifiant généré 463
    - DSN 450
    - exceptions 466
    - exécuter une requête 456
    - gestion des erreurs 465
    - MySQL 37, 450
    - Oracle 451
    - PostgreSQL 451
    - récupérer les données 457, 472
    - requêtes préparées 468
    - transactions 466
  - PDO (PHP Data Objects) 37, 443, 445
  - PDO (Portable data objects) 11, 44
  - PEAR 44
  - Pear 628, 639, 644
  - PECL 44
  - performances 4, 239
  - personnaliser le questionnaire d'erreurs 498
  - php.ini 238
    - Voir* configuration 30
  - PHPEdit 738
  - phpinfo 143
  - PHPLib 602, 613
  - phpMyAdmin 422
  - PHPSESSID 234, 238
  - pires 141
  - politique de gestion d'erreur 509
  - POO (Programmation orientée objet)
    - Voir* objet 253
  - POP 3
    - Voir* e-mail 402
  - portabilité 5
  - PostgreSQL 446
  - Prado 767
  - Programmation orientée objet (POO)
    - Voir* objet 253
  - programme
    - exécution 337, 339
    - interactif 341
    - sécurité 347
    - tâche de fond 340
  - protection de caractères 67
  - protections et échappements 114, 168
  - proxy 207, 635
- R**
- RAD (Rapid Application Development) 749
  - ramasse-miettes 239, 244
  - RDF (Resource Description Framework) 521
  - référence 264
  - références 80, 102, 103
  - register\_globals 168, 713
  - répertoire 310, 323
    - création 326
    - effacement 326
    - parcourir 323
  - réseau 155, 347
  - résolution DNS 155
  - REST (REpresentational State Transfer) 577
  - RSS (Really Simple Syndication) 519
- S**
- safe\_mode 333, 347
  - SAX (Simple API for XML) 539
    - analyse 547
    - espace de noms 542
    - événements 540
    - initialisation 541
    - jeux de caractères 542
    - réagir à des événements 543

- sécurité 47, 241, 463, 705
    - chiffrement 729
    - configuration 708
    - cookie 225
    - cross site scripting 116, 168, 722
    - déontologie 708
    - droit d'accès 710, 716
    - échappement 712
    - e-mail 406
    - exécution de programme 347
    - fichier 333
    - fixation de session 714, 729
    - formulaire 196
    - injection SQL 116, 721
    - mode CGI 709
    - module Apache 710
    - mot de passe 730
    - open\_basedir 711
    - safe\_mode 711
    - session 240, 245, 713, 728
    - stockage 715
    - type de données 720
    - variable globale 713
    - vol de session 728
  - sérialisation 222, 237, 238, 244
  - serveur web 3, 57, 201
  - services Web 3, 520, 575
  - session 231
    - accès concurrents 237
    - cache 239
    - configuration 49, 238
    - écriture 232
    - expiration 239
    - fixation de session 714, 729
    - fonctionnement interne 234
    - gestionnaire personnalisé 241
    - identifiant 235, 240, 245
    - initialisation 238
    - lecture 232
    - module 237
    - nettoyage 239, 244
    - nom 235
    - paramètres
      - d'initialisation 235
      - du cookie 237, 239
    - réécriture des liens 714
    - répertoire de stockage 713
    - sauvegarder des objets 285
    - sécurité 713, 728
      - Voir* sécurité 245
    - stockage 236, 238, 241
    - suppression 235
    - vol de session 728
  - SGBD (Système de gestion de bases de données) 4, 5, 36, 116, 413, 443
    - description 413
    - MySQL
      - Voir* MySQL 443
  - SHA1 160
  - shell 338
  - SimpleXML 526
    - accéder à un attribut 532
    - accéder à un nœud 528
    - afficher le contenu 531
    - export XML 527
    - extension 535
    - import XML 527
    - lister des nœuds 529
    - lister les attributs 532
    - modifier un attribut 532, 533
    - recherche Xpath 533
    - sauvegarde 527
  - site semi-statique 639
  - sleep 285
  - smarty 602, 616
  - SOAP (Simple Object Access Protocol) 520, 575, 577, 580
  - socket 347
    - cliente 361
    - fermeture 349
    - lecture et écriture 349
    - mode bloquant 349
    - nom 358
    - ouverture 348
    - protocole 348
    - serveur 358, 361
    - temps d'expiration 350
    - type 348, 354
    - unix 349
  - SQL (Simple Query Language)
    - sécurité 721
  - SQL (Structured Query Language) 423
    - créer une base 424
    - créer une table 425
    - doublon 440
    - effacer des données 435
    - filtrer avec WHERE 437
    - insérer des données 432
    - jointure 441
    - limiter le nombre de résultats 439
    - maximum et minimum 440
    - modifier des données 434
    - modifier une table 431
    - remplacer des données 436
    - sélection sur plusieurs tables 441
    - sélectionner des données 438
    - supprimer une table 432
    - texte ressemblant avec LIKE 438
    - transaction 441
    - trier 439
  - SQLite 10, 446
  - ssl 348
  - string
    - Voir* chaînes de caractères 66
  - structures de contrôle 88
  - superglobale
    - Voir* variable 180
  - surcharge 286
  - sûreté de programmation 276
  - switch 92
  - Symfony 766, 768
  - syndication 519
  - système de cache 627
    - accès concurrents 631
    - caches globaux 628
    - HTTP 633
    - mise à jour 637
    - Pear 628
    - principe 629
    - proxy 635
    - site semi-statique 639
    - utilité 627
- ## T
- tableau 71
    - associatif 72
    - différence 140
    - doublon 141
    - extraction 135
    - fusion 138
    - index numérique 71
    - intersection 140
    - liste des clés 137
    - liste des valeurs 137
    - multidimensionnel 74
    - recherche d'élément 128
    - remplacement 136
    - séparation 139
    - sérialisation 135
    - taille 127
    - trier 131

tableur 307  
 taille  
   d'une chaîne  
     *Voir* chaîne de caractères 111  
   de fichier 328  
 tampon 317, 356  
   de sortie 377  
 téléchargement 49  
   de fichier 192  
 template 601  
   approche par composants 608  
   approche PHP natif 604  
   bibliothèques Open Source 613  
   choix 610  
   fonctionnement 603  
   PhpLib  
     *Voir* PhpLib 602  
   search&replace 606  
   smarty  
     *Voir* smarty 602  
   templeet  
     *Voir* templeet 603  
   XML et XSLT 609  
 templeet 603, 622  
 temps d'exécution 48, 666  
 test d'égalité  
   *Voir* opérateurs 84  
 throw 504  
 thumbnail 680  
 timestamp 222  
 transtypage 76, 720  
 trier un tableau  
   *Voir* tableau 131  
 troncature 319  
 try 504  
 type 593  
 type de données  
   *Voir* variable, type 60, 64

**U**

UDDI (Universal Description,  
 Discovery and Integration) 578  
 UltraEdit 736  
 UML (Unified Modeling Language)  
 294, 754, 759  
 UML2PHP5 759  
 URL 209

**V**

variable 59  
 affichage 146  
 dynamique 63  
 effacement 62  
 environnement 212  
 existence 62  
 globale 48, 61, 168, 713  
 portée 60, 101  
 sécurité 717  
 superglobale 180, 201, 218,  
 232  
 transtypage 720  
 type 60, 64  
 verrou de fichier 318  
 verrouillage 237  
 vol de session 728  
 vue 764, 771

**W**

wakeup 285  
 WAMP (Windows Apache MyS-  
 QL PHP) 29  
 WDDX 237  
 webmail 386  
 WSDL (Web Services Description  
 Language) 578, 581

**X**

XML (eXtensible Markup Lan-  
 guage) 116, 168, 237, 579

caractères spéciaux 517, 521  
 création manuelle 521  
 DOM 549  
 exemples d'utilisation 514  
 EXSLT  
   *Voir* XSLT 572  
 jeu de caractères 516  
 SAX  
   *Voir* SAX 539  
 SimpleXML  
   *Voir* SimpleXML 526  
 structure 515  
 template 609  
 utilité 513  
 validation 569  
 Xinclude 569  
 Xpath  
   *Voir* Xpath 533  
 XSLT  
   *Voir* XSLT 570  
 XML Schema 581  
 XML-RPC (Remote Procedure  
 Call) 520, 577  
 Xpath 533  
 XSL (eXtensible Styleshe et Lan-  
 guage) 521, 570, 609, 610  
   chargement de la feuille de  
   style 571  
   EXSLT 572  
   initialisation 571  
   paramètres 572  
   sortie HTML 572  
   transformation 571

**Z**

Zend 6, 746  
 Zend Engine 8, 10, 13  
 Zend framework 766  
 Zend Studio 746