les Cahiers du Programmeur

PHP 5

Stéphane **Mariel**

avec la contribution de Jean **Zundel**

EYROLLES

les Cahiers du Programmeur

PHP 5

Stéphane **Mariel**

les Cahiers du Programmeur PHP 5

Avec la contribution de Jean Zundel
et Jean-Marie Thomas

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Avant-propos

PHP n'a pas dix ans. Malgré le succès incontestable qui fait de ce langage l'un des premiers outils du Web – et le module Apache le plus installé, certains peinent encore à reconnaître en PHP un environnement et langage professionnel pour des applications d'entreprise.

Il est vrai que depuis l'explosion de la bulle Internet, le monde des nouvelles technologies (TIC) est entré dans une phase d'industrialisation équivalente à celle que le secteur informatique dans son ensemble a entamée depuis près d'une décennie. Les utilisateurs d'informatique et de solutions TIC attendent désormais des offres et des méthodes qui, au-delà de l'approche « prototype » qui prévalait, démontrent tout au contraire leur fiabilité et leur robustesse.

Cette exigence impose naturellement la mise en œuvre d'outils ou méthodes adaptés, à même de simplifier et de sécuriser le processus de développement. C'est le cas du modèle objet, des design patterns et des frameworks de développement très en vogue aujourd'hui. Dans ce contexte, la version 5 de PHP devrait rallier les plus réticents.

PHP est en effet sur le point de parvenir à allier à sa simplicité d'utilisation et de mise en œuvre, une grande richesse fonctionnelle en se dotant d'un véritable support du modèle objet, de capacités de spécification et d'un support XML enfin mûr. Autant de raisons de voir PHP gagner en popularité, y compris dans les projets les plus critiques.

Quel est l'objectif de cet ouvrage ?

Ce cahier du programmeur propose une mise en pratique simple mais caractéristique des fonctionnalités clés de PHP en matière de développement web, qu'il s'agisse des nouveautés apportées par PHP 5, ou de fonctions déjà présentes dans PHP 4 que leur complexité apparente ou réelle avait pu desservir.

L'annexe B fournit des éléments sur les extensions retenues dans PHP 5.

Loin des bibles qui peinent à s'éloigner des manuels de référence disponibles pour PHP, cet ouvrage ne vise en aucun cas l'exhaustivité, bien au contraire : plutôt que d'égrener la litanie des fonctions et des extensions, l'objectif est de proposer au lecteur une synergie avec le manuel de référence de PHP ou les livres de références existant sur ce sujet.

Les concepts clés sont mis en pratique dans le cadre d'une étude de cas que nous avons choisie simple mais fonctionnellement riche : un système de *chat*. Par cette mise en situation, nous nous attachons à faire connaître les possibilités exceptionnelles de PHP, mais aussi à les rendre plus simples à appréhender.

Cette démarche, que l'on pourrait qualifier d'initiatique sur des sujets phares du développement web, tente d'offrir les moyens et le socle pour approfondir ses connaissances, en tirant notamment parti de la source d'informations inépuisable que constitue le manuel de référence PHP et les livres de référence qui existent sur le sujet.

À qui s'adresse cet ouvrage ?

Cet ouvrage est-il accessible à un lecteur débutant en PHP ? Oui, mais il lui faudra une attention particulière et de la méthode, et ne surtout pas négliger le chapitre d'introduction.

Cet ouvrage s'adresse à tous les utilisateurs de PHP. Au-delà de l'aspect syntaxe du langage, il se penche sur les éléments clés de toute application web ambitieuse : gestion des sessions utilisateur, internationalisation, etc. Il propose également un tour d'horizon des aspects techniques les plus pointus pour réaliser des développements de qualité : utilisation des interfaces, paradigme objet, XML…

Comment lire cet ouvrage ?

Pour découvrir le potentiel de la nouvelle version de PHP, nous dérouleront tout au long de l'ouvrage une étude de cas qui nous servira de fil conducteur. Mais attention, que vous soyez débutant PHP ou amateur averti, ne faites pas l'impasse sur le chapitre d'introduction, qui est l'occasion de (re)découvrir les bases du langage… peut-être certains éléments surprendront-ils ceux

qui pensent déjà bien connaître PHP. Passée cette étape, chacun pourra dévorer le chapitre de son choix.

Les adeptes du modèle objet pourront sans nul doute jeter leur dévolu sur les **chapitres 2, 4** et **6**. Les **chapitres 7 à 10** devraient pour leur part recueillir les suffrages de ceux qu'XML a conquis, ou qui attendent d'être conquis par lui. Les **chapitres 5** (sessions utilisateur), **12** (internationalisation) ou encore **13** (options avancées), peuvent être lus et relus ponctuellement pour découvrir un sujet ou se rafraîchir la mémoire.

Certains chapitres sont plus difficiles d'approche que les autres, c'est le cas des chapitres consacrés à l'objet et à XML. Le lecteur débutant pourra dans ce cas procéder en deux temps : un premier survol, accompagné de quelques tests clés, puis, en fonction des besoins, une relecture plus fine de certains aspects évoqués.

Enfin, l'ouvrage se complète de quatre annexes indispensables : l'**annexe A** résume ce qu'il faut savoir pour héberger son serveur PHP à domicile, l'**annexe B** parcourt les extensions standards intégrées à PHP, l'**annexe C** rappelle l'essence de la norme DOM et enfin, l'**annexe D** offre un récapitulatif de la structure du code associé à notre étude de cas.

Où trouver le code associé à cet ouvrage ?

L'intégralité du code source est disponible en téléchargement sur le site de l'auteur. Par ailleurs, une mise en situation réelle de l'application avec son code source est aussi disponible pour des tests.

- http://www.stephanemariel.com
- http://www.phpsaloon.com

Remerciements

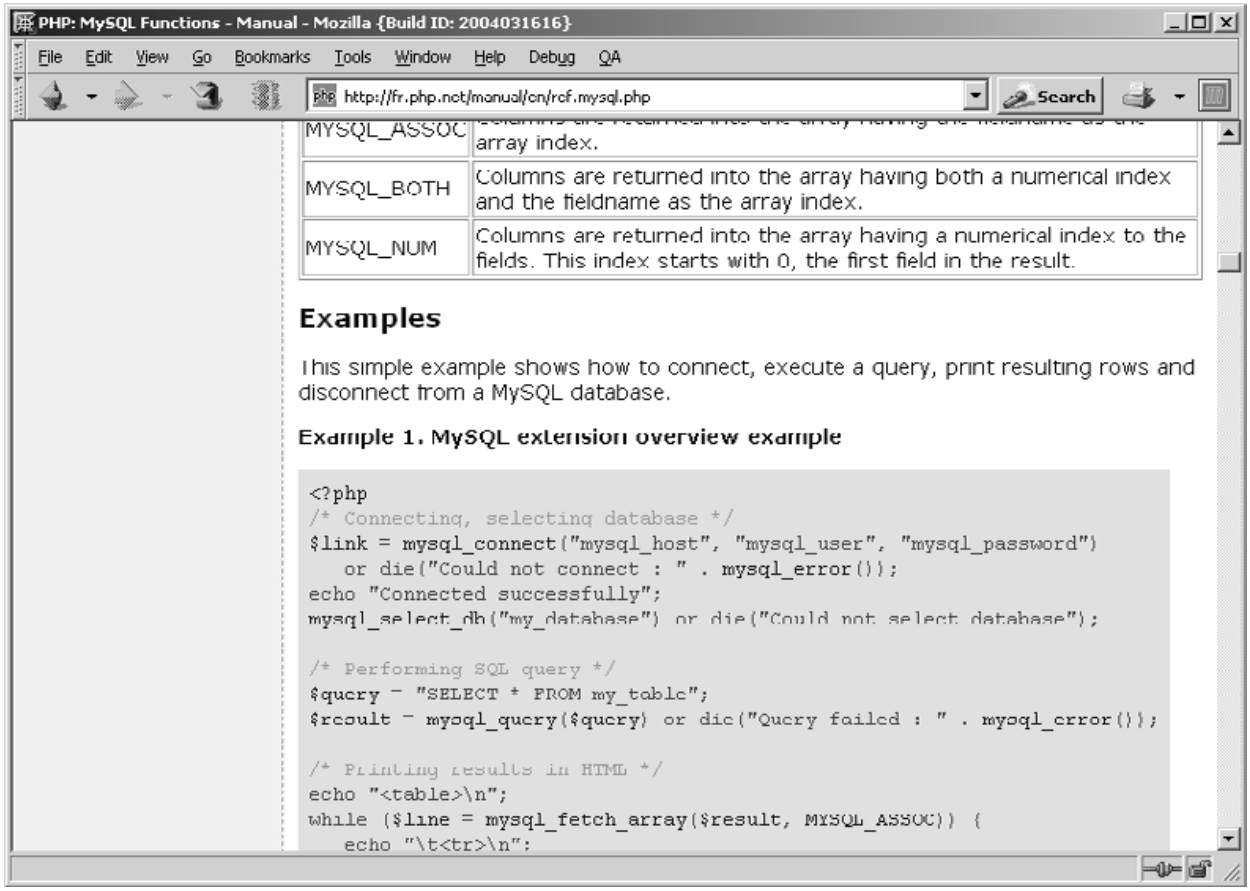
La création d'un ouvrage est un processus de longue haleine. Un cheminement fait de recherches, d'interrogations et de réponses toutes plus ou moins partielles. Sur un sujet comme PHP 5 que sa jeunesse rend encore mouvant, l'aventure réclame encore plus d'énergie.

Merci donc à ceux qui m'ont apporté leur enthousiasme sans compter : Muriel, mon éditrice favorite, Jean qui ne m'aura néanmoins pas converti à Perl et Xavier dont les relectures vous auront certainement préservé, cher lecteur, des tournures les plus alambiquées, sans oublier bien sûr Jean-Marie ainsi que Sophie, Anne et toute l'équipe des éditions Eyrolles.

Stéphane Mariel

- http://www.stephanemariel.com/
- stf@stephanemariel.com

Table des matières


PHP: MySQL Functions - Manual - Mozilla {Build ID: 2004031616}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://fr.php.net/manual/en/ref.mysql.php
Search
MYSQL_ASSOC
array index.
MYSQL_BOTH
Columns are returned into the array having both a numerical index and the fieldname as the array index.
MYSQL_NUM
Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.
Examples
This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.
Example 1. MySQL extension overview example
<?php
/* Connecting, selecting database */
$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
or die("Could not connect : " . mysql_error());
echo "Connected successfully";
mysql_select_db("my_database") or die("Could not select database");
/* Performing SQL query */
$query = "SELECT * FROM my_table";
$result = mysql_query($query) or die("Query failed : " . mysql_error());
/* Printing results in HTML */
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
echo "\t<tr>\n";

Introduction lapidaire à PHP

PHP a l'immense avantage d'être facile à appréhender. Avant d'entrer dans notre étude de cas, quelques rappels nous ont paru indispensables pour bien démarrer et prendre en main les éléments clés du noyau dur de PHP.

SOMMAIRE

MOTS-CLÉS

- Structures de contrôle
- Variables « superglobales »
- Chaînes dynamiques

PHP vous est inconnu ou vous avez besoin de raviver votre expérience ? Cette prise en main est faite pour vous. Fort des quelques éléments clés qui constituent le noyau dur de PHP, vous aurez accès à tout le potentiel du langage. Mais chaque chose en son temps, partons pour un petit tour d'horizon pratique.

PHP, un langage immergé dans les documents

Dans la plupart des cas, les langages de programmation sont conçus pour être mis en œuvre isolément. D'un côté, le code des programmes, de l'autre, des informations. PHP, qui s'inscrit d'emblée dans une logique web, est de ce point de vue totalement différent. À la différence des langages comme C, Java voire Perl, PHP est prévu pour être intégré directement dans des documents (HTML, XML).

Pour cette raison, il est nécessaire de distinguer, dans un fichier PHP, les informations statiques (squelette de page HTML, par exemple) du code PHP lui-même. Par chance, le W3C a tout prévu, et pas seulement pour PHP, mais pour tout langage ayant vocation à être immergé dans des documents web.

Ainsi, le W3C définit des séquences d'échappement qui délimitent les frontières entre le code et l'information classique. Le code est alors désigné sous le terme de *processing instructions*. Deux syntaxes sont possibles. La plus verbeuse, rarement utilisée avec PHP (ou ASP), ne vous est cependant pas inconnue car elle est malgré tout très utilisée, notamment avec JavaScript : il s'agit de délimiter le code avec la balise `script`.

```
<html>
   <head>
      <title>
         Première page dynamique en PHP
      </title>
   </head>
   <body>
      Cette phrase est un contenu statique. <br/>
      <script language="PHP">
         print "Mais celle-ci est dynamique.<br/>";
      </script>
      Et celle-ci de nouveau statique.
   </body>
</html>
```

Cette syntaxe est valide pour l'ensemble des langages, JavaScript côté client, ou ASP et PHP côté serveur.

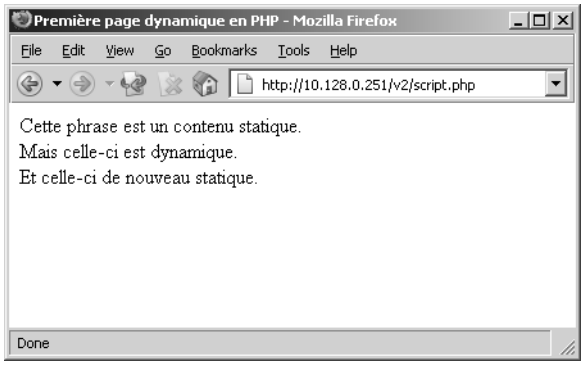


Figure 1 Première page dynamique avec PHP

Il existe néanmoins une syntaxe raccourcie plus répandue dans le cas de PHP. Le début du code est marqué par `<?php` et la fin par `?>`. De manière plus générale, la syntaxe est de la forme `<?X ... ?>` où `X` représente le langage dans lequel est écrit le code. Notre exemple peut alors s'écrire :

```
<html>
   <head>
      <title>
         Première page dynamique en PHP
      </title>
   </head>
   <body>
      Cette phrase est un contenu statique. <br/>
      <?php
         print "Mais celle-ci est dynamique.<br/>";
      ?>
      Et celle-ci de nouveau statique.
   </body>
</html>
```

Cette syntaxe, déjà très simple, peut encore être raccourcie en supprimant le `php` de la balise d'ouverture. Toutefois, cette dernière version, quoique très utilisée, n'est pas vraiment orthodoxe et peut entrer en collision avec d'autres déclarations comme celle de la déclaration initiale des documents XML. Mieux vaut donc taper trois caractères de plus.

Comme le montrent nos deux premiers exemples, le code PHP est totalement immergé dans l'information. Un document PHP pourrait même d'ailleurs ne pas contenir de code (pour une phase de test) et être totalement statique. Progressivement, il sera possible d'ajouter ici ou là dans le document des instructions PHP, car il n'existe pas de limites quant au nombre ou au positionnement du code PHP dans le document. Celui-ci peut être utilisé pour obtenir tout un morceau HTML, un simple attribut ou construire dynamiquement un tableau.

REGARD DU DÉVELOPPEUR

PHP en ligne de commande

Même si PHP a été conçu dans une logique web, rien n'interdit de créer des documents entièrement PHP. Dans ce cas, le premier élément du document est la balise `<?php` et le dernier la balise `?>` car, quoi qu'il arrive, un document est toujours ouvert en mode statique et lu comme tel jusqu'à ce que l'on rencontre du code PHP.

Enfin, si PHP est le plus souvent utilisé en association avec un serveur web, rien n'interdit de s'en servir en ligne de commande pour créer ses propres applications. Il existe d'ailleurs une version de PHP couplée à la bibliothèque graphique GTK pour construire des applications autonomes.

▸ http://gtk.php.net/

Une syntaxe simple

Nos premiers exemples le démontrent, la syntaxe de PHP n'est vraiment pas difficile et tient en quelques règles simples :

- Les instructions sont séparées par un « ; » (comme dans la plupart des langages sensés).
- Le langage n'est pas sensible à la casse des caractères (majuscules, minuscules).
- Les variables sont désignées par leur nom, précédé du symbole « `$` » (comme en Shell) et PHP n'est pas typé. Une variable peut donc contenir toute information, quelle que soit la nature de celle-ci. Exemple : `$niveau`, `$message`.
- Il n'est pas nécessaire de déclarer les variables, même si cela reste possible avec l'instruction `var`. Plusieurs instructions peuvent être regroupées dans un bloc pour former une instruction composée ; le bloc est alors délimité par des accolades (« `{` » et « `}` »), là encore, comme dans la grande majorité des langages courants.
- Les commentaires peuvent être introduits à la manière de C++ sur une seule ligne avec « `//` » ou comme dans C avec « `/*` » et « `*/` ».

On le voit, PHP ne mise pas sur l'originalité et si vous avez déjà expérimenté un langage de programmation parmi les grands standards, vous ne risquez guère d'être décontenancé.

Du côté des expressions, PHP supporte grosso modo le même jeu d'expressions que le langage C. Les opérateurs logiques sont comme en C « `==` » pour la comparaison, « `&&` » pour le « et logique » et « `||` » pour le « ou ». Les adeptes de Pascal ou d'Ada pourront néanmoins utiliser « `or` » et « `and` ». PHP dispose, en outre, de la très grande majorité des fonctions numériques et des opérations sur les bits.

COMPRENDRE Expressions et instructions

En PHP comme en C, les instructions sont le plus souvent perçues comme des expressions et peuvent être intégrées dans des expressions plus complexes.

ASTUCE Quelques raccourcis de syntaxe

PHP copie C encore sur un autre point : la possibilité de recourir à des syntaxes raccourcies pour certaines petites instructions courantes :

Forme classique	Forme raccourcie
`$X = $X + $Y;`	`$X += $Y;`
`$X = $X - $Y;`	`$X -= $Y;`

Comme on peut s'y attendre, cette syntaxe s'étend à la multiplication et à la division, ainsi qu'aux opérateurs logiques.

Enfin, PHP supporte aussi les syntaxes du type `$X++` (respectivement `$X--`) pour incrémenter (respectivement décrémenter) une variable.

Des types élémentaires... mais pas de typage

Nombres et caractères

Si chaque variable en PHP peut contenir indifféremment tout type d'information, PHP n'en propose pas moins les types élémentaires et traditionnels :

- booléens ;
- nombres ;
- chaînes de caractères.

En PHP, tous ces types peuvent être manipulés comme des chaînes de caractères ; la conversion est alors transparente. De même, les booléens `true` et `false` peuvent implicitement être convertis en `1` et `0` au besoin.

Les chaînes de caractères sont représentées soit entre apostrophes simples (*quotes*) « `'chaîne'` », soit entre guillemets (« `"chaîne"` »). La distinction entre les deux représentations est importante car nous verrons par la suite que l'utilisation des guillemets se prête à divers raffinements. Par défaut, le plus sage et le plus performant est toujours d'utiliser les apostrophes simples (*simple quotes*).

L'accès à chaque caractère d'une chaîne s'effectue en utilisant les accolades , par exemple, si `$valeur` contient une chaîne de caractères, alors le deuxième caractère peut être obtenu via `$valeur{1}`. Attention, PHP ne dispose pas du type `caractère`, le résultat est donc une nouvelle chaîne de caractères. La fonction `strlen()` permet de connaître la taille d'une telle chaîne.

Tableaux

Si les types de données disponibles dans PHP peuvent paraître simplistes, la notion de tableau proposée est très riche et permet de créer des tableaux associatifs dynamiques et multi-dimensionnels.

Il n'est pas nécessaire de déclarer un tableau. La simple affectation d'un élément provoque la création implicite d'un tableau et son stockage dans la variable. L'accès à un élément particulier se fait en utilisant les crochets.

Dans l'exemple suivant :

```
$a[6] = 7;
```

La variable `$a` contient alors un tableau d'un seul élément (dont la clé d'accès est 6). Il est alors possible d'ajouter autant d'éléments que souhaité, voire d'ajouter des dimensions.

```
$a[9]['couleur'] = 'rouge';
$a['element'] = 'oxygène';
```

Ici, nous avons ajouté une deuxième dimension, `$a[9]` est alors un tableau. Par ailleurs, l'indexation des tableaux ne se limite pas aux seuls entiers.

La fonction `print_r()` permet d'obtenir une représentation texte d'un tableau comme le montre la figure 2. La fonction `var_dump()` disponible pour tous les types de données permet d'obtenir encore davantage d'informations.

À RETENIR Utilisez les quotes ou les guillemets pour les index de type texte

Quand l'index d'un élément dans un tableau est une chaîne de caractères, il est possible d'utiliser `$a[rouge]` et non `$a['rouge']` pour accéder à l'élément.
Ceci n'est en réalité qu'une tolérance (PHP recherchant dans le premier cas une constante nommée `rouge`) et nous le déconseillons très fortement.

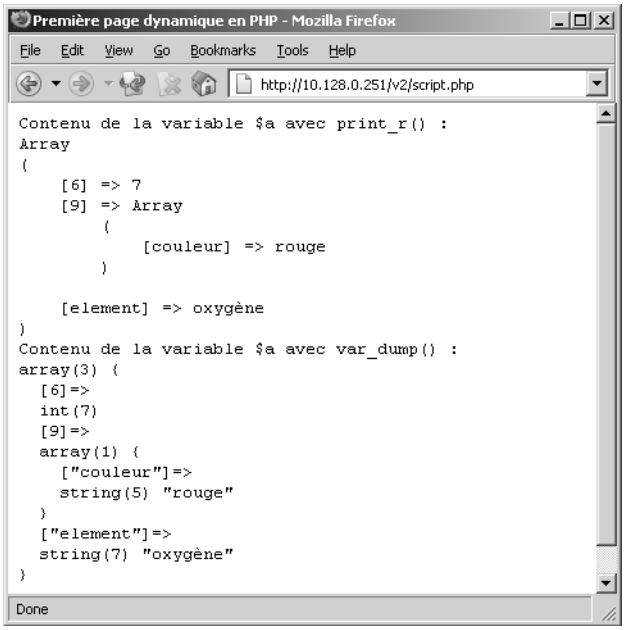


Figure 2 Structure d'un tableau avec `print_r()` et `var_dump()`

Enfin, la fonction `array()` définit un tableau de manière globale, qu'il soit associatif ou non :

```
$tableau = array( 'element 0',
                  'fruit' => 'pomme',
                  array( 'message' => 'Bonjour',
                         'code' => 3));
```

Dans cet exemple, `'element 0'` va se voir attribuer l'index `0`, faute d'avoir un index imposé. Il en va de même pour le troisième élément, qui est un tableau.

Le résultat présenté par `print_r()` est le suivant :

```
Array
(
    [0] => element 0
    [fruit] => pomme
    [1] => Array
        (
            [message] => Bonjour
            [code] => 3
        )
)
```

L'instruction `unset()` peut alors être utilisée pour supprimer un élément du tableau :

```
unset ($tableau['fruit']);
```

Cette même instruction peut tout autant supprimer une variable classique ou être utilisée en collaboration avec `isset()` qui détermine pour sa part l'existence ou non d'une variable ou d'un élément de tableau.

Sous-programmes et modularité

Il va de soi qu'un programme PHP conséquent ne se limite pas à quelques instructions exécutées en séquence. Comme l'ensemble des langages, PHP permet de définir des fonctions. La syntaxe utilisée est la suivante :

```
// une fonction foo()
function foo($param1, $param2, $param3 = 45) {
   // code de la fonction
   return $param1+$param2;
}
// procédure bar()
function bar($param1) {
   // code de la fonction bar sans return,
   // ou avec un return sans paramètre.
}
```

Il n'est fait aucune différence entre procédures et fonctions au niveau de cette syntaxe. Dans les fonctions, on utilisera l'instruction `return` pour terminer l'exécution et retourner le résultat.

Pour aller plus loin en termes de modularité, PHP permet d'inclure des fichiers avec quatre instructions distinctes :

- `include`
- `include_once`
- `require`
- `require_once`

À RETENIR **Accéder aux variables globales depuis une fonction**

Dans une fonction, seules les variables définies localement ou les paramètres sont accessibles. Pour accéder aux variables définies dans le corps principal du programme il faut utiliser l'instruction `global` : `global $mavariableglobale;`

Ces quatre versions correspondent en réalité aux combinaisons de deux variantes :

- `require` déclenche une erreur et arrête le programme si le fichier inclus n'est pas trouvé. `include` se contente de provoquer un avertissement.
- les versions « `_once` » gèrent les risques d'inclusions multiples. Avec `include_once` (comme avec `require_once`), PHP s'assure que chaque fichier n'est inclus qu'une seule et unique fois, même si plusieurs demandes sont faites.

Dans la pratique, on utilisera de préférence l'instruction `require_once` (sauf besoins spécifiques) car elle présente le maximum de garanties.

Il faut retenir que les fichiers inclus sont traités de façon rigoureusement identique à tout fichier PHP standard. L'interpréteur PHP commence donc l'analyse du document en mode « contenu statique ». Comme dans un fichier classique, il faudra donc utiliser les balises `<?php` et `?>` pour entourer le code PHP.

Fichier principal

```
<?
require_once'module.php';
foo();
?>
```

Fichier module.php correct

```
<?
   function foo() {
      return'Hello';
   }
?>
```

Fichier module.php incorrect

```
function foo() {
   return'Hello';
}
```

Pour les modules composés uniquement de code PHP (c'est le cas des bibliothèques de fonctions), la balise `<?php` sera positionnée au tout début du fichier et `?>` à la fin.

Une source très répandue d'ennuis consiste d'ailleurs à oublier, soit sur la première ligne (avant `<?php`), soit en fin de fichier (après `?>`) des blancs (espaces, tabulations) ou des retours à la ligne. Lors de l'inclusion, ces caractères parasites seront affichés dans le document HTML final. Avec, à la clé, des problèmes de mise en page et très souvent des problèmes d'en-tête HTTP (comme pour les sessions).

Les structures de contrôle habituelles

Pour décrire la structure algorithmique de votre programme, PHP propose la panoplie complète des structures de contrôle habituelles :

- `for`,
- `while` et `do/while`,
- `if` et `if/else`,
- `switch`.

En termes de syntaxe, PHP reste très conventionnel (voir l'exemple ci-après). Dans tous les cas de figure, une instruction simple peut être remplacée par un bloc d'instructions entre accolades.

```
// Test simple :
if ( $i == 0 )
   print'I est nul';
// Test avec alternative.
if ( $i == 3 )
   print'I vaut 3';
else
   print'I est différent de 3';
// Tests multiples
switch($i) {
   case'3':
      print'I vaut 3';
      break;
   case'foo':
      print'I est une chaîne et vaut foo.";
      break;
   default:
      print'I est banal.';
}
```

Dans le cas de l'instruction `switch`, on veillera à ne pas omettre `break`, sauf cas particulier. À défaut, plusieurs alternatives seront agglomérées.

```
switch($i) {
   case'3':
      print'I vaut 3';
   case'foo';
      print'I est une chaîne et vaut foo.";
      break;
   default:
      print'I est banal.';
}
// Si $i contient la valeur 3, affichera :
// I vaut 3.
// mais aussi
// I est une chaîne et vaut foo.
```

Les structures de boucles sont, elles aussi, très conventionnelles :

```
for($i = 0; $i < 10; $i++) {
   print "Le compteur vaut $i\n";
   // ...
}
$i = 0;
while ($i < 15) {
   print "Le compteur vaut $i\n";
   $i++;
}
```

On peut sortir du mode PHP à tout moment, pour revenir au contenu statique, ce qui peut surprendre de prime abord :

```
<table>
<?php
for($i = 0; $i < 10; $i++) {
?>
   <tr><td><?php print "Le compteur vaut $i\n"; ?></td></tr>
<?php
}
?>
</table>
```

Ici, chaque itération de la boucle produit une ligne dans une table HTML. Cette segmentation du code peut être utilisée partout en PHP. On peut donc créer un squelette HTML et intercaler le code PHP nécessaire aux bons endroits. Toutefois, dans le cas particulier des boucles, cet usage n'est pas recommandé si l'on souhaite conserver le maximum de performances (et, accessoirement, de lisibilité).

Il est enfin possible d'optimiser le parcours des tableaux en PHP avec des instructions particulièrement adaptées comme `foreach` et `each` :

```
$tableau = array ( 'rouge' => 'v1', 9 => 'v2', 'foo' => 'v3' );
```

`$valeur` prend successivement les valeurs v1, v2, v3.

```
foreach($tableau as $valeur) {
   print "Élément courant : $valeur\n";
}
```

Dans cette adaptation, la clé et la valeur sont disponibles à chaque instant, `$cle` prendra les valeurs `rouge`, `9`, `foo`.

```
foreach($tableau as $cle => $valeur) {
   print "Élément $cle du tableau : $valeur\n";
}
```

Pour chaque tableau, une variable interne indique l'élément courant. `Reset()` repositionne ce pointeur en début de tableau.

```
reset($tableau);
```

`Each()` retourne la ligne courante du tableau (sous la forme d'un tableau de deux éléments : clé et valeur. En outre, le pointeur interne est avancé d'une ligne.

```
while ( $ligne = each($tableau) ) {
```

```
   print "Élément $ligne[0] du tableau : $ligne[1]\n";
}
reset($tableau);
while ( list($cle , $valeur) = each ($tableau) ) {
   print "Élément $cle du tableau : $valeur\n";
}
```

L'instruction `list()` affecte directement plusieurs variables à partir d'un tableau (ici celui retourné par `each()`).

Les tableaux superglobaux

Jusqu'à présent, nous avons traité des fonctionnalités propres à PHP, un peu à la manière d'un langage autiste. En réalité, PHP, langage conçu pour le Web, est tout sauf replié sur ses propres capacités, d'abord parce qu'il dispose d'une pléiade d'extensions qui sont autant de fenêtres sur l'extérieur, mais aussi et surtout parce qu'il propose un accès direct à toutes les informations échangées entre l'utilisateur (le plus souvent via son navigateur et un serveur web comme Apache) et l'interpréteur PHP.

Pour cela, PHP propose des tableaux d'informations accessibles sous forme de variables prédéfinies. Le nom de ces variables commence le plus souvent par « _ ».

COMPRENDRE Les variables prédéfinies

Le manuel de PHP décrit de manière approfondie les différentes variables évoquées ici.

- http://www.php.net/manual/en/reserved.variables.php

Nom	Contenu
$GLOBALS	Ce tableau contient l'ensemble des variables globale définies. Il constitue une alternative à l'utilisation de l'instruction `global` puisque les variables globales peuvent directement être lues et écrites à partir de ce tableau.
$_COOKIE	Ce tableau réunit l'ensemble des variables transmises par le navigateur du visiteur de la page PHP via les cookies.
$_ENV	L'interpréteur PHP est toujours exécuté dans un environnement donné, avec différentes variables système (les chemins, le nom de l'architecture). Ce tableau permet de lire ce jeu de variables et même peut les modifier ou en créer de nouvelles.
$_FILES	Dans un formulaire HTML, il est possible de télécharger un fichier. L'ensemble des fichiers téléchargés est décrit dans ce tableau. Il est alors possible de savoir où ceux-ci sont stockés (pour les recopier par exemple dans un endroit propre à l'application).
$_GET	Toujours dans le cadre des formulaires HTML, $_GET contient la liste des variables transmises via la méthode HTTPGET (c'est-à-dire sur l'URL).
$_POST	En complément, les variables transmises via la méthode POST sont disponibles dans la variable $_POST.
$_REQUEST	Le tableau $_REQUEST agrège $_POST, $_GET, $_COOKIE en un seul tableau fédérateur. L'ensemble de ces valeurs a été transmis à l'interpréteur via le réseau, depuis le navigateur de l'utilisateur. Par définition, il regroupe les variables dont le contenu pourrait être nocif et qui doivent être traitées avec précaution.
$_SERVER	Ce tableau permet de connaître le détail de la requête en cours (nom, paramètres, chemin exact de la page demandée) et les éléments plus spécifiques au serveur web (nom, version) ainsi qu'à la connexion (IP distante, paramètres du navigateur).
$_SESSION	Ce tableau permet de manipuler directement les données de session. Il est décrit plus en détail dans le chapitre 5.

Toutes ces variables sont dites « superglobales », c'est-à-dire qu'elles sont visibles et disponibles à tout instant dans un programme PHP. Il est donc inutile de les déclarer globales dans les fonctions, cette visibilité étant implicite.

Des facilités exceptionnelles sur les chaînes de caractères

Dans un langage comme PHP, à la fois non typé et destiné à construire du contenu, les opérations sur les chaînes de caractères sont fondamentales. PHP dispose de multiples fonctions agissant sur les chaînes (comparaison, remplacement, expressions régulières, transformations en tableau) mais le langage permet aussi de procéder à des manipulations directement dans les chaînes elles-mêmes.

Pour délivrer le maximum de performances, PHP distingue les chaînes statiques (entre apostrophes) des chaînes qu'on pourrait qualifier de dynamiques (entre guillemets) et qui feront l'objet d'un traitement particulier.

Il est possible, la plupart du temps, de se contenter de chaînes statiques, c'est le cas notamment pour le nom des fichiers à inclure ou pour certains messages.

Dans de nombreuses situations, la chaîne attendue consiste cependant en une agglomération de portions statiques, de valeurs stockées dans des variables ou encore de caractères spéciaux.

Vous pouvez bien entendu créer une telle chaîne en concaténant les différents éléments (c'est le rôle de l'opérateur « . »).

```
$str = 'Ceci est une' . $superbe[$index1][$index2] . 'chaînes de ' .
$caractères . ' !';
```

Nous agglomérons, dans notre exemple, outre des chaînes statiques, un élément de tableau, à savoir la valeur d'une variable. On imagine que cette notation peut devenir très lourde si les chaînes à créer sont quelque peu complexes.

Avec les chaînes dynamiques, PHP offre une alternative très confortable. Il devient effectivement possible d'intégrer directement dans les chaînes la valeur des variables PHP.

```
$str = "Le nombre de colis est $stock !";
```

Ici, le contenu de la variable `$stock` sera substitué dans la chaîne. Comme certaines situations peuvent être difficiles à interpréter, PHP utilise aussi les accolades pour délimiter le nom de la variable ou l'expression à substituer.

```
$nb = 3;
$stock[$nb] = 12;
// sans accolades, seules les substitutions simples vont être reconnues
print "Le nombre de colis de la semaine $nb est $stock[$nb]";
// avec les accolades il est possible de délimiter plus précisément la
variable
print "Les données pour la semaine ${nb}[2004] sont disponibles";
$stock[$nb]['lundi'] = 34;
// voire de substituer des variables plus complexes (noter que
l'accolade ouvrante est placée avant le « $ ».
print "Le nombre de colis reçus le lundi de la semaine $nb est
{$stock[$nb]['lundi']}";
```

Enfin ce type de chaînes permet aussi d'insérer des caractères spéciaux comme la tabulation (« \t ») ou le retour chariot (« \n »). Notre exemple précédent peut donc s'écrire :

```
$str = "Ceci est une ${superbe[$index1][$index2]} chaîne de $caractères
!\n"
```

Enfin, dernière possibilité, inspirée des shells Unix et surtout utile pour écrire des chaînes longues, l'utilisation de l'opérateur « <<< ». La syntaxe est un peu particulière :

```
$variable = <<<FIN
bla-bla, $encore plus
de
bla-bla
FIN;
// suite de votre code PHP
```

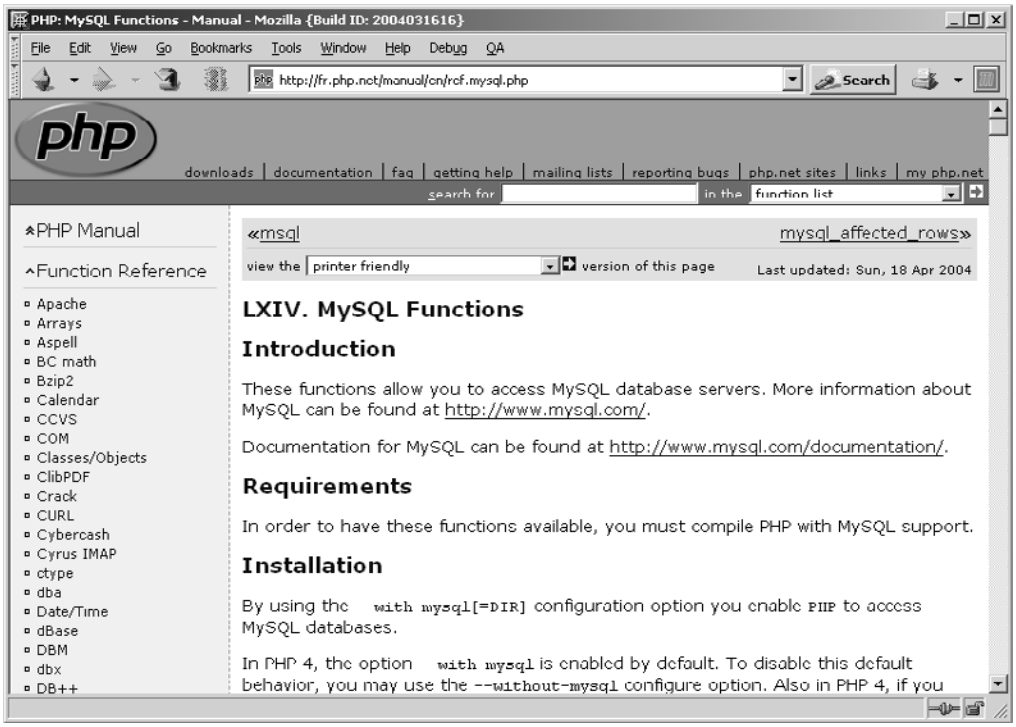
Les possibilités en termes de caractères d'échappement et de remplacement sont identiques à celles disponibles pour les chaînes entre guillemets. Il faut noter que le mot qui délimite la fin du texte, dans notre exemple `FIN`, est libre et ne doit naturellement pas être utilisé dans la chaîne. Par habitude, on utilise souvent `EOF` ou `EOT` (respectivement *end of file* et *end of text*). Enfin, ce délimiteur doit apparaître seul, suivi d'une virgule sur la dernière ligne (pas d'espace ou d'autre caractère parasite).

Méthode de survie

L'ensemble des éléments précédents représente le strict minimum vital pour démarrer l'exploration de PHP. À partir de là, tout est permis. La galaxie PHP peut paraître bien intimidante avec ses dizaines de milliers de fonctions, mais un peu d'habitude et de méthode suffira non seulement à garder la tête hors de l'eau, mais à rapidement tirer profit de cet environnement tout en améliorant sensiblement sa productivité avec PHP.

Premier point, il faut savoir exploiter pleinement le manuel en ligne de PHP. C'est une habitude que l'on peut avoir perdu à force de subir les documentations souvent aseptisées des logiciels propriétaires. Avec PHP, les termes de manuel et de documentation prennent, au contraire, tout leur sens.

Figure 3
Le résumé de l'extension MySQL



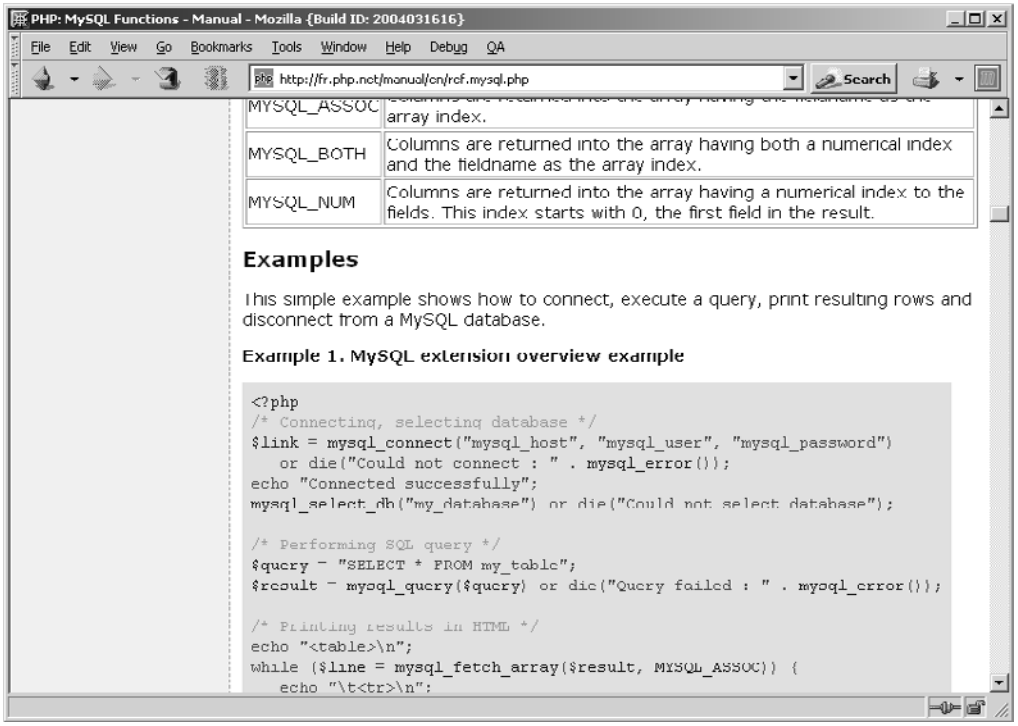


Figure 4
Un des exemples disponibles avec le résumé

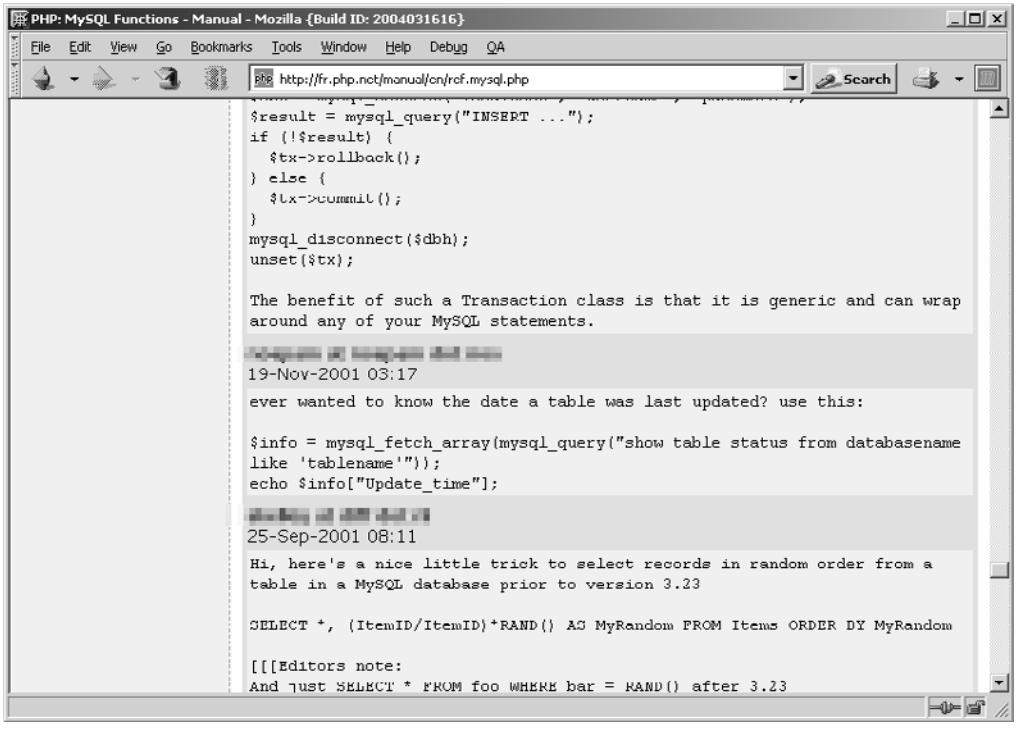


Figure 5
Quelques extraits des contributions disponibles

Tout PHP est dans ce manuel. C'est pourquoi, dès maintenant, adoptez les réflexes des développeurs PHP aguerris. Vous avez un besoin particulier ? Allez directement dans le manuel, explorez les extensions et vous ne manquerez pas de détecter celle qui correspond à votre attente. Si vous découvrez l'extension, lisez attentivement le résumé général au lieu de plonger tête baissée dans le détail des fonctions. Le plus souvent, vous aurez un exemple type et une explication du rôle et du fonctionnement général de l'extension dans PHP.

Naturellement, la suite logique consiste à se rendre sur la page de la fonction précise qui répond à vos besoins. Là encore, outre le synopsis, vous trouverez le plus souvent plusieurs exemples clés.

Vous pouvez naturellement télécharger le manuel PHP directement sur votre ordinateur. Il est disponible dans de multiples formats, y compris sous forme de fichiers d'aide Microsoft Windows. Une traduction française est aussi disponible, sans parler des versions intégrées aux outils de développement.

Mais si télécharger le manuel permet d'accéder plus rapidement à une information précise, l'expérience a montré que le manuel en ligne est un outil beaucoup plus précieux. En effet, au-delà de l'aspect documentaire, chaque page du manuel en ligne est complétée par les commentaires des utilisateurs. Ce dernier point fait du manuel en ligne un outil fantastique pour comprendre PHP. Que vous découvriez PHP ou que vous soyez un expert en la matière, il y a fort à parier qu'un autre utilisateur, et probablement même un grand nombre, a partagé vos incompréhensions et vos difficultés. Ainsi, ce sont des milliers d'utilisateurs à travers le monde qui, chaque jour, annotent ou complètent le manuel PHP en ajoutant des éclaircissements et des exemples qui permettent de mieux bénéficier de telle ou telle fonctionnalité.

C'est pourquoi le manuel en ligne est irremplaçable. Il peut être judicieux, dans cette optique, de ne pas limiter sa recherche au manuel PHP traduit en français. Car même si la communauté francophone des utilisateurs de PHP est très importante, le nombre de contributions est encore plus important pour le manuel anglais. Et n'ayez crainte, l'anglais des informaticiens n'est pas bien difficile à comprendre, probablement d'ailleurs au grand dam des vrais anglophones et des linguistes !

Avec le manuel, vous avez donc immédiatement accès à l'information de référence ainsi qu'aux exemples additionnels des utilisateurs. Si tout cela se révèle insuffisant, les forums et les bibliothèques de scripts peuvent vous apporter des réponses encore plus concrètes.

Pour finir, Google et les moteurs de recherche en général sont vos amis ; quelques mots-clés et vous pourrez explorer une liste encore plus importante de ressources PHP. Au fil du temps, vous avez là un bon moyen pour vous constituer un annuaire de sites, à votre convenance.

RÉFÉRENCE Où trouver le manuel PHP ?

Le site de référence est naturellement www.php.net, mais le miroir français peut être souvent plus véloce. Dans tous les cas, les commentaires des utilisateurs sont bien les mêmes.
Pour les manuels en français, la société Nexen propose une mine d'informations, dont les manuels de PHP et de MySQL.

- http://ww.php.net/manual/en
- http://fr.php.net/manual/en
- http://www.nexen.net/docs/php/annotee/manual.php

Quelques adresses

Il existe une multitude de sites consacrés à PHP, en voici quelques-uns parmi les plus connus :

- http://www.phpfrance.com
- http://www.hotscripts.com/
- http://www.phpindex.com/

Enfin, l'Association française des utilisateurs de PHP (AFUP) propose sur son site une liste de ressources complémentaires.

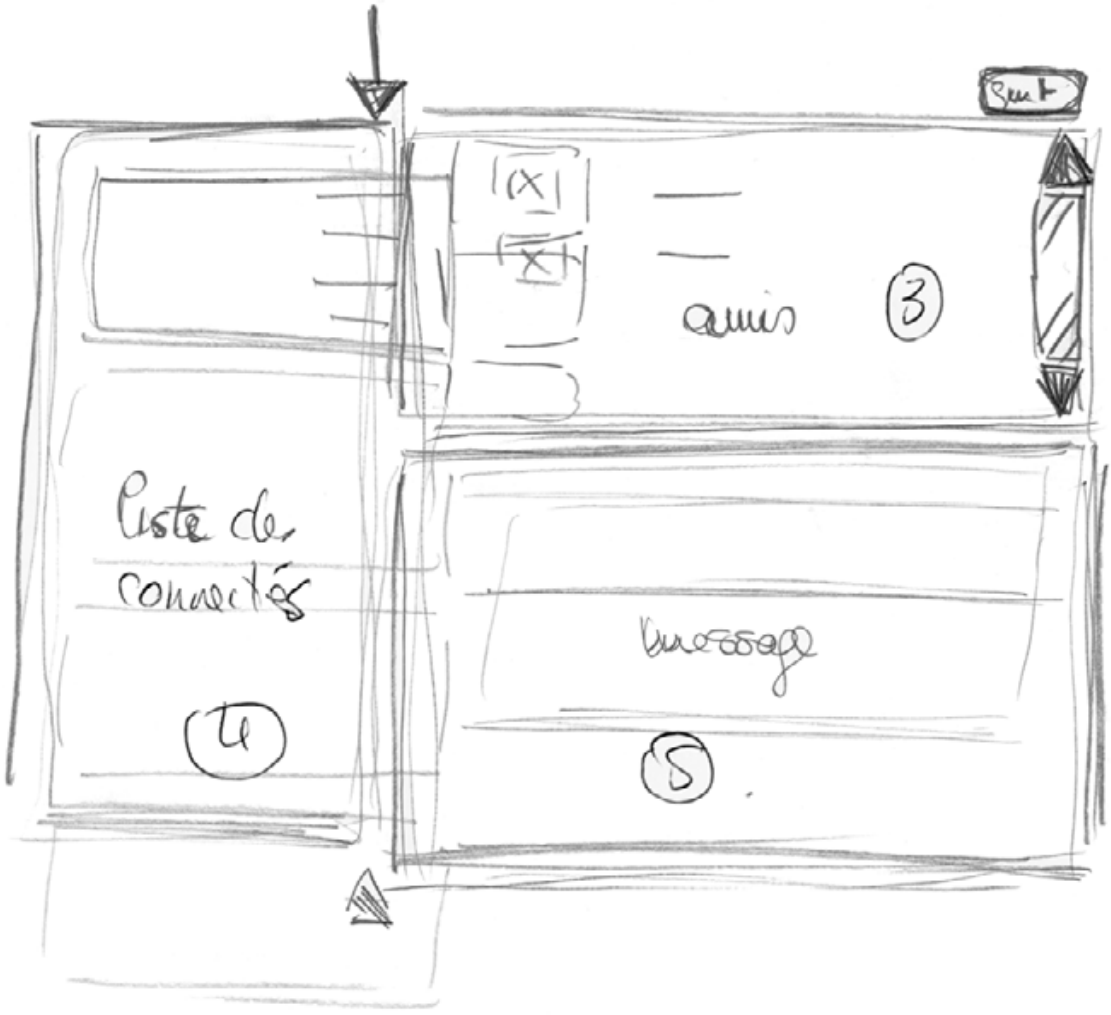
- http://www.afup.org

En résumé...

La syntaxe de PHP est un mélange cohérent des syntaxes les plus connues et les plus usitées. Elle ne recèle aucune originalité, ce qui explique sans doute l'extrême facilité à prendre en main ce langage. Débuter avec PHP est donc un jeu d'enfant. Après cette phase initiale, PHP n'est pas plus complexe pour autant. Il suffit de garder toujours à l'esprit que, passé l'apprentissage de la syntaxe, la puissance de PHP ne se révèle que dans sa fantastique bibliothèque d'extensions.

En fonction de vos besoins, la bonne technique consiste donc à explorer les extensions disponibles pour choisir la plus adaptée. Le manuel en ligne de PHP se révèle alors l'outil indispensable pour découvrir le détail des fonctionnalités et les commentaires des utilisateurs sur toutes les subtilités qui pourraient avoir échappé au manuel.

chapitre 1



L'application web exemplaire en PHP 5

Notre étude de cas, une application de discussion en ligne, est choisie simple mais riche.

Elle illustre en quoi PHP, et en particulier PHP 5 avec son implémentation objet, est adapté aux besoins des applications et sites Internet.

SOMMAIRE

MOTS-CLÉS

- Cahier des charges
- Spécifications
- Maintenance
- Langage glu

PHP Saloon, un chat en PHP

PHP et a fortiori PHP 5 sont d'une grande richesse fonctionnelle. Tous les exemples de ce livre reposent sur une étude de cas simple, qui nous servira de fil conducteur. Il s'agit d'un service de dialogue entre connectés, c'est-à-dire un chat, pour utiliser le terme du moment. Nous avons baptisé ce service : PHP Saloon.

PHP Saloon représente le prototype même du site dynamique. Cette véritable application web met en scène les éléments classiques attendus pour tout service professionnel :

- création de sessions personnalisées ;
- accès à des services réservés ;
- délivrance d'informations ;
- parcours d'un catalogue.

Ce sont là des éléments qu'on pourrait tout à fait retrouver sur un site marchand.

Et en PHP4 ?

Nous aurions pu développer PHP Saloon en utilisant les seules possibilités de PHP 4. C'était une alternative. Cependant, nous le verrons tout au long de cet ouvrage, un site Internet dynamique requiert du temps et de l'énergie. C'est une raison plus que suffisante pour ne pas se priver des améliorations considérables apportées par PHP 5.

PHP Saloon en détail

PHP Saloon est donc une application de chat en ligne qui crée une sorte de point de rencontre, de lieu d'échange virtuel sur Internet. Ces applications sont très en vogue et la plupart des fournisseurs de messagerie comme Yahoo! ou MSN ont élargi leur offre à ce type de communautés virtuelles.

Le système n'est pas très éloigné des solutions de messagerie instantanée. Cependant, dans la pratique, la messagerie instantanée est plutôt utilisée entre cercles d'amis déjà constitués. Le chat au contraire propose plutôt une approche de type melting pot où les contacts s'établissent entre des personnes qui ne se connaissent pas forcément. PHP Saloon propose un mélange des deux mondes en permettant à la fois de discuter avec ses amis et d'entrer facilement en contact avec les autres visiteurs connectés.

Chaque visiteur s'identifie à son arrivée, ce qui lui permet d'apparaître dans l'annuaire des personnes connectées. Grâce à cet annuaire, les différents utilisateurs vont pouvoir se contacter et échanger des messages. Naturellement, chaque habitué du système peut créer son annuaire personnel pour référencer ses amis et ne pas avoir à les chercher dans la liste lors de ses visites successives.

Cette première description, à laquelle adhère la quasi-totalité des systèmes de chat, permet d'ores et déjà de se faire une idée de l'architecture que pourrait avoir PHP Saloon.

Pour le moment, nous allons identifier les différents éléments mis en jeu, préciser notre vision de l'application et composer ainsi un cahier des charges.

Une inscription préalable

Tout système de chat, et plus généralement d'échanges entre connectés, repose sur la notion d'annuaire. Contrairement à la visite d'un site web classique, il est nécessaire de s'inscrire avant d'utiliser le service.

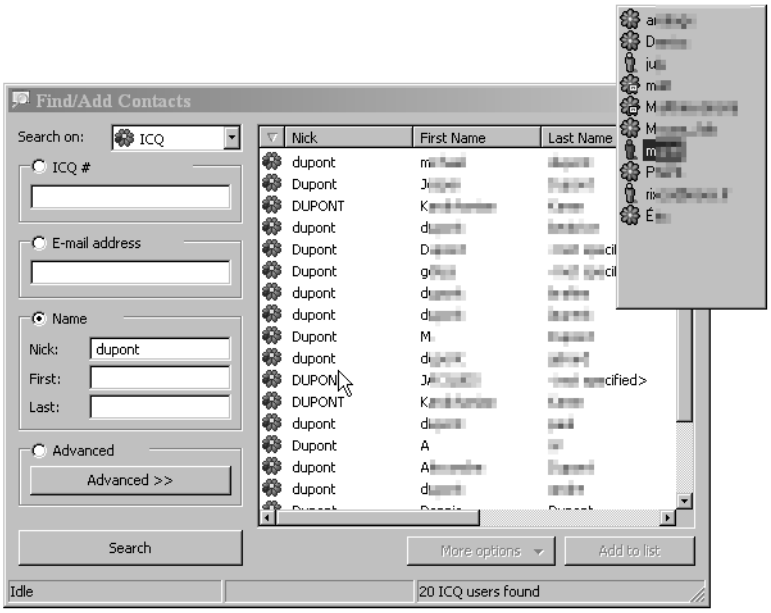


Figure 1–1 Exemple de messagerie instantanée avec annuaire des utilisateurs

Lors de la première visite, l'inscription va permettre au visiteur de se constituer un portrait. Celui-ci permettra aux autres connectés de se faire une idée du visiteur avant de le contacter éventuellement.

Chaque chat est libre de définir les éléments du portrait. Pour PHP Saloon nous allons adopter les éléments les plus classiques :

- nom de code ;
- âge ;
- sexe ;
- pays ;
- ville ;
- émoticône.

Une identification à chaque visite

Une fois inscrit, le visiteur devra s'identifier lors de chaque visite. Il précisera son nom de code et le mot de passe qu'il a choisi. Ceci fera apparaître son portrait dans l'annuaire, en le rendant visible aux yeux des autres connectés.

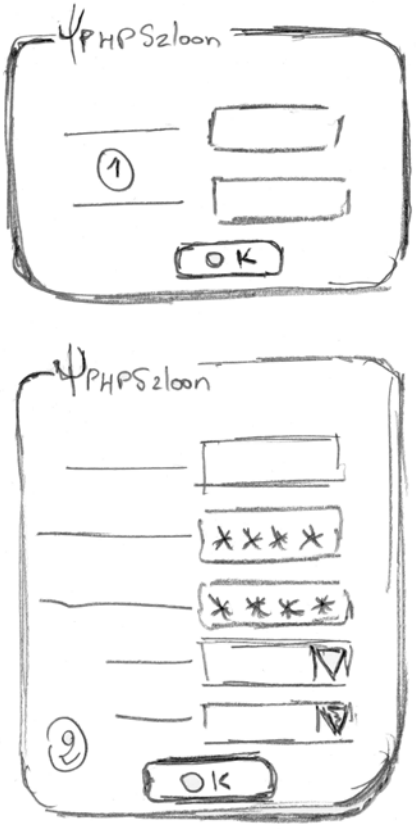


Figure 1–2 Inscription et identification dans PHP Saloon

Figure 1–3 Maquette de la page d'identification de PHP Saloon

Un tableau de bord en trois morceaux

Une fois connecté, le visiteur doit pouvoir accéder à trois groupes de fonctions :

- parcours de l'annuaire des connectés ;
- parcours de la liste des amis ;
- lecture et écriture des messages.

Ces trois fonctions pourront être accessibles sur une même page, si le client est un navigateur traditionnel (la figure 1-4 propose un croquis d'une présentation possible), ou présentées séparément, par exemple sur un téléphone mobile.

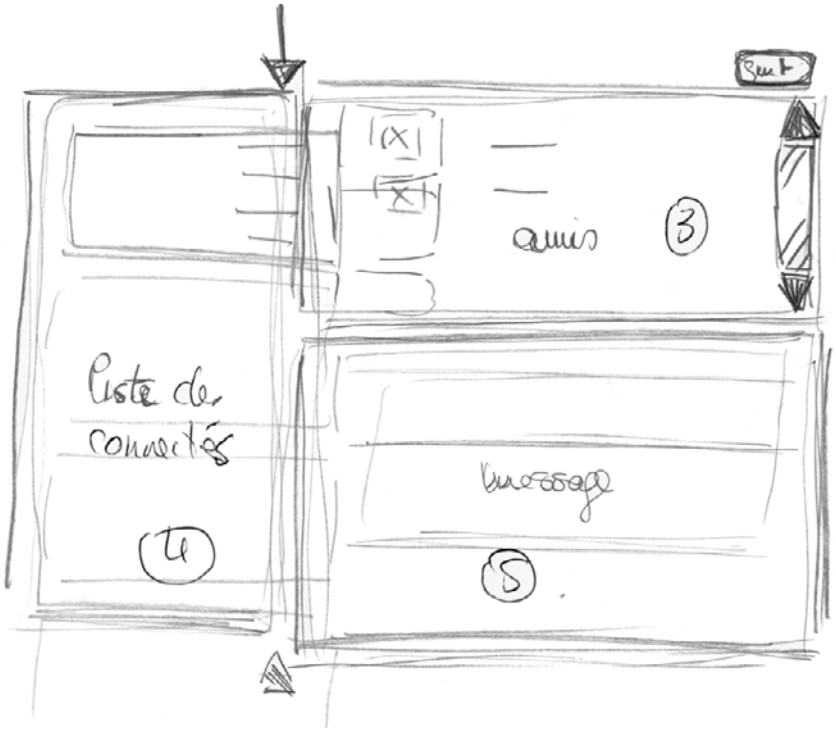


Figure 1–4
Exemple de tableau de bord pour PHP Saloon

Pourquoi choisir PHP 5 ?

La conception de notre application logicielle doit tenir compte de possibles évolutions futures, chose quasiment inévitable dans le domaine du logiciel. Il faut tenir compte des contraintes de maintenance, d'évolution et d'exploitation.

Dans une application professionnelle qu'on gérera au quotidien, il est alors essentiel d'optimiser l'entretien au long cours – un coût considérable, même pour un site personnel !

📖 J.-L. Bénard *et al.*, *L'Extreme Programming*, Eyrolles 2002.

RAPPEL **Répartition de l'effort dans un projet informatique**

Pourquoi tant d'attention portée à la maintenance et l'exploitation de notre application ? Tout simplement parce qu'il s'agit de la phase la plus coûteuse dans la vie d'un logiciel.

Les statistiques ci-dessous présentent la répartition du temps passé sur la vie du logiciel et plus précisément sur la phase de développement. Elles reprennent des valeurs classiquement admises dans l'industrie du logiciel, mais ne nous leurrons pas, cet état de fait s'applique aussi à votre site web personnel… Qui n'a pas passé des soirées entières à modifier son site, à en changer un détail ou finalement à en refaire des pans entiers…

Ce qui, pour un site personnel, peut émousser l'attrait du Web et décourager en laissant la technique prendre le pas sur la motivation de départ a naturellement un impact plus important en entreprise, mais le fond du problème reste identique.

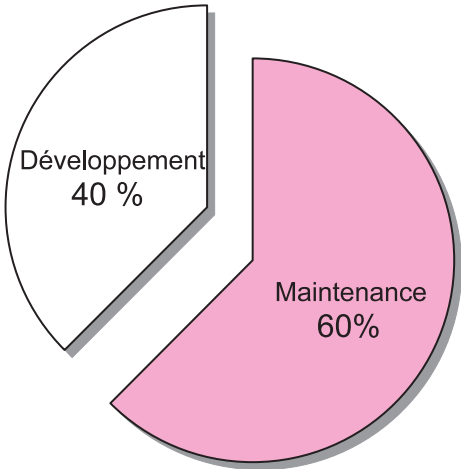


Figure 1–5 Répartition du temps entre la phase de création et celle de maintenance

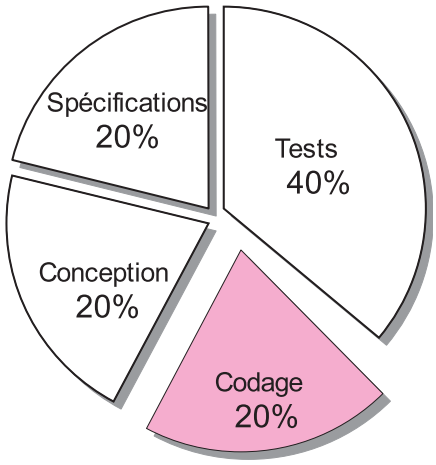


Figure 1–6 Répartition du temps passé lors du développement d'un logiciel

PHP 4

Dans cette optique, PHP, qui n'a pas encore 10 ans, paraît bien jeune et souffre de nombreuses lacunes. Ces lacunes empêchent les développements et limitent la possibilité de créer des applications extensibles et faciles à maintenir.

Aller vers un modèle à composants

Si PHP 4 permet en apparence de créer des objets, il n'en connaît en réalité pas vraiment la notion, et se contente de voir ces objets comme des tableaux associatifs banals. Cette vision des choses revient à évacuer la plupart des propriétés du modèle objet, pour ne retenir finalement que la notion d'agrégat d'informations (attributs ou méthodes). À ceci, il faut ajouter la contre-performance liée aux opérations de copie de ces tableaux.

Ainsi, faute de modèle objet efficace, un très grand nombre d'applications PHP ont fait l'impasse sur ce concept. Est-ce un drame ? Non bien sûr, mais loin de vouloir faire de l'objet pour de l'objet, ce modèle, complété par une prise en compte des composants, nous aurait permis de réaliser une implantation beaucoup plus modulaire et réutilisable dans d'autres applications, et facile à entretenir sur la durée.

COMPRENDRE **L'histoire à grande vitesse de PHP**

PHP est un jeune langage. Il tire son origine d'un langage moins évolué : PHP/FI créé en 1995 par Rasmus Lerdorf. Comme dans beaucoup de cas, Rasmus a d'abord développé PHP/FI pour son usage personnel (d'où le nom : Personal Home Page, Form Interpretor) sous la forme d'une bibliothèque de fonctions Perl, puis d'une véritable implantation en C. En 1997, pour sa version 2, PHP/FI connaît un succès d'estime non négligeable, mais reste un projet personnel.

Au même moment, ce sont deux utilisateurs de PHP/FI, Zeev Suraski et Andi Gutmans qui, mécontents des performances, vont réécrire PHP/FI et ainsi donner naissance à PHP 3.0, la première version véritable du langage tel que nous le connaissons. Ceci sonnera le glas de PHP/FI qui laissera place à son successeur PHP (dont le sigle est désormais synonyme de « PHP : Hypertext Processeur »).

Robuste, et doté de capacités d'extensions, PHP 3.0, officiellement publié en juin 1998, aura conquis en un an plus de 10% des serveurs web du monde.

Ce laps de temps a permis aux deux créateurs de repenser le cœur de l'interpréteur PHP et de créer l'entreprise Zend (une contraction de Zeev et Andy) et son développement Open Source phare : le Zend Engine. Ce moteur amélioré animera PHP 4.0, officiellement annoncé en 2000, nouvelle version plus performante de PHP, intégrant encore plus d'extensions et le support de plusieurs serveurs web.

L'histoire ne s'est pas arrêtée là, et l'arrivée du Zend Engine 2 permet à PHP dans sa version 5 de continuer plus avant sur le chemin des performances et de l'enrichissement fonctionnel, et surtout d'entrer de plain-pied dans l'univers des langages objet.

- http://www.php.net/manual/fr/history.php
- http://www.zend.com/

Améliorer la gestion des erreurs

Autre faille de notre application actuelle : la gestion des erreurs. En effet, PHP 4 ne propose pas de mécanisme particulier en la matière. Les erreurs sont détectées et reportées en utilisant des codes de retours.

Un tel système devient très vite cauchemardesque et, au final, pour ne pas compliquer inutilement le code par des tests incessants, on en arrive toujours à ignorer çà et là telle ou telle possibilité d'échec.

Les échecs ne manqueront pas de survenir lors de l'utilisation de l'application, et faute d'un traitement approprié, ils laisseront le visiteur bien perplexe devant une page illisible où s'enchevêtrent contenu et messages d'erreurs PHP.

Améliorer le support XSL

Dernier élément, l'utilisation d'XML et le rendu à base de transformations XSL complexes à utiliser pour des performances par ailleurs très moyennes... Pourtant ce procédé permet de séparer le traitement de l'information de son rendu pour différents médias (navigateur, téléphone) et améliore sensiblement la lisibilité du code développé.

On peut naturellement laisser au navigateur le soin de réaliser la transformation XSL. C'est une simplification importante qui évite les affres de la transformation avec PHP 4 mais il s'agit surtout d'une limitation considérable. Seuls Mozilla et Internet Explorer 6 sont en mesure de réaliser de telles transformations.

On perd donc l'intérêt de ces transformations qui permettent en théorie de produire simplement des versions du site pour plusieurs médias (navigateur, assistant personnel ou téléphone mobile par exemple).

Adopter PHP 5 pour conserver PHP

L'ensemble des limitations qui viennent d'être évoquées pourrait nous amener à nous interroger sur notre choix. PHP est-il finalement le plus adapté ? Ce serait oublier un peu vite les qualités intrinsèques du langage, qualités désormais dopées par les améliorations de la version 5. Et quelles améliorations !

PHP, un environnement simple

Parmi les raisons le plus souvent avancées pour justifier l'abandon d'une application, ou de son propre site personnel, l'exploitation quotidienne est certainement la plus importante. Dans cette optique, une des erreurs courantes consiste à déployer des outils qu'on ne saura pas maîtriser.

B.A.-BA **Le principe KISS**

KISS signifie en anglais : *keep it simple, stupid*. Il s'agit d'une devise initialement adressée aux développeurs et qui constitue un véritable éloge de la simplicité. Ce principe, parmi les plus connus du monde Unix et Internet, consiste à toujours rechercher la simplicité même si cela conduit à ignorer, parfois temporairement, des innovations. Ce principe a guidé la définition des protocoles web et Internet qui, à défaut d'être révolutionnaires, sont simples à mettre en œuvre et plus économiques que des concurrents a priori plus sophistiqués ou plus évolués. PHP illustre parfaitement ce principe : rien d'exceptionnel ni compliqué, juste un langage fonctionnel qui fournit le résultat attendu.

Tout le monde peut en être victime ; jour après jour, on tente de nous démontrer à force de publicité et d'arguments imparables le caractère indispensable de solutions toutes plus merveilleuses les unes que les autres. Il faut savoir résister.

Une des qualités essentielles de PHP est sa simplicité de mise en œuvre. Ce n'est pas un hasard si PHP est le langage retenu en priorité par les hébergeurs. Il offre tout simplement une solution robuste et fiable.

Dans un autre domaine, le succès grandissant des blogs et des wikis montre lui aussi que la réponse à un besoin donné ne passe pas nécessairement par des solutions complexes.

C'est pourquoi, indépendamment des manques réels dans PHP, y compris dans la version 5 à laquelle cet ouvrage est consacré, PHP est, dans une large majorité de cas, la solution la plus économique et la plus simple à maîtriser pour mettre en œuvre une application ou un service web.

Aussi, avant de choisir l'environnement qui motorisera votre application, posez-vous d'emblée les bonnes questions : saurai-je maintenir l'outil jour après jour ? Ai-je les moyens de déléguer à des tiers ? S'il s'agit d'un site personnel, avez-vous l'envie, le temps pour entretenir une solution complexe ?

CULTURE **Blog & Wiki**

Un blog (raccourci de WebLog) est une sorte de journal personnel publié sur le Web. C'est une réponse technique simple apportée à un besoin d'expression. Nulle question, dans un blog, d'HTML, balises ou autres. Chaque auteur peut publier jour après jour ses billets. Naturellement, leur contenu varie du carnet intime au journal thématique, en fonction de l'auteur.

Quant au Wiki, il offre la possibilité de créer du contenu issu de la collaboration de plusieurs auteurs. Chacun est libre d'enrichir les pages du site – ce qui a permis la naissance de plusieurs projets d'encyclopédies sous forme de wikis. Tout un chacun peut le compléter en fonction de ses connaissances et de ses découvertes.

Là encore, on mise sur la simplicité : quelques raccourcis typographiques et surtout pas de HTML. Dans les deux cas, ce sont des solutions simples, techniquement parlant, qui sont plébiscitées et répondent aux souhaits des utilisateurs. Plusieurs implantations existent en PHP.

REGARD DU DÉVELOPPEUR **PHP 5, regrets et espoirs**

La nouvelle mouture de PHP 5 apporte une couleur professionnelle à PHP, cela représente déjà un effort de développement important pris en charge par les concepteurs et développeurs de PHP. Naturellement, chacun voudrait voir **sa** fonctionnalité présente. Mais il faut à un moment donné savoir fixer le périmètre des fonctionnalités qui seront intégrées dans une version. Cela impose des choix, et forcément on peut toujours avoir certains regrets.

Au chapitre regret justement, on pourra évoquer les namespaces qui après quelques tests ne seront pas intégrés à PHP 5.

Les namespaces peuvent être décrits grossièrement comme des containers. Ils permettent d'englober tout le code d'un module. Les éléments du module sont alors accessibles en préfixant leur nom par celui du container (par exemple : `mysql::connect` en utilisant la notation `::` classique dans le monde objet). Les risques de conflits entres modules sont donc éliminés.

Comme nous pouvons le constater, les namespaces pourraient être un outil fantastique pour doper la création de composants réutilisables en PHP, ceci indépendamment du modèle objet.

Toujours au chapitre composant, faute de namespaces peut-être, PHP ne permet pas d'importer facilement les composants et a fortiori une hiérarchie de composants, comme le permet par exemple Java avec l'instruction `import`. Il faut soi-même, à la main si l'on peut dire, inclure les différents fichiers.

Par ailleurs, PHP ne permet pas d'inclure des fichiers déjà compilés pour l'interpréteur PHP ou mieux, des archives de fichiers compilés (naturellement le parallèle Java est le JAR).

Un langage glu

De nombreuses interrogations persistent au niveau du langage même. Mais si PHP n'est ni Java, ni C++ ni Python, vous restez en mesure de le maîtriser, et sinon, de nombreux collègues de l'entreprise seront à même de le faire.

PHP correspond ainsi très exactement à la définition d'un langage glu. C'est à notre sens l'une des raisons essentielles de son succès, comme cela fut le cas pour Visual Basic. Il met à la portée de tous la réalisation d'applications sur mesure.

Naturellement, l'écriture des extensions proposées par PHP reste l'affaire de spécialistes, mais PHP va nous permettre d'en tirer profit.

B.A.-BA **Qu'est-ce qu'un langage glu ? L'exemple de Tcl**

John Ousterhout, créateur du langage Tcl, a illustré une des qualités essentielles que partagent autant Tcl que PHP ou encore Visual Basic, en utilisant le terme de langage glu. Un langage glu est un langage simple à apprendre et à utiliser, qui va servir de socle (de ciment) pour créer des applications personnalisées en combinant des composants techniques, souvent complexes, écrits dans d'autres langages, en général mieux adaptés aux ingénieurs et informaticiens chevronnés.

Rappelons qu'une définition rapide de Tcl pourrait être : le PHP de l'ère pré-Internet. Tcl est un langage interprété simple, qui comme PHP dispose d'une bibliothèque impressionnante d'extensions. Dans le cas de Tcl, le client est un client lourd avec une interface construite en Tk (une bibliothèque graphique, qui dans notre analogie avec PHP, va jouer le rôle d'HTML). Tcl a ainsi été très largement utilisé pour créer des applications client/serveur graphiques en un tour de main, en particulier utilisées par les administrateurs réseau.

Tcl est encore largement utilisé aujourd'hui, ainsi un des outils d'administration de PostgreSQL, PgAccess, est développé en Tcl/Tk. PgAccess est ainsi disponible sur de multiples plates-formes, PC et Mac compris.

▸ http://www.tcl.tk/

Le modèle objet complet de PHP 5

PHP 5 tente de combler les lacunes encore présentes dans PHP 4 en intègrant un nouvel interpréteur : la version 2 du moteur Zend (le fameux Zend Engine).

Ce nouvel interpréteur est plus rapide comme on pouvait s'y attendre et surtout, il rénove complètement le support des objets.

Désormais, PHP va réellement disposer d'un type objet jusqu'au cœur du moteur. S'il est traité de manière spécifique, c'est pour éviter les recopies de données inutiles en mémoire lors de la manipulation d'objets (les spécialistes reconnaîtront là l'usage des références) et pour intégrer les fonctionnalités clés du modèle objet jusqu'à présent absentes : gestion des constructeurs et destructeurs, champs privés et statiques…

EN COULISSE **Rouages internes de PHP : le Zend Engine**

Pour comprendre l'importance du Zend Engine, il est nécessaire d'avoir une idée du fonctionnement de PHP. Celui-ci est le plus souvent utilisé en collaboration avec un serveur web, mais peut aussi être utilisé seul. C'est pourquoi depuis sa version 4, PHP dispose d'une interface (SAPI) qui normalise les échanges entre PHP et le programme qui intègre PHP (Apache, IIS ou un logiciel tiers).

Dans le cas ci-après, lorsque le serveur web reçoit une requête 1 (la demande d'une page par exemple), il transmet le code de la page demandée à PHP en utilisant l'interface SAPI. Ce code est dirigé vers le compilateur à la volée (JIT : Just In Time) du Zend Engine 2.

Celui-ci va analyser le code, en vérifier la syntaxe et produire un équivalent codé (on parle de code objet) beaucoup plus rapide à manipuler. Ce code est transmis à l'interpréteur du Zend Engine 3. Le moteur va dérouler le code codé et ainsi produire le code (probablement HTML) résultat de l'exécution de notre page PHP.

Naturellement, si l'interpréteur sait exécuter le cœur du langage (boucle, traitements des variables), il va déléguer aux extensions de PHP le traitement des instructions qui les concernent 4, par exemple l'accès à une base de données, ou la création d'une image.

Une fois l'exécution du code terminée, le résultat produit est renvoyé au serveur web via l'interface SAPI 5… le serveur web renvoyant lui-même le résultat au navigateur 6.

Cet exemple très simple montre le rôle de chef d'orchestre du Zend Engine qui seul sait traiter le code PHP et faire appel aux extensions disponibles. La définition d'extensions est alors un jeu d'enfant puisqu'il suffit de consciencieusement respecter l'API prévu par le Zend Engine. Cette parfaite organisation explique très largement la multitude d'extensions présentes dans la distribution PHP.

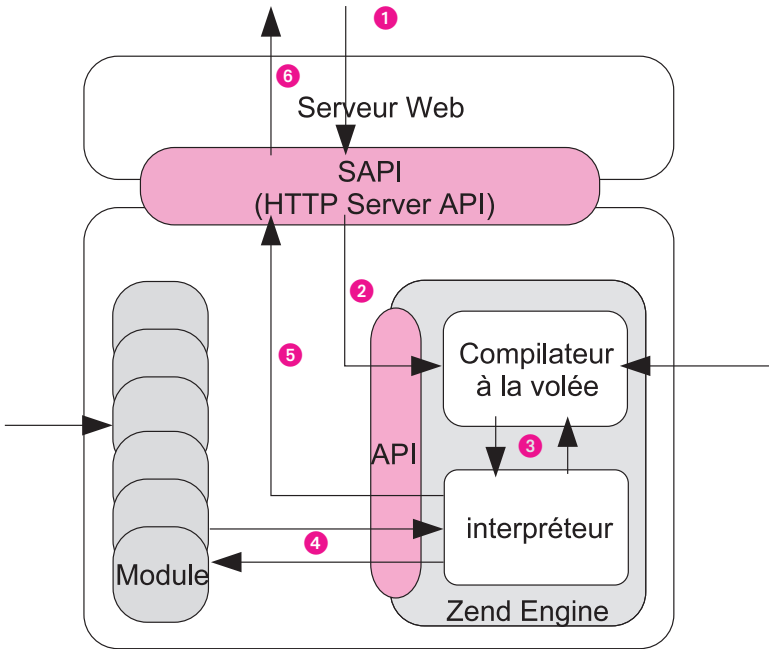


Figure 1–7
Architecture interne de PHP

Un nouvel ensemble de types prédéfinis

Suite logique de ce nouveau potentiel, PHP intègre désormais une extension regroupant tout un ensemble de classes modélisant des structures de données classiquement utilisées dans les algorithmes. Il s'agit par exemple des piles, des files d'attentes, des ensembles ou encore des arbres. On parle de types abstraits de données (en anglais Abstract Data Types ou ADT) pour désigner ces structures.

Là où le développeur devait réinventer la roue, comme d'ailleurs les auteurs de Pear ont dû le faire de leur côté, PHP propose un socle commun. Une base partagée qui devrait :

- simplifier le code développé, en évitant à tous de redévelopper des outils inclus et maintenus dans PHP ;
- mettre à disposition via ce socle commun un code de meilleure qualité.

En effet, plus un code est utilisé, plus on dispose de retours. Ces retours d'expérience permettent de résoudre les défauts et sont inaccessibles à une telle échelle au développeur isolé (même pour les applications d'entreprise).

B.A.-BA Pear (PHP Extension and Application Repository)

Pear est une bibliothèque de code Open Source pour PHP. Cette bibliothèque est organisée de la manière la plus rigoureuse possible.
Le code intégré doit respecter un guide de style spécifique et est ensuite classifié rigoureusement dans la bibliothèque.
Pour faciliter le travail des développeurs, Pear propose par ailleurs une série d'outils utiles lors des phases d'installation ou de maintenance.
Le développeur PHP a toujours intérêt à jeter un œil du côté de Pear avant de réinventer la roue, même s'il est vrai que le projet Pear, plus récent que PHP, n'a pas le même niveau de maturité et qu'il convient de s'assurer du niveau de stabilité des modules convoités.

- http://pear.php.net

Refonte du support XML/XSL

Enfin, PHP tire profit de la maturité des bibliothèques XML et XSL utilisées par le projet Gnome qui deviennent dans PHP 5 le moteur de référence utilisé pour traiter les fichiers XML.

Une décision qui devrait :

- améliorer les performances ;
- clarifier le fonctionnement de ces extensions.

Écrites en C, les bibliothèques choisies sont en effet sensiblement plus rapides que les diverses implantations disponibles par ailleurs.

Enfin, et c'est probablement le plus important après l'intégration d'un premier outil (Sablot), puis la modification des interfaces XSL, la pérennité semble enfin au rendez-vous. Un point essentiel si l'on souhaite voir l'usage d'XML et des transformations XSL largement adopté.

B.A.-BA Gnome

Gnome est un environnement de travail graphique complet pour les utilisateurs. Il est largement utilisé sous GNU/Linux et sous différents systèmes Unix, et une large partie de ses logiciels est aussi disponible sous Windows, notamment Gimp.

- http://www.gnome.org
- http://www.gimp.org

COMPRENDRE Démystifier XML/XSL

Démystifions tout de suite XML/XSL. Le manque de maturité des outils a pu faire croire que les langages en eux-mêmes, notamment pour XSL, étaient inaccessibles au commun des développeurs. Il n'en est rien. Il convient juste de bien appréhender leur logique et de ne pas l'oublier en cours de route. Nous verrons au chapitre 7 comment utiliser ces langages, dont il est difficile de faire l'impasse aujourd'hui pour des applications interopérables et réutilisables.

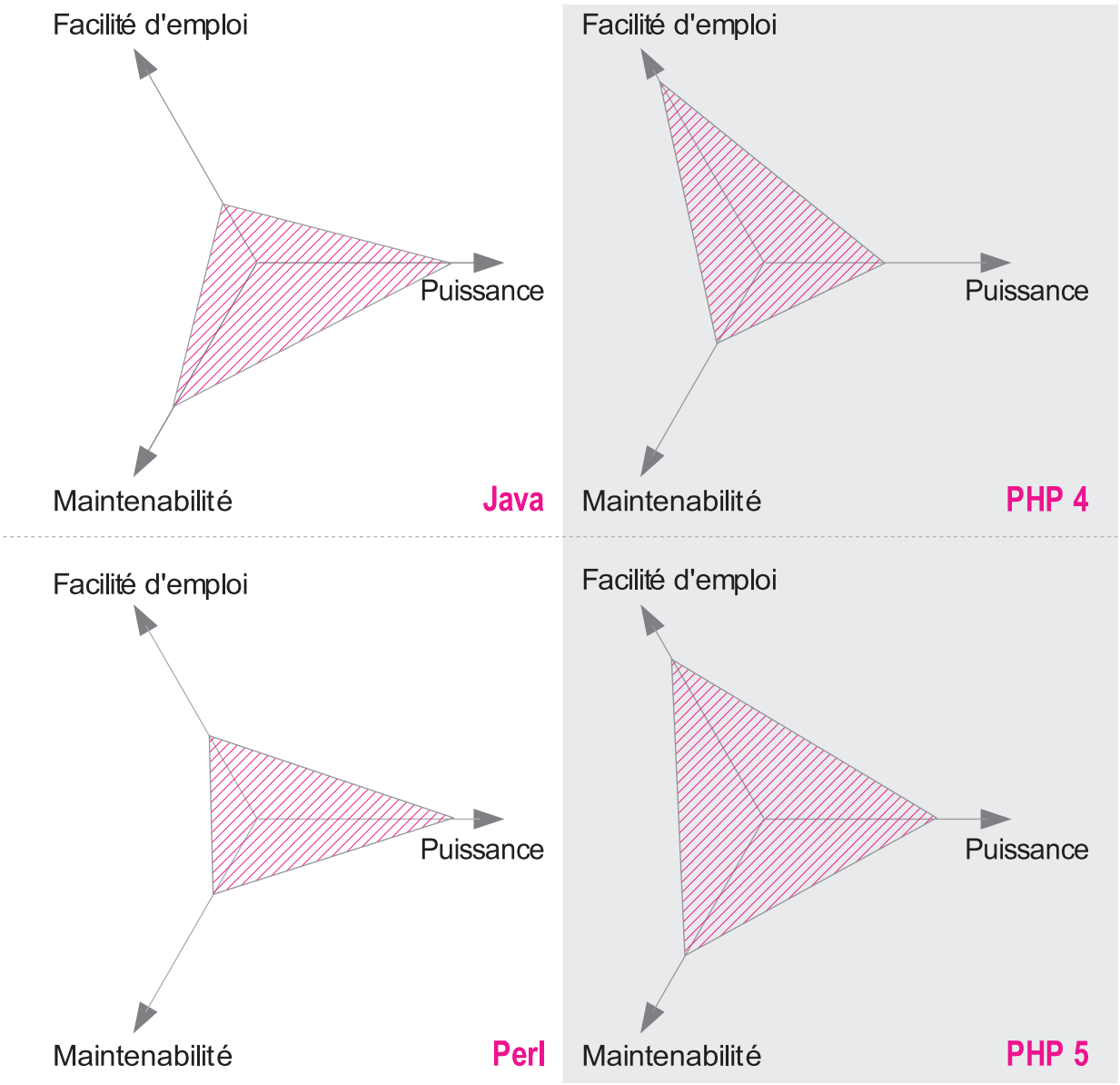


Figure 1–8 Ratios puissance, complexité et facilité d'emploi des solutions web les plus utilisées

En résumé…

Dans ce chapitre, nous avons conforté PHP, et naturellement PHP 5, en tant que langage de développement idéal pour notre type de projet. Ce choix est autant redevable de la simplicité du langage que de sa richesse fonctionnelle.

Nous avons de même défini précisément notre étude de cas, un point capital qui va maintenant nous permettre de tirer pleinement profit des nouvelles possibilités de PHP 5 et du modèle objet.

À RETENIR PHP n'est-il pas devenu trop complexe ?

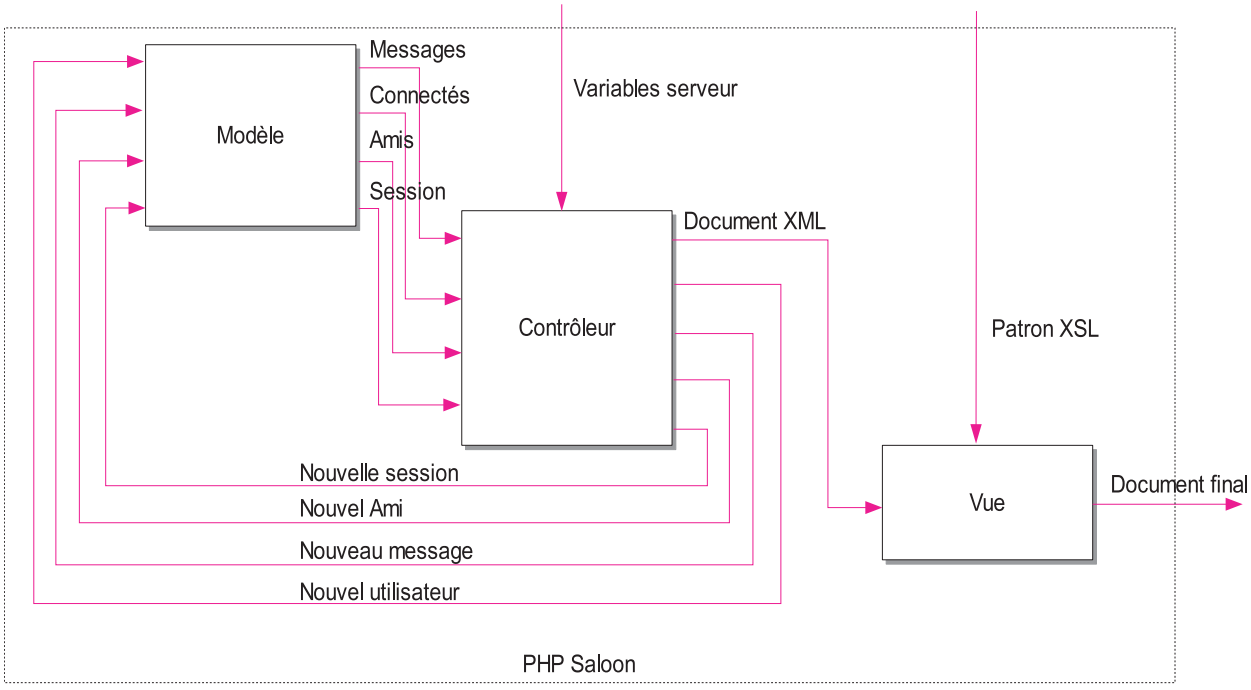
Nouveau modèle objet, fonctionnalités revues et corrigées, design patterns, PHP 5 ne perd-il pas l'essence même de ce qui a fait le succès du langage ? Il est vrai que l'avalanche de nouveautés peut effrayer, mais ne nous y trompons pas, le noyau dur du langage demeure inchangé. Libre à vous de n'utiliser que les fonctionnalités qui vous intéressent avec, encore et toujours, ce langage simple et efficace qu'est PHP.

À ce sujet, une comparaison rapide (et assez grossière) des différents langages adoptés pour le développement des applications web montre que si PHP, dans cette nouvelle mouture, offre des outils qui le rapprochent de ses concurrents les plus sophistiqués, il reste un langage très simple d'emploi et dépassant largement les solutions fondées sur Java qui nécessitent une réelle expertise des frameworks mis en œuvre (J2EE, BEA WebLogic…).

Une aptitude qui n'entrave pas pour autant PHP (a fortiori PHP 5) dont la puissance et les performances ont d'ores et déjà conquis les plus grands sites mondiaux comme le démontrent les retours d'expérience présentés sur le site de l'AFUP (Association française des utilisateurs de PHP).

- http://www.afup.org/

chapitre 2


Modèle
Messages
Connectés
Amis
Session
Variables serveur
Contrôleur
Document XML
Patron XSL
Vue
Document final
Nouvelle session
Nouvel Ami
Nouveau message
Nouvel utilisateur
PHP Saloon

Organisation et découpage du travail avec les interfaces

Avant d'écrire le code d'une application, il est sage de procéder à un découpage en sous-briques, plus simples à réaliser. C'est ce que nous allons faire pour PHP Saloon.
Le résultat de cette analyse sera réalisé avec d'autant plus de facilité que PHP 5 permet désormais de définir les interfaces de nos modules.

SOMMAIRE

MOTS-CLÉS

- MVC
- Interface
- XML
- Décomposition top-down

Premiers éléments de l'architecture logicielle

Les croquis rapidement réalisés du chapitre précédent donnent une idée du résultat souhaité pour PHP Saloon, mais il faut maintenant regarder sous cette apparence, et organiser le développement.

Pour nous simplifier la tâche, nous allons, en suivant une sorte d'approche « top-down », découper de plus en plus finement notre application et les éléments qui la composent. Au final, il nous restera des modules simples à coder (qui par ailleurs serviront chacun à illustrer les nouvelles capacités de PHP 5).

Nous n'allons pas réaliser le découpage de PHP Saloon au hasard mais en nous inspirant d'un modèle d'architecture : l'architecture MVC ou Modèle, Vue, Contrôleur. Cette architecture est très répandue et est notamment très en vogue dans le monde Java.

Ce type de découpage était jusqu'à présent assez délicat à réaliser en PHP. D'une part, le code PHP se trouve en général au beau milieu du code HTML, il n'est plus question alors de distinguer vue et contrôleur. D'autre part, la faiblesse du modèle objet disponible jusqu'à présent ne permettait pas vraiment de décrire les différents aspects, modèle, vue ou contrôleur.

En choisissant PHP 5 et XML comme noyau technologique pour PHP Saloon, nous adoptons au contraire les outils qui vont nous permettre de correctement organiser notre application :

- En utilisant XML (et les transformations XSL), nous allons séparer de fait le rendu (la vue) des différents traitements qui produisent le code XML (le contrôleur).
- Enfin, le modèle objet complet, disponible dans PHP, va nous permettre de spécifier les interfaces des modules qui composent PHP Saloon et ainsi d'identifier précisément les composants qui relèvent du modèle ou du contrôleur.

MÉTHODE Deux approches : bottom-up et top-down

Pour décrire une application, on peut procéder de deux manières.
La première est de définir les différents composants élémentaires, puis de les agréger progressivement les uns aux autres pour finalement aboutir à l'application finale. C'est l'approche bottom-up (du particulier au général).
La seconde, à l'inverse, consiste à d'abord considérer l'application dans son ensemble puis à la décomposer en modules de premier niveau, en redécoupant à nouveau pour arriver aux composants élémentaires. C'est l'approche top-down (du général au particulier).
Chaque approche a ses avantages et ses inconvénients, l'approche top-down offre une vision plus synthétique des développements, l'approche bottom-up permet de définir plus facilement des bibliothèques de composants réutilisables.

ALTERNATIVES Autres types d'architectures : .NET et JSF

L'architecture MVC est retenue par nombre de projets, c'est notamment le cas de Jakarta Struts qui met en œuvre MVC sur une plate-forme Java.
Cependant les alternatives existent, on peut notamment adopter un modèle événementiel, proche du modèle client/serveur. C'est le choix fait par .Net, l'environnement de Microsoft, mais aussi par les JSF (Java Server Faces), récemment proposé par le père de Java, Sun Microsystems.

MÉTHODE L'architecture MVC (modèle-vue-contrôleur)

Le modèle MVC est l'un des modèles de conception les plus répandus. Il consiste à séparer une application en trois composants distincts de manière à en faciliter la conception et la réalisation.

Le composant `modèle` décrit l'ensemble des données qui sont manipulées par l'application. Il peut s'agir d'objets ou d'éléments de base de données.

Le composant `vue` définit les mécanismes qui vont permettre de prendre en compte les actions de l'utilisateur et de représenter les données.

Enfin, le composant `controleur` gère la logique métier de l'application, l'ensemble des traitements sur les données. Il constitue une sorte d'interface entre la vue et le modèle.

Les flux d'information dans PHP Saloon

Pour définir précisément l'interface de nos trois composants de premier niveau, identifions les flux d'information qui circulent au sein de l'application. Pour la vue, les choses sont très simples. À partir d'un document XML et d'un patron XSL, la vue génère une sortie. Dans le cas général il s'agit de HTML, mais nous traiterons aussi le cas de formats comme le cHTML pour téléphones i-mode ou Wap.

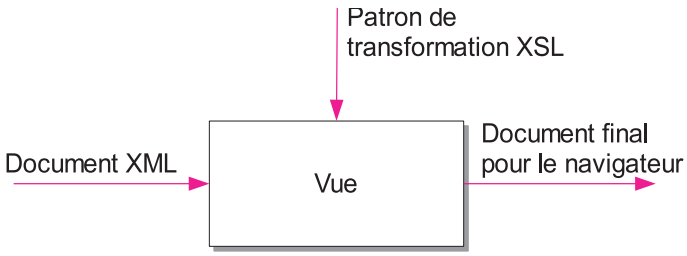
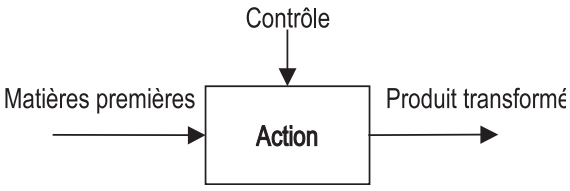


Figure 2–1 Flux d'information pour la vue

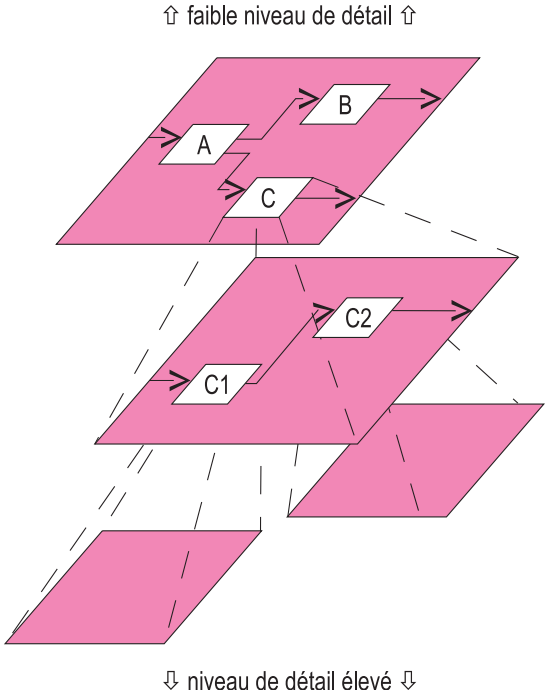
RAPPEL **SADT (Structured Analysis and Design Technique)**

SADT est une méthode très graphique dans sa présentation, les processus y sont décrits sous forme de rectangles et de flèches : les rectangles vont représenter les actions, comme afficher une carte, stocker les informations client ; les flèches vont matérialiser les flux d'information tels le nom du client, la position d'un aérodrome, etc. On respecte en général les conventions suivantes : à gauche, les « input » ou matières premières, à droite les « output » ou produits finis. En haut, les éléments de contrôle du système. Les diagrammes ainsi obtenus portent le nom d'actigrammes (en référence aux activités que représentent les boîtes).



Enfin, on part d'une vue d'ensemble, comportant quelques actions, pour aboutir à plusieurs décompositions en sous-systèmes, un peu à la manière des poupées russes.

La méthode SADT ne se limite naturellement pas à ces quelques éléments graphiques et s'attache à décrire l'intégralité des flux manipulés, qu'il s'agisse des traitements, comme dans le cas de nos actigrammes, ou des données. Enfin, SADT définit un cycle complet entre auteurs et relecteurs. Un processus qui prend tout son intérêt dans la relation client/prestataire.



Le contrôleur, de son côté, est le producteur du document XML qui est transformé par la vue. C'est un peu le cœur de PHP Saloon. Les échanges entre ce noyau central et le modèle sont donc naturellement plus complexes. En reconsidérant les croquis du chapitre 1, on peut cependant identifier différents flux :

- les échanges liés à l'identification et au contrôle de la session utilisateur ;
- la manipulation de la liste des abonnés ;
- la gestion de la liste des amis ;
- la manipulation des messages.

L'ensemble de ces flux d'information sont à destination ou à l'origine du modèle.

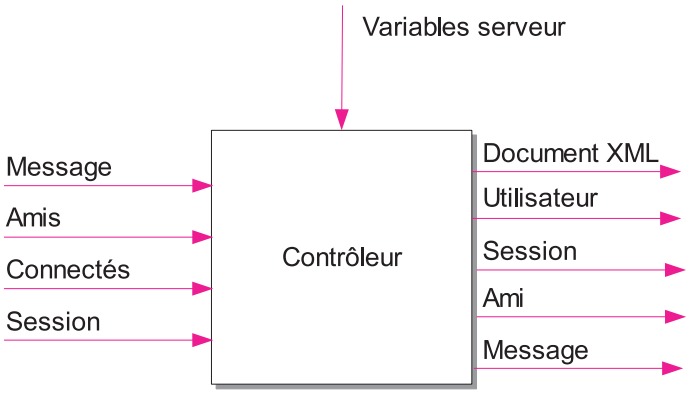


Figure 2–2 Flux d'information pour le contrôleur

L'identification des flux du modèle est élémentaire. Cela concerne exclusivement les échanges avec le contrôleur.

Figure 2–3 Flux d'information pour le modèle

En regroupant ces différents éléments, nous obtenons le datagramme SADT suivant qui va nous servir de référence pour définir nos interfaces.

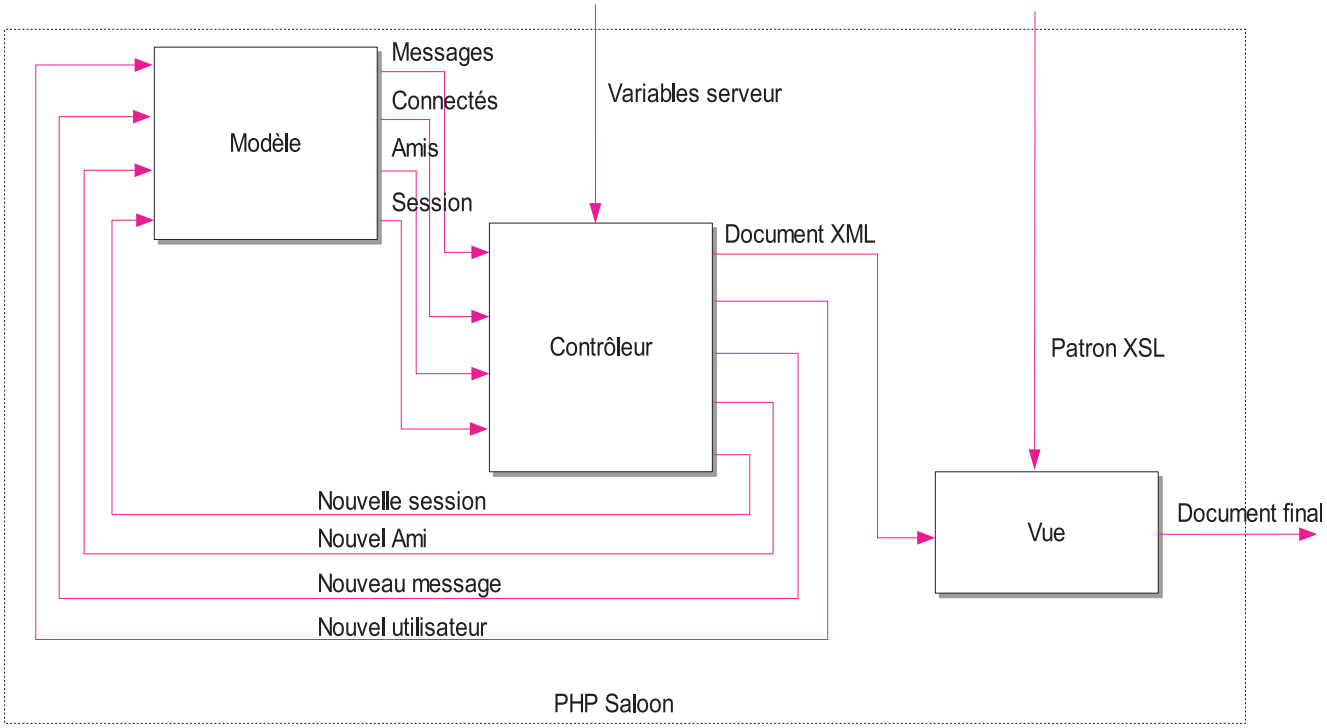


Figure 2–4 Décomposition SADT complète

Les interfaces vues par PHP

À partir du découpage précédent, nous pourrions tout à fait commencer à écrire le code des objets et des fonctions PHP. Mais ce serait prendre un risque important pour la durée de vie de notre application.

Rien dans ce cas ne nous permettrait d'assurer que le code réalisé est conforme à notre modèle. Pire, même en faisant abstraction des différences inévitables qui surgiront entre modèle et réalisation et en supposant que celles-ci ne gêneront pas le fonctionnement courant du logiciel, que se passera-t-il lors des prochaines évolutions ?

La notion d'interface, telle que l'introduit PHP 5, répond à ce problème. L'interface décrit les pré-requis d'une spécification, pré-requis auxquels toute implantation doit se conformer.

Ce pré-requis est donné en listant les prototypes des méthodes qui devront exister dans l'implantation.

Cet ensemble va constituer une sorte de contrat à respecter. Un objet ne sera déclaré conforme à une interface que s'il respecte tous les pré-requis. C'est-à-dire s'il comporte bien les différentes méthodes référencées par l'interface.

L'exemple suivant, très simple conceptuellement, va nous permettre de découvrir l'approche adoptée par PHP avant de nous lancer dans la définition effective des interfaces de PHP Saloon.

L'idée est de définir une interface à la notion de fichier (par exemple pour réaliser une implantation réseau compatible avec une version de référence classique). Les méthodes obligatoires imposées dans l'interface pourraient être :

- ouvrir ;
- fermer ;
- lire ;
- écrire.

Peu importe la nature exacte de l'implantation, tout fichier compatible devra offrir ces quatre méthodes. En PHP, on écrirait :

```
interface fichier {
   function ouvrir($nom);
   function fermer();
   function lire($nombre);
   function ecrire($donnees);
}
```

La syntaxe adoptée par PHP est élémentaire. Par la suite, lors de la réalisation effective d'un type d'objet fichier, on pourra préciser que celui-ci respecte notre interface comme suit :

```
class fichier_traditionnel implements fichier {
   function ouvrir($nom) {
      ...
   }
   function fermer() {
      ...
   }
   function lire($nombre) {
      ...
   }
   function ecrire($donnees) {
      ...
   }
}
```

On observe immédiatement le gain en lisibilité ; avant même de regarder le détail de la classe `fichier_traditionnel` on sait que celle-ci disposera au moins des fonctionnalités décrites dans l'interface `fichier`.

Naturellement, au-delà de ce gain immédiat, l'important est de savoir que tout type d'objet qui voudra implanter la spécification `fichier` devra disposer des méthodes ad hoc. Sinon, le code sera refusé par le moteur PHP avant même son exécution.

Un type d'objet peut implanter plusieurs interfaces différentes, on pouvait s'y attendre. En reprenant notre exemple, on peut définir une nouvelle interface :

```
interface manipulations {
   function copier($nouveaunom);
   function effacer();
   function deplacer($nouveaunom);
}
```

Et compte tenu de celle-ci, on pourrait imaginer une implantation de la classe `fichier` plus sophistiquée que la précédente, qui remplirait les conditions imposées par les deux interfaces `fichier` et `manipulation` :

```
class fichier_etendu implements fichier, manipulations {
   ... // définition de toutes les méthodes imposées par les
interfaces.
}
```

B.A.-BA **Prototype et signature**

Le prototype d'une fonction (le terme mathématique est signature) ne s'intéresse pas au code de la fonction. Il constitue lui-même une sorte d'interface, de spécification. Dans un prototype, on précise en reprenant le plus souvent la syntaxe du langage :

- le nom de la fonction ;
- les paramètres en entrée ;
- le résultat.

PHP n'etant pas typé, on se contente la plupart du temps de préciser le nombre des paramètres en les nommant (le nom utilisé pour le prototype ne sera pas nécessairement repris par l'implantation) :

```
function example ($param1, $param2);
```

Cependant il est possible de définir plus précisément la nature des paramètres en précisant le type d'objet attendu :

```
function exemple (Class1 $param1, Class2 $param2);
```

B.A.-BA **Objet, classe et méthodes**

Enfin si aucun paramètre n'est spécifié, PHP n'effectue aucune vérification. La fonction pourra alors comporter autant de paramètres souhaités.

Dans cet ouvrage, nous allons beaucoup évoquer les objets avec des termes qui peuvent parfois prêter à confusion.

Ainsi la notion de classe, qui peut paraître obscure, ne représente en réalité rien d'autre que l'extension aux objets de la notion de type.

Une classe, c'est donc la définition d'un type d'objet. Dans la classe, on précise la nature des données stockées (on parle d'attributs) et les fonctions disponibles pour manipuler l'objet (on parle de méthodes).

Une fois la classe définie, on peut alors effectivement créer les objets en tant que tels. L'opération de création porte le nom d'instanciation. Chaque objet représente une instance (un exemplaire) du type défini par la classe.

Le modèle objet ne se contente pas de reprendre la notion de type, il complète cette notion en assurant notamment la protection des données avec la possibilité de définir certains attributs ou méthodes comme éléments privés. Ceux-ci ne seront alors pas accessibles par des objets ou des fonctions tiers. Seules les méthodes de l'objet disposeront du droit d'accès.

B.A.-BA **Implantation ou implémentation**

Le terme implantation est courant, il désigne le code effectivement réalisé pour répondre à un besoin en général décrit dans un cahier des charges ou des spécifications.

L'important est de ne jamais oublier qu'une même demande peut donner lieu à un grand nombre d'implantations différentes. En effet, il existe bien plus d'une manière de réaliser un logiciel, et il est souvent souhaitable, voire nécessaire, de disposer d'implantations spécifiques (à un matériel ou pour les tests).

C'est là que l'usage des interfaces est crucial : il va permettre de normaliser les échanges entre les implantations de chaque module. Le respect de ces normes permet ensuite de modifier les implantations sans risques pour les autres modules.

À noter qu'on utilise souvent l'anglicisme équivalent : implémentation.

À ce stade, il est important de ne pas mélanger interface et héritage multiple classique. Les interfaces ne sont pas des classes au sens commun du terme. Leur rôle ne relève pas du domaine de l'implantation. Leur vocation est de décrire et de normaliser une interface de programmation.

Compte tenu des éléments précédents, une crainte légitime quant aux interfaces est de s'interroger sur l'évolutivité du code ainsi réalisé. Nous sommes à même de nous poser des questions. Pourra-t-on le réutiliser ? Comment le faire évoluer ?

Dans les deux cas, derrière une apparente rigidité, les interfaces constituent au contraire un facteur de dynamisme. En ce qui concerne la réutilisation, la présence d'une interface claire permet de mieux déterminer la correspondance entre les besoins et le type d'objet convoité, ce qui simplifie l'identification des objets réutilisables.

Enfin, les interfaces peuvent elles-mêmes être étendues par héritage, exactement comme les objets eux-mêmes. Il est donc possible de faire évoluer en parallèle les objets et les interfaces et proposer ainsi simultanément plusieurs versions de bibliothèques de composants.

```
interface manipulations_etendues extends manipulations {
   function compresse();
}
```

On le voit, les interfaces introduites dans PHP 5 apportent une réelle amélioration en matière de développement logiciel. Dans PHP Saloon, ces interfaces vont nous permettre de figer les interactions entre nos modules et ainsi clarifier les échanges, puis permettre de tester des implantations différentes.

La vue

Compte tenu de la décomposition déjà réalisée, on constate que le seul rôle de la vue dans PHP Saloon est de transformer le document XML, d'où l'interface PHP suivante :

```
interface iVue {
   function transforme($document, $style);
}
```

Voilà, nous venons de définir la première interface de PHP Saloon. Pour le moment, la nature exacte des paramètres est encore inconnue. Le moment venu, nous préciserons le type de ces paramètres.

Le contrôleur

Le cas du contrôleur n'est guère plus complexe ; en effet, la plupart des flux d'information que nous avons détectés précédemment sont liés au modèle. Le contrôleur se contente de réaliser les traitements sur les données de celui-ci.

La seule méthode indispensable d'un contrôleur est celle qui est responsable de la production du document XML. Nous pouvons donc proposer l'interface :

```
interface iControleur {
   function genere();
}
```

En fonction des pages appelées par le visiteur, le contrôleur réalisera des opérations d'agrégations sur des données plus ou moins compliquées, mais en réalité, du point de vue extérieur, seule l'opération de génération est importante.

Nous pourrions donc en rester là, cependant, et bien que notre interface soit tout à fait valide, nous allons adopter une version légèrement modifiée :

```
interface iControleur {
   function saveXML();
}
```

Cette dernière mouture anglophone nous permettra, par la suite, de tirer profit d'objets PHP disponibles en standard.

Le modèle

Le cas du modèle est plus complexe. C'est en effet lui qui va nous fournir le moyen d'accéder aux données. Au lieu de définir une seule interface globale, nous allons adopter une approche plus modulaire et définir pour chaque type de données manipulées une interface propre.

Les données de session

L'objet session en lui-même est élémentaire. Son rôle va consister à gérer un compteur de temps. Lorsque l'utilisateur est actif, le compteur est mis à jour et chaque page de PHP Saloon teste la date de dernière mise à jour via notre objet session, pour décider de l'expiration ou non de la session en cours.

Deux méthodes doivent donc être implantées pour tout objet session :

- la mise à jour du compteur de session pour préserver sa validité ;
- le test d'expiration.

On peut donc proposer l'interface suivante :

```
interface iSession {
function update();
function active();
}
```

Les listes d'information

Outre les sessions, PHP Saloon manipule de nombreuses listes, parmi lesquelles, les listes de connectés et les listes d'amis. On peut donc logiquement espérer pouvoir définir une interface commune à ces listes.

En réalité, il est même possible d'aller plus loin en utilisant les conclusions proposées pour un design pattern très classique : l'itérateur (*iterator*).

Le design pattern itérateur décrit des situations où il est nécessaire de parcourir une suite d'éléments (par exemple nos listes de connectés). C'est un cas très courant dans le développement logiciel.

MÉTHODE Design patterns

Les design patterns sont d'abord une preuve de bon sens. Ils reposent sur un constat simple : même si chaque logiciel est spécifique, une très large partie des problèmes conceptuels rencontrés lors du développement se retrouve d'un projet à un autre. L'approche des design patterns vient compléter la notion de bibliothèque, qui est plus adaptée aux problèmes techniques purs et peu transposable à la résolution de problèmes d'architecture logicielle.

Sans aller jusqu'à proposer des solutions techniques prêtes à l'emploi, les design patterns offrent pour chaque situation une analyse précise et une proposition de modélisation – on pourrait presque dire, un guide de développement.

On peut décrire autant de design patterns que l'on souhaite, chacun répondant à une situation donnée, cependant il existe un jeu de design patterns très usuels, ceux-ci sont en général décrits dans les ouvrages consacrés au sujet.

Chaque projet peut ainsi tirer profit de l'analyse du problème qu'ont déjà réalisée les meilleurs spécialistes. Il est important d'avoir toujours en tête le concept du design pattern avec PHP 5, car ceux-ci ont très largement inspiré les développeurs.

Il propose une interface type que nous allons tout simplement reprendre.

```
interface Iterator implements Traversable


{
     function rewind();




     function current();


     function key();



     function next();

     function valid();
}
```

L'extension SPL définit une interface abstraite, `traversable`, dont dérivent toutes les interfaces du type itérateur.

Lors du parcours d'un objet `traversable` un pointeur permet de connaître à chaque instant la position courante parmi la séquence d'éléments. `Rewind()` repositionne le pointeur vers le tout premier élément.

`Current()` renvoie l'élément courant et incrémente le pointeur de sorte qu'il désigne l'élément suivant dans la séquence.

`Key()` désigne une clé d'accès à l'élément. L'interface ne dit rien quant à la nature de la clé, chacun est libre d'implanter le concept comme il le souhaite.

`Next()` incrémente le pointeur et retourne l'élément associé.

Appelée `hasmore()` en phase de développement cette extension permet de savoir si la fin de la séquence est atteinte ou s'il reste des éléments à parcourir.

L'interface retenue est très classique. Dans le cas de PHP Saloon, les éléments de la liste seront des connectés au chat.

L'adoption de cette interface recèle par ailleurs un avantage particulier. En effet, PHP lui-même utilise cette interface pour certaines opérations, c'est le cas notamment de l'instruction `foreach`.

En utilisant l'extension SPL, nous allons pouvoir directement utiliser l'instruction `foreach` sur nos listes au lieu de recourir nous-mêmes aux méthodes `current()`, `next()` et `valid()`... Une simplification syntaxique importante et surtout un gain en lisibilité appréciable vont en découler.

Les messages

L'utilisation de l'interface `iterator` va aussi apporter son lot de simplification à la gestion des messages. Celle-ci comporte trois actions-clés :

- la transmission effective d'un message à un interlocuteur ;
- la création du formulaire d'envoi ;
- l'affichage du fil de discussion en cours.

Les deux premières étapes nous permettent de proposer une interface partielle du type :

```
interface iMessage {
   function nouveau(); // création du formulaire
   function transmet(); // sauvegarde du message envoyé
}
```

Pour le fil de la discussion, on retrouve le désormais classique parcours d'une liste (ici des différents messages échangés). On peut donc tout à fait reprendre l'interface `iterator`. L'objet contrôleur qui créera la page au moment venu n'en sera que plus simple.

On aboutit donc à l'interface :

```
interface iMessage extends iterator {
   function nouveau();
   function transmet();
}
```

REGARD DU DÉVELOPPEUR **L'extension SPL**

L'extension SPL écrite par Marcus Boerger, contributeur majeur à PHP, ne se borne pas à définir une interface pour le design pattern `iterator` mais s'applique à tout un jeu de patterns classiques.

De plus, dans cette optique, le code de l'interpréteur PHP lui-même a été revu pour tirer profit de ses interfaces quand cela est possible. C'est le cas notamment pour l'instruction foreach.

`Foreach` opère normalement exclusivement sur les tableaux, c'est notamment le cas dans PHP 4.

```
$tableau = array ( 1, 2 , 3);
foreach ($tableau as $element) {
   print $element;
}
```

Cependant, on peut constater sans difficulté que les fonctions proposées dans l'interface `iterator` suffisent à parcourir le tableau du début à la fin.

Dans PHP 5, le code de `foreach` a donc été réécrit de manière à ne requérir que les fonctions de l'interface `iterator`. Il est alors possible d'utiliser `foreach` sur tous les objets qui implantent cette interface.

Cette même logique s'est appliquée à la gestion des tableaux dans PHP 5. Pour peu qu'il nous faille respecter une interface ad hoc, toute classe pourra utiliser la notation [] auparavant réservée aux seuls tableaux PHP.

Pratiquement, l'extension SPL est partie intégrante de PHP 5 et devrait être activée par défaut dans toutes les versions. Si vous devez compiler vous-même votre interpréteur PHP, il est possible de s'assurer de la chose en utilisant l'option `–enable-spl` lors de la phase de configuration.

L"activation de l'extension peut être vérifiée en utilisant le traditionnel `phpinfo()`, une section dédiée à SPL est alors présente :

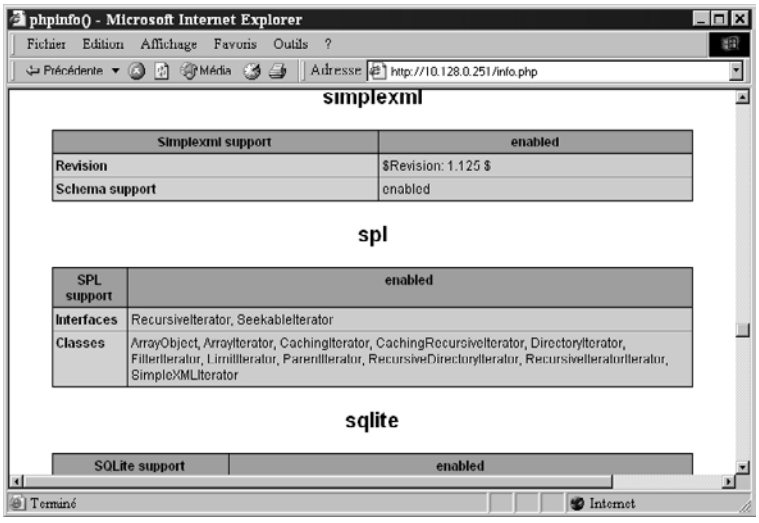


Figure 2–5 Activation de l'extension SPL dans PHP 5

En résumé...

Nous avons découpé notre application selon le modèle MVC et défini les interfaces nécessaires pour baliser ce découpage :

```
interface iVue {
   function transforme($document, $style);
}
interface iControleur {
   function saveXML();
}
interface iSession {
   function update();
   function active();
}
interface iMessage extends iterator {
   function nouveau();
   function transmet();
}
```

Génération du document final (html)

Sauvegarde du document DOM en XML

Mise à jour du compteur de session

Test de l'ancienneté de ce compteur

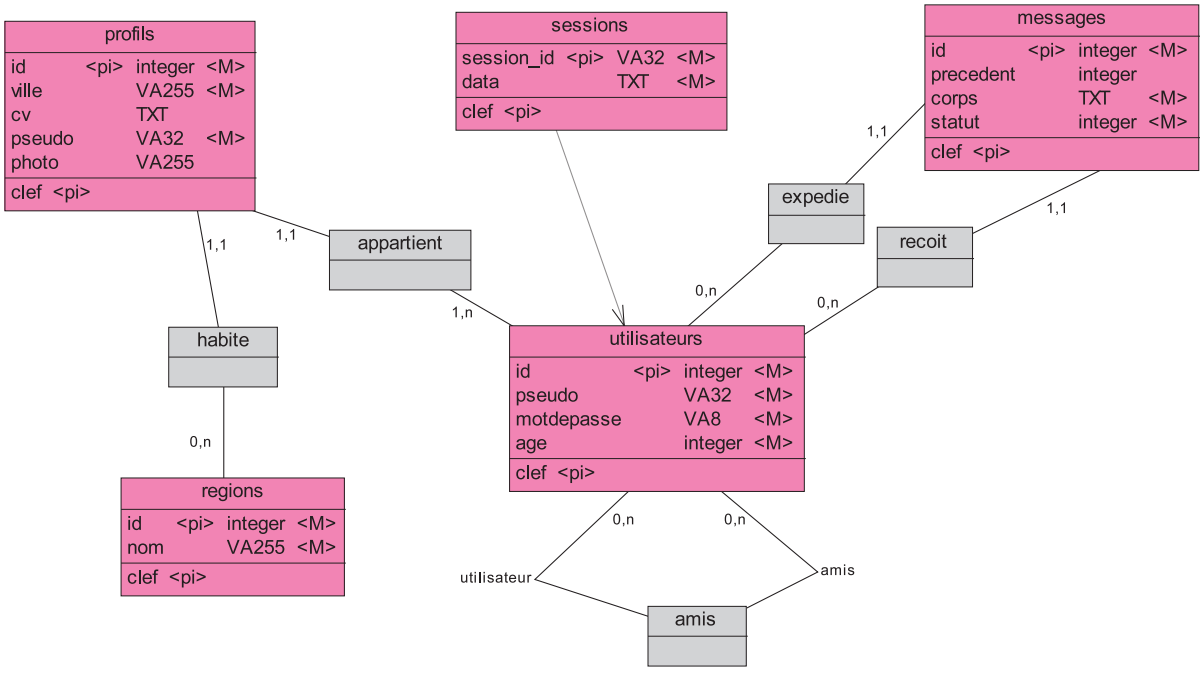
Création du code XML pour un formulaire

Transmission d'un nouveau message

L'ensemble de nos listes de connectés respectera l'interface `iterator`.

Le périmètre à respecter par l'implantation est donc désormais déterminé. Il serait donc maintenant tout à fait possible de confier le développement de chaque module aux différents membres d'une équipe. Mais la définition de ces interfaces permettra aussi aux plus téméraires de réaliser sans risques des implantations alternatives à celles que nous allons maintenant réaliser.

chapitre 3


profils
id <pi> integer <M>
ville VA255 <M>
cv TXT
pseudo VA32 <M>
photo VA255
clef <pi>
sessions
session_id <pi> VA32 <M>
data TXT <M>
clef <pi>
messages
id <pi> integer <M>
precedent integer
corps TXT <M>
statut integer <M>
clef <pi>
appartient
expedie
recoit
habite
utilisateurs
id <pi> integer <M>
pseudo VA32 <M>
motdepasse VA8 <M>
age integer <M>
clef <pi>
regions
id <pi> integer <M>
nom VA255 <M>
clef <pi>
amis
utilisateur
amis
1,1
1,1
1,1
1,1
1,n
0,n
0,n
0,n
0,n
0,n

Modèle de données avec SQLite

Une application Internet est rarement créée pour un seul utilisateur. Il faut donc assurer la sauvegarde et l'accès partagé à l'information.
C'est l'objet d'un système de gestion de base de données tel que SQLite que nous utiliserons pour stocker l'ensemble des données manipulées dans PHP Saloon.

SOMMAIRE

MOTS-CLÉS

- Entité/relation
- Modèle conceptuel
- Modèle physique
- SQL
- Contraintes d'intégrité

Construire un modèle de données, c'est mettre à plat et décrire toutes les informations qui devront être conservées, même après l'exécution d'un programme. Ce travail est indispensable quel que soit le langage de développement, même s'il est objet !

Un modèle de données : pour quoi faire ?

Nous venons de le voir, PHP 5 permet de créer des objets, ce sont eux que nous manipulons dans PHP Saloon, notamment au niveau du contrôleur.

Cependant ces objets :

1. ne sont pas partagés entre les différentes instances des programmes PHP ;
2. sont détruits à la fin du processus de création de la page web (qui coïncide avec la fin du programme PHP).

Cela signifie qu'avec les seuls objets PHP, rien ne survit à la page en cours (ni les messages, ni les inscriptions). On pourrait naturellement prendre soin de stocker manuellement ces objets dans des fichiers. Il serait alors possible de les récupérer à chaque début de script PHP.

Une telle méthode répond à la première objection mais laisse la seconde en suspens. En effet, comment partager sans risques ces informations quand plusieurs utilisateurs vont vouloir consulter ou modifier les objets stockés dans nos fichiers ?

Le problème décrit ici est celui de l'accès concurrent à l'information. Contrairement à ce qui est le plus souvent mis en avant, un système de gestion de base de données (SGBD ou DBMS en anglais) n'a pas comme seul objectif de stocker l'information (on voit qu'il nous serait aisé de tout stocker dans des fichiers, mieux, PHP est susceptible de le faire pour nous avec les sessions, voir le chapitre suivant), mais aussi d'assurer la gestion des accès concurrents à une même information.

Dans PHP Saloon, nous allons donc confier le stockage permanent des informations contenues dans nos objets à un SGBD (SQLite). Pour cela, il va nous falloir décrire ces informations (qui ne sont pas nécessairement celles contenues exactement par les objets).

Le modèle de données constitue cette description de l'information à stocker. Il existe plusieurs méthodes pour représenter et décrire ce modèle, une des plus simples et des plus répandues en France est la méthode Merise. Elle est assez largement destinée au type de SGBD le plus répandu : les SGBD relationnels.

B.A.-BA Les tests Acid pour les bases de données

La question de l'accès concurrent à l'information est un point crucial pour toute application transactionnelle, ce qui est le cas du commerce électronique mais aussi de la plupart des échanges inter-entreprises, et ceci bien avant l'ère d'Internet.

Un jeu de prérequis de références a donc été mis au point pour s'assurer qu'un SGBD garantit les propriétés indispensables à l'exécution de transactions concurrentes, il s'agit des propriétés Acid (de nombreux tests existent pour vérifier la conformité à ces propriétés. La signification de cet acronyme est :

- atomicité ;
- cohérence ;
- isolement (isolation) ;
- persistance (*durability*).

L'atomicité garantit que chaque transaction se comporte comme une boîte noire dont l'exécution n'est pas décomposable. En outre, l'isolement garantit que les différents utilisateurs concurrents n'auront aucune visibilité sur les changements d'états à l'intérieur de la transition, chacun observant successivement l'état antérieur et l'état postérieur à l'exécution des transactions concurrentes.

L'ensemble des instructions d'une transaction devra naturellement respecter les contraintes d'intégrité inhérentes aux modèles de données (ce qui garantit la cohérence de la base sur la durée).

Le SGBD devra assurer la permanence des changements opérés dans une transaction dès lors que celle-ci est validée. Ceci même en cas de crash.

Ces SGBD stockent l'information sous forme de tables et la méthode Merise va nous permettre de décrire de manière générique (indépendamment du SGBD choisi) les tables (entités) et les associations qu'elles entretiennent.

Description avec Merise

La méthode Merise est particulièrement adaptée pour décrire un modèle de données. Plus ancienne qu'UML, elle reste la méthode de référence pour ce type de description et dispose d'un parc logiciel considérable, y compris dans le monde Open Source.

Merise impose de découper l'analyse des données en deux phases. Une première étape conceptuelle permet de décrire les entités (dans Merise on ne parle pas d'objet) et les relations (associations) entre elles sans entrer dans les contraintes du système de stockage.

La deuxième étape consiste à traduire, pour un type de stockage donné, le modèle conceptuel en un modèle physique adapté. Dans cette étape, les entités du modèle conceptuel sont transformées en tables et les associations donnent lieu à la création de contraintes d'intégrité entre tables (clés étrangères par exemple).

OUTIL NetObjects

Il est souvent difficile de trouver un équivalent libre aux outils métier comme Sybase Powerdesigner ou Rational Rose, cependant pour Merise et entité/relation, qui sont assez proches, il existe NetObjects. Cet outil permet de modéliser de A à Z un modèle de données et de générer le code SQL pour la plupart des SGBDR du marché.

▸ http://nextobjects.sourceforge.net

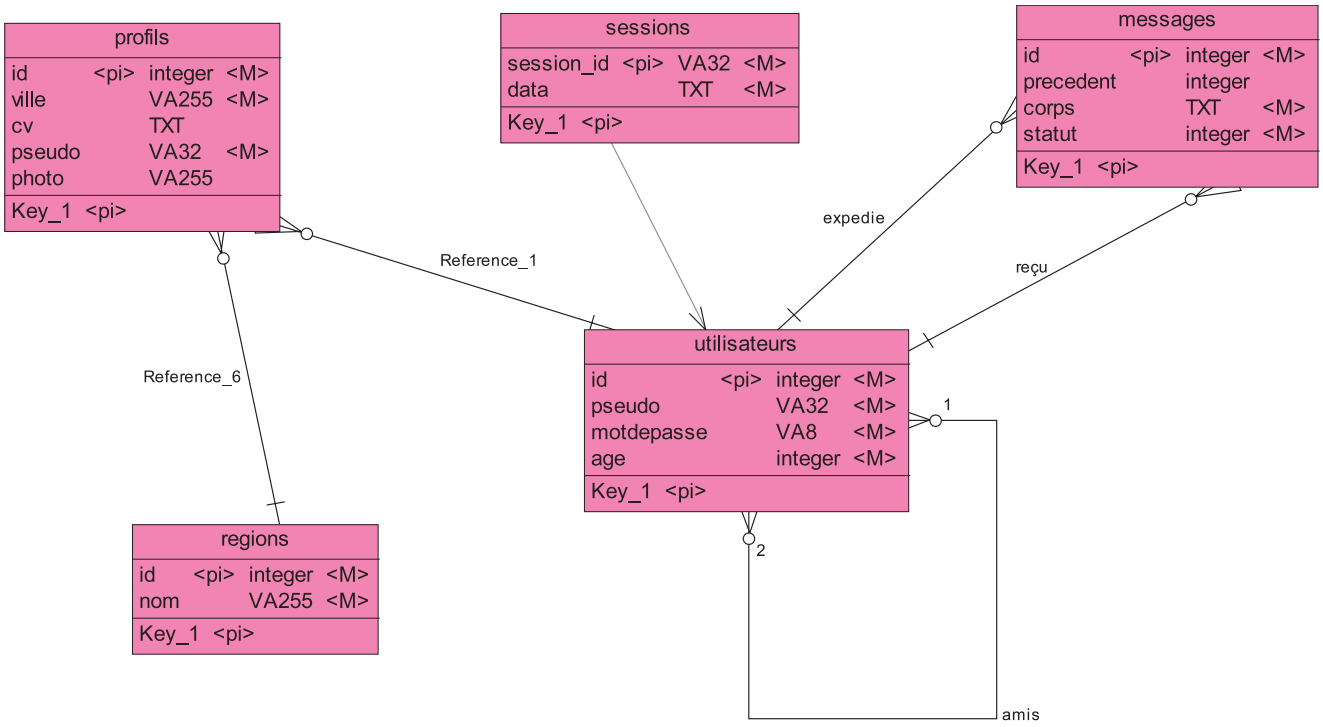


Figure 3–1 Modèle de données avec la méthode entité/relation

ALTERNATIVES **UML et entité/relation**

La méthode Merise est très répandue en France mais d'autres méthodes sont utilisables pour décrire nos données, c'est le cas de la méthode Entité/Relation ou d'UML, qui est très utilisée dans le monde objet.

Pour PHP Saloon, il aurait naturellement été possible d'utiliser un environnement de développement UML à tous les niveaux. Cependant :

- UML requiert en soi un ouvrage.
- Rares sont les IDE capables de gérer le code PHP (il est vrai que cela n'a de sens que depuis PHP 5).
- Enfin il n'est pas évident qu'UML soit le choix le plus judicieux pour décrire un modèle de données élémentaire.

Les experts UML pourront cependant se pencher sur Waterproof::UML, un tout nouvel outil de développement UML prévu spécifiquement pour PHP et qui vient compléter le déjà célèbre PHPEdit.

▸ http://www.waterproof-software.com/

📖 Pascal Roques, *Cahier du programmeur UML*, Eyrolles 2002

📖 Pascal Roques, *UML par la pratique*, Eyrolles 2003

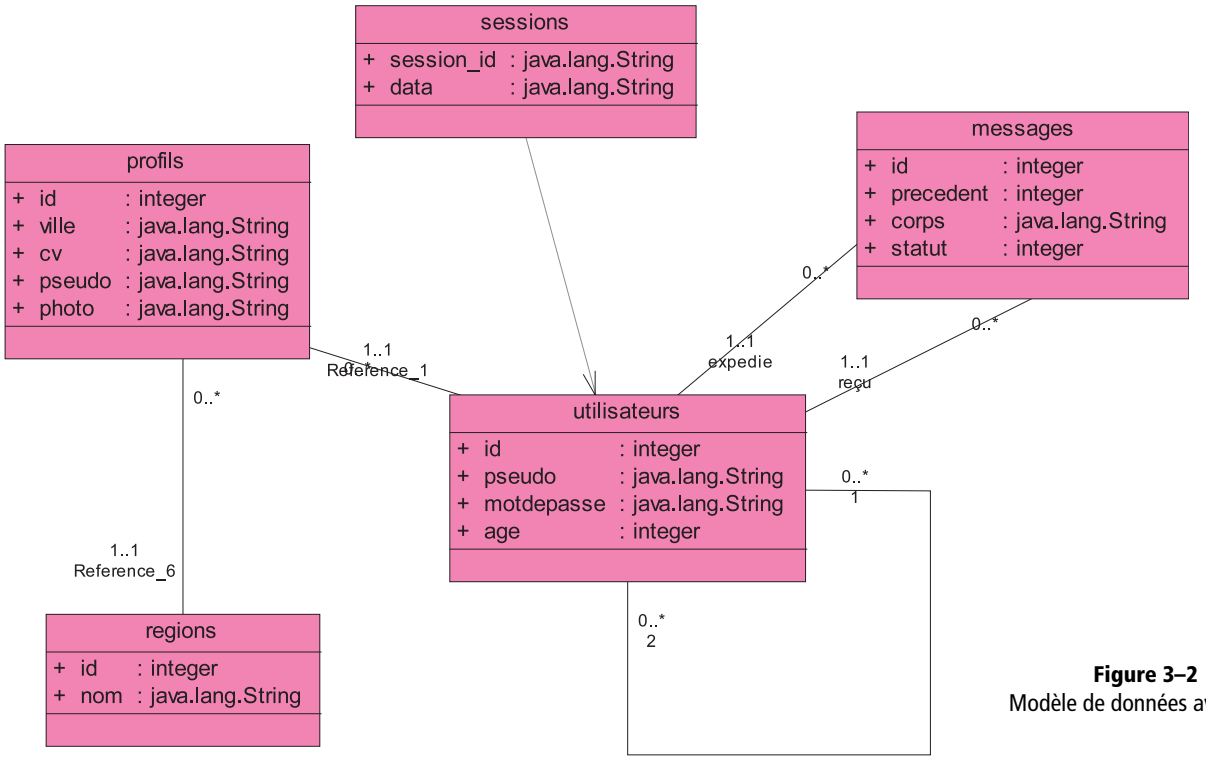


Figure 3–2
Modèle de données avec UML

Dans le cas de PHP Saloon, il est donc nécessaire en premier lieu d'identifier pour chaque classe d'objets les informations à stocker de manière permanente.

Si l'on considère par exemple le cas d'un utilisateur, il est nécessaire de mémoriser son pseudo, le mot de passe associé et son âge. À ceci s'ajoute un identifiant unique, qui va nous servir de clé d'accès à toutes ces informations (dans les schémas, la notation PI, *primary identifier*, est utilisée).

Nous pourrions utiliser aussi le pseudo comme clé d'accès, celui-ci étant unique, mais dans l'interrogation de la base il est toujours plus facile de comparer un simple numéro qu'une chaîne de caractères.

On obtient donc l'entité de la figure 3-3.

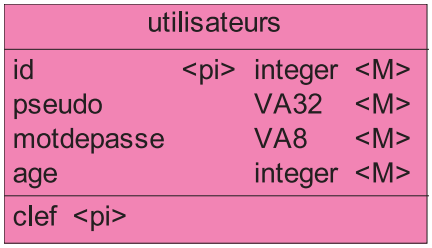


Figure 3–3 Entité utilisateurs

Chaque attribut de l'entité dispose d'un type, qui n'est pas le type physique qui sera utilisé au final, mais plutôt un type logique. Cette technique assure une relative indépendance du modèle conceptuel par rapport au SGBD qui implantera le modèle physique. Ainsi, `VA255` indique une chaîne de caractères de taille variable (d'où le `VA`) d'au maximum 255 caractères. On indique par ailleurs le caractère obligatoire ou non de l'attribut (avec la lettre M pour `mandatory`). La notation peut varier d'un outil à un autre mais le concept reste le même.

Nous devons procéder de même pour toutes les entités manipulées dans PHP Saloon, sans oublier de préciser les différentes associations.

Une association va mettre en évidence la relation entre deux ou plusieurs entités. On précisera en général les attributs qui sont corrélés et les cardinalités.

Pour PHP Saloon, on obtient alors le schéma conceptuel de la figure 3-4.

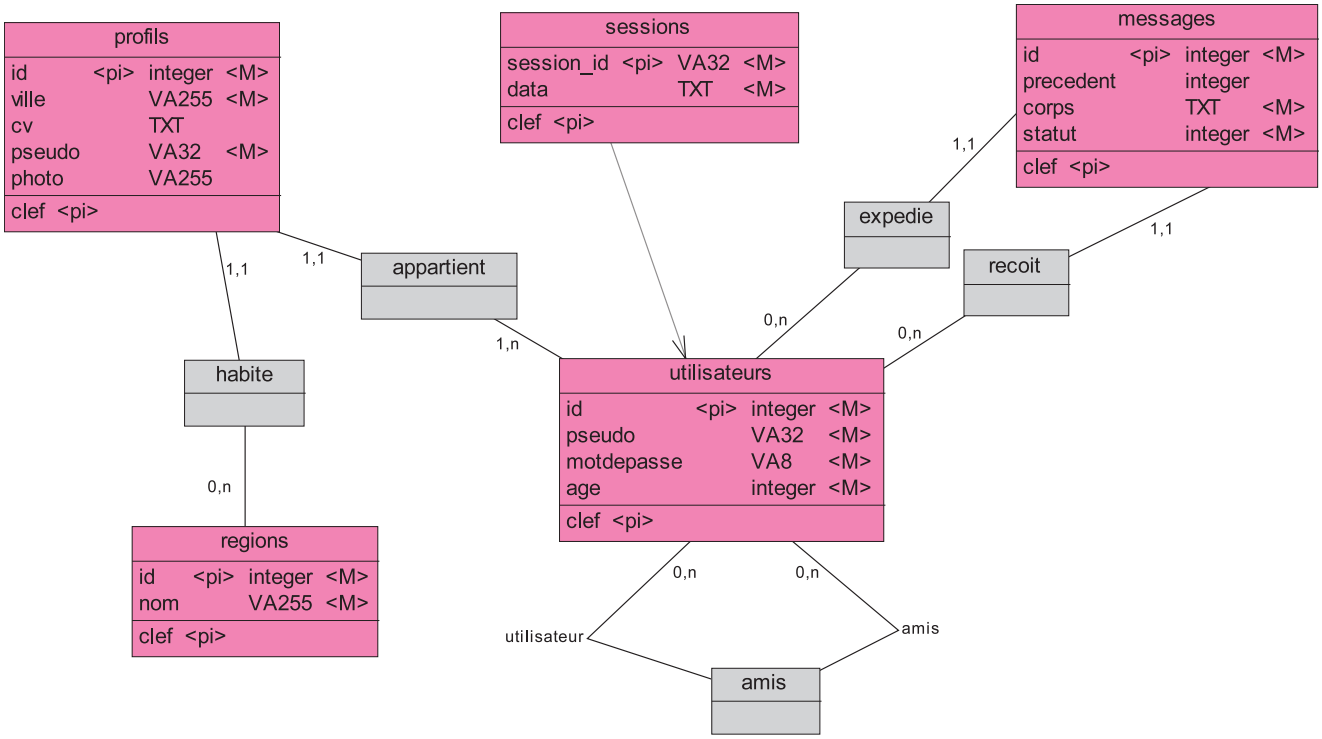


Figure 3–4 Modèle de données conceptuel de PHP Saloon

Les informations liées au profil des utilisateurs ont été regroupées dans une entité séparée. Il devient alors possible de proposer la création de plusieurs profils pour un même utilisateur.

L'association `appartient` matérialise la relation entre les deux entités `profils` et `utilisateurs`. En effet, dans l'entité `profils`, l'attribut `utilisateur` représente nécessairement la clé d'accès à un utilisateur décrit par l'entité `utilisateurs`. Cette contrainte va ainsi éviter que ne se créent des profils fantômes sans utilisateur associé. Lorsque le SGBD utilisé au moment de l'implantation le prend en charge, ce type de contrainte permettra même d'effacer automatiquement les profils associés à un utilisateur lors de sa suppression.

Le même type d'association est précisé pour l'entité `messages` dont les attributs `expediteur` et `destinaire` représentent tous deux une clé d'accès à un utilisateur.

Notre choix étant fait pour le type de stockage (un SGBD relationnel), nous pouvons transformer ce premier schéma en un modèle physique, qui va décrire non plus des entités mais des tables et des contraintes portant sur celles-ci.

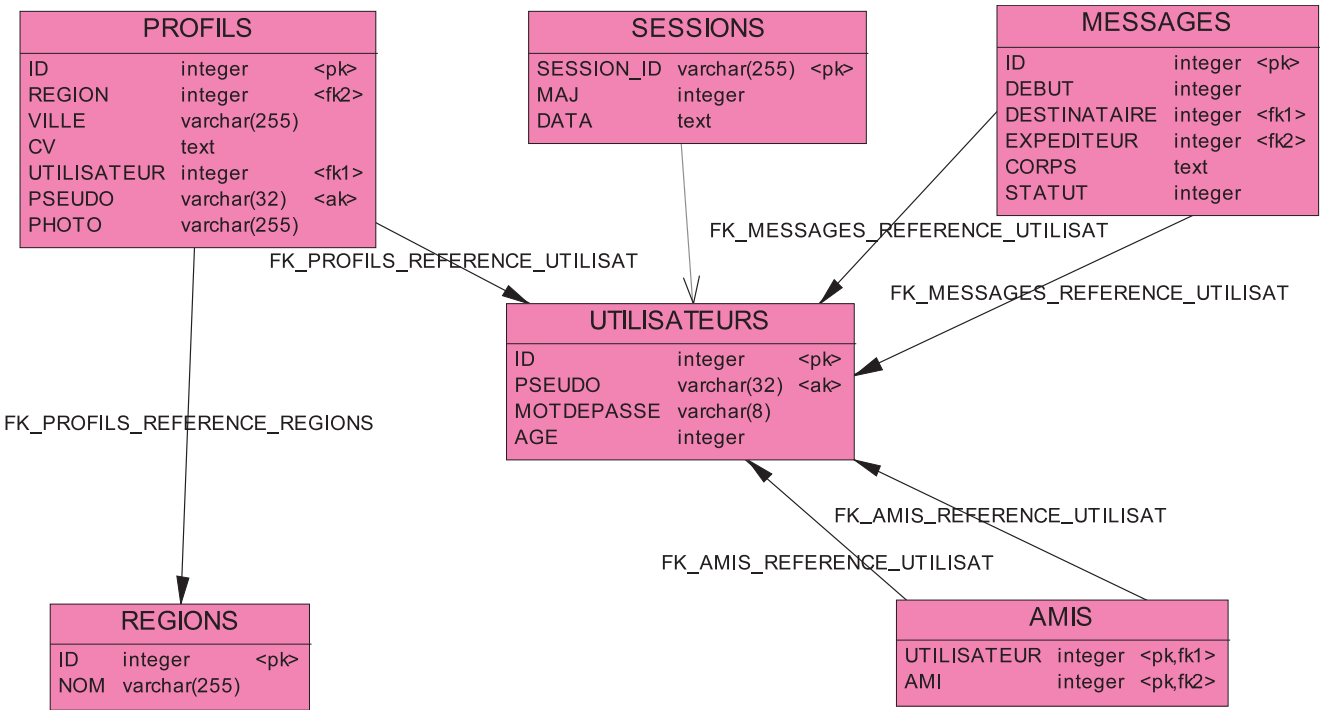


Figure 3–5 Modèle physique des données

MÉTHODE Modèle conceptuel de données

La définition d'un modèle conceptuel de données n'est en rien obligatoire. On aurait tout à fait pu créer le modèle physique immédiatement.
Cependant, la création d'un modèle conceptuel offre plus de souplesse en permettant de se concentrer sur les données sans d'emblée être confronté aux contraintes du modèle relationnel.

On obtient alors notre modèle de données final. Dans le cas présent, la plupart des entités ont été directement transformées en tables et les associations en contraintes d'intégrité. On observe cependant que l'association a donné lieu à la création d'une table. La transformation du modèle conceptuel en modèle physique n'est donc pas toujours aussi élémentaire qu'il n'y paraît.

Mise en œuvre de SQLite

Le modèle physique doit lui-même être transformé pour un SGBD donné. Si le langage utilisé par tous les SGBD relationnels est SQL, chacun en étend les possibilités ou hélas en élimine d'autres. Bref, au-delà d'un socle minimal, il faut savoir adapter au cas par cas.

Avant d'aller plus avant dans la transformation, il faut donc regarder d'un peu plus près ce qui se cache derrière SQLite.

SQLite, un SQL classique

ATTENTION Utiliser SQLite comme SGBD peut aussi avoir des inconvénients

SQLite est agréable lors des tests car il va accepter un code SQL plus large que ce qu'il implante. Il importe de réserver la méthode aux phases de test et de mise au point :

- Des risques de comportements différents sur le SGBD choisi pour la production peuvent persister.
- L'exportation des données de SQLite vers la production peut se révéler délicate si les contraintes d'intégrité ne sont pas respectées (ce qui au terme de la mise au point ne devrait plus être le cas néanmoins).

Du côté du langage SQL, SQLite adopte une approche réellement originale en étant compatible a priori avec la syntaxe du langage la plus complexe (y compris les contraintes d'intégrité), même si au final un certain nombre d'éléments sera ignoré. L'approche classique consiste plutôt à refuser les constructions non compatibles.

L'intérêt de cette méthode est double :

- Il devient possible d'utiliser un code SQL standard avec SQLite, ce qui veut dire qu'il n'y a aucune pénalité à commencer avec SQLite avant peut-être d'adopter une base plus lourde, en production par exemple.
- Si par chance certaines fonctionnalités sont compatibles avec les prochaines versions, cela le sera de manière transparente.

Ainsi, SQLite va donc ignorer toutes les définitions de contraintes liées aux associations de notre modèle conceptuel. Aucune vérification ne sera par exemple faite pour s'assurer que la clé utilisateur de la table profil représente bien effectivement un utilisateur existant dans la table.

Autre simplification, qui est de taille, au moment de la création de table, comme nous l'avons vu précédemment, chaque attribut se voit associer un type, chaîne de caractères, entier... SQLite va tout simplement ignorer l'ensemble de ces définitions pour ne retenir qu'un type de stockage : la chaîne de caractères. Un certain nombre de vérifications ne seront donc pas faites, mais fondamentalement le modèle reste le même. On imagine comment la mise au point d'une application peut être ainsi facilitée.

ATTENTION **SQLite ne supporte pas les procédures stockées**

Si le support SQL de SQLite est plutôt complet, il n'est cependant pas possible de réaliser de procédures stockées.

Les procédures stockées sont des procédures ou fonctions définies dans un langage propre au SGBD (mais en général on propose aussi les langages de scripts courants) et qui sont exécutées directement par le SGBD lui-même dans les requêtes.

Il en résulte de meilleures performances car la manipulation des données est effectuée au sein du SGBD sans aller-retour entre le programme utilisateur et la base. De plus, certains SGBD vont stocker le code des procédures de manière optimisée.

À défaut de procédures stockées, l'extension SQLite de PHP permet de faire appel à n'importe quelle fonction PHP via une fonction interface. Par exemple :

```
select php('strlen', 'abc');
```

Cette requête va exécuter la fonction PHP `strlen` sur la chaine `'abc'` et retourner 3. Toutes les fonctions PHP sont ainsi accessibles depuis le code SQL.

Attention, ce mécanisme n'est disponible que pour l'extension PHP et ne l'est donc pas pour les autres clients SQLite, comme le client en ligne de commande.

À l'inverse, SQLite supporte parfaitement un point essentiel, si l'on tient compte de la définition donnée précédemment d'un SGBD : les transactions. Vous ne rencontrerez aucune difficulté pour créer une application de commerce en ligne et il est inutile de définir des tables de quel que type que ce soit. SQLite prend en charge les transactions nativement.

SQLite, un SGBD sans serveur

Autre originalité de l'approche, la quasi-totalité des systèmes de SGBD relationnels met en œuvre un logiciel serveur (sous Windows un service). Chaque programme se comporte en client et doit se connecter au serveur, le plus souvent en utilisant les protocoles réseau classiques (ceux-là même qui sont utilisés entre un navigateur et le serveur web consulté).

La conséquence de cette architecture, certes robuste, est une certaine lourdeur. Il faut effectivement administrer le serveur, en vérifier le bon fonctionnement et mettre en œuvre les protocoles pour s'y connecter.

Avec SQLite, rien de tout cela, le moteur qui exécute les instructions SQL n'est plus dans un serveur mais dans une simple bibliothèque. Et les données sont stockées dans un fichier. Difficile de faire plus simple.

Inutile alors de se demander si son hébergeur dispose de serveurs de base de données, de s'interroger sur leur nature, il suffit de se munir d'un PHP avec l'extension SQLite (elle est disponible par défaut) pour pouvoir utiliser le potentiel d'un SGBD relationnel et du langage SQL.

Cette approche prend aussi tout son intérêt dans la création d'applications non plus forcément web mais de logiciels autonomes, par exemple en utilisant PHP-GTK. Il devient possible de disposer en standard d'un SGBD intégré.

EXTENSION **PHP-GTK**

PHP-GTK est une extension de PHP qui ajoute au langage le support de la bibliothèque GTK (Gnome ToolKit). Il s'agit d'une bibliothèque graphique multi-plates-formes qui permet de créer de véritables applications graphiques aussi bien sous MS-Windows que GNU/Linux.

PHP-GTK constitue donc un environnement complet pour créer des applications autonomes.

- http://Gtk.php.net
  la home page du projet PHP-GTK.

SQLite est donc particulièrement séduisant pour le développement rapide d'applications ou quand la charge ne nécessite pas une base digne des plus grandes multinationales.

Implantation de notre modèle

Tables et contraintes

SQLite va nous permettre de procéder à une transformation standard de notre modèle physique en SQL. L'outil utilisé pour réaliser les schémas de cet ouvrage et générer le SQL a été configuré pour PostgreSQL, un des SGBD les plus connus du monde Open Source avec MySQL. Le code généré sera pourtant tout à fait correctement interprété par SQLite...

On constate que ce code SQL comporte en réalité deux parties :

- la création des tables proprement dites ;
- les contraintes d'intégrité sur le modèle.

Les contraintes sont ignorées dans la version actuelle de SQLite. Cependant, si vous souhaitez utiliser une base de données plus classique, ce code sera pleinement pris en compte.

Si vous découvrez SQL, sachez que ce langage constitue la référence pour manipuler les données stockées dans les SGBD (relationnels principalement). Les bases du langage sont aisément compréhensibles, on pourrait presque lire les instructions comme du mauvais anglais. Bien entendu, le langage ne se limite pas aux instructions relativement simples, mises en œuvre dans PHP Saloon, et si vous souhaitez en savoir plus, SQL draine une littérature abondante.

- C. Soutou, *De UML à SQL*, Eyrolles 2002.
- G. Gardarin, *Bases de données*, Eyrolles 1999.
- F. Brouard, *SQL Développement*, Campus Press 2001.

ALTERNATIVES **Le support des SGBD par PHP**

PHP propose une interface à la quasi-totalité des bases de données du marché, depuis Microsoft SQL Server à Oracle, en passant par PostgreSQL ou MySQL.

Pour PHP Saloon, nous mettons SQLite en avant, cependant le code SQL utilisé est tout à fait standard, une très large majorité des SGBD supportés par PHP peuvent donc être utilisés. Il suffit pour cela de remplacer les quelques instructions SQLite par leurs équivalents.

Requêtes SQL dans PHP Saloon

Dans PHP Saloon, outre la création des tables et l'intégration des contraintes dans celles-ci, qui ne s'effectuent qu'une seule fois, nous allons exécuter abondamment 3 types de requêtes SQL :

- `select` pour des interrogations ;
- `insert` pour des insertions ;
- `delete` pour des suppressions.

Les interrogations consistent à isoler, parmi les données stockées dans les tables, celles qui respectent un certain nombre de contraintes. Ces données peuvent alors être extraites de la base de données pour être utilisées.

Pour obtenir l'intégralité des informations d'un profil, on va extraire pour un utilisateur donné l'ensemble des colonnes des tables profils et utilisateurs :

```
select
   profils.*, utilisateurs.*
from
   profils, utilisateurs
where
   utilisateurs.uid = profils.utilisateurs$$
   and
   utilisateurs.uid = 3

order by
   profils.pseudo
   asc
```

- Sélection des colonnes à isoler.
- Tables impliquées.
- Conditions à vérifier. Ici, on fait le rapprochement entre un profil et l'utilisateur qui en est le propriétaire (on parle de jointure entre les tables utilisateurs et profils), en l'occurrence l'utilisateur dont l'identifiant est 3.
- L'instruction `select` dispose d'options qui permettent d'ordonner les données ou de les regrouper. On souhaite avoir les profils classés par pseudo pour cet utilisateur.

Ces opérations de sélection n'ont de sens qu'à la condition d'avoir au préalable pu insérer des données, c'est le rôle de l'instruction `insert`. Ajoutons un deuxième profil à l'utilisateur portant le numéro `3` :

```
insert into
   profils
   ( id, ville, region, cv, utilisateur)
values
   ( 3 , 'Paris', 1 , 'Fan de PHP 5', 3)
```

- Table dans laquelle insérer les données. Si seules certaines colonnes comportent des données, celles-ci sont précisées entre parenthèses.
- Données à insérer.

Enfin, le troisième type d'instruction à être largement utilisé dans PHP Saloon est `delete`, qui va nous permettre d'effacer les sessions expirées ou de supprimer un utilisateur de test :

```
delete from
   utilisateurs
where
   uid = 3
```

- Table dans laquelle effacer les données.
- Condition optionnelle. Sans condition, toutes les lignes de la table sont effacées.

Tester SQLite en direct

Toutes ces requêtes SQL vers la base de données vont naturellement être exécutées par les objets PHP au sein même du code de PHP Saloon. PHP 5 définit pour ce faire 3 classes :

- `sqlite_db` : une connexion sqlite ;
- `sqlite_query` (et `sqlite_ub_query` en variante non bufférisée) : le résultat d'une requête ;
- `sqlite_exception` : exception éventuellement levée.

Chaque classe dispose de méthodes qui vont permettre d'interagir avec les bases SQLite.

Tableau 3–1 Méthodes de la classe `sqlite_db`

Méthode	Description
`query`	Exécution d'une requête.
`array_query`	Exécution d'une requête et retour de l'intégralité des valeurs sous forme de tableau.
`single_query`	Exécution d'une requête et retour de la première colonne de tous les résultats.
`unbuffered_query`	Exécution d'une requête sans préchargement des résultats.
`last_insert_rowid`	Retourne l'identifiant de la dernière ligne insérée.
`create_aggregate`	Création d'un agrégat.
`create_function`	Création d'une fonction.
`busy_timeout`	Retourne le délai d'attente maximal en vigueur lorsque la base est bloquée par une autre instruction.
`last_error`	Retourne le numéro de la dernière erreur.

Tableau 3–2 Méthodes de la classe `sqlite_query`

Méthode	Description
`fetch_array`	Récupération de la ligne courante du résultat sous forme de tableau.
`fetch_object`	Récupération de la ligne courante sous forme d'objet.
`fetch_single`	Récupération de la première colonne de la ligne courante du résultat.
`fetch_all`	Récupération de l'intégralité du résultat de la requête sous forme de tableau.
`column`	Retourne la valeur d'un élément dans la ligne courante du résultat.
`changes`	Retourne le nombre de changements opérés par la requête.
`num_fields`	Retourne le nombre de colonnes dans le résultat.
`field_name`	Retourne le nom d'une colonne.
`current`	Retourne la ligne courante dans le résultat.
`next`	Retourne la ligne suivante dans le résultat.
`prev`	Retourne la ligne précédente dans le résultat.
`hasprev`	Vrai si la ligne courante n'est pas la première dans le résultat.
`rewind`	Revient à la première ligne du résultat.
`num_rows`	Retourne le nombre de lignes du résultat.
`seek`	Permet de se positionner parmi les lignes du résultat.
`valid`	Vrai si la ligne courante n'est pas la dernière dans le résultat.

B.A.-BA Variante non bufférisée

Lorsqu'on exécute une requête SQL, le résultat est construit en mémoire. Il occupe donc naturellement de l'espace et si le nombre de lignes retournées est important, il nous faut beaucoup d'espace.
Dans ce cas, on a donc très souvent intérêt à ne pas construire l'intégralité du résultat à l'exécution de la requête mais au contraire à ne délivrer progressivement que les éléments demandés.
SQLite permet de ne pas faire exploser l'espace mémoire occupé par le script PHP (en général limité à 8 Mo). Dans le cas des SGBD classiques, cela évite en plus de faire transiter inutilement des données sur le réseau.

PHP 4 Sqlite est aussi disponible via des fonctions très classiques

Si vous utilisez PHP 4, pas de panique ! L'extension SQLite définit certes des classes, ce qui est la manière la plus simple de faire dans l'univers objet, mais les fonctions classiques comme celles disponibles pour les autres bases n'ont pas pour autant disparu.
Vous pourrez ainsi utiliser `sqlite_connect()` ou `sqlite_query()` d'une façon très similaire à celle proposée dans le code.

Une séquence classique d'utilisation est la suivante :

```
<?
$db = new sqlite_db('base.sqlite');
$db->query('create table foo ( bar int not null)');
$db->query('insert into foo values ( 4 )');
$db->query('insert into foo values ( 5 )');
print_r($db->array_query('select * from foo'));
?>
```

Par ailleurs, SQLite dispose d'interfaces orientées vers de nombreux langages. La bibliothèque peut bien sûr être utilisée nativement en C, mais aussi avec le langage Tcl ou encore en Perl. À ces interfaces, s'ajoute un petit utilitaire en ligne de commande. Ce programme permet d'interroger directement un fichier SQLite. Cette possibilité est évidemment particulièrement utile pendant les phases de mise au point. Il est ainsi aisé d'observer le résultat des actions réalisées en PHP et de suivre l'évolution de la base.

ALTERNATIVE ADOdb & Pear DB

Dans PHP Saloon, nous utilisons les méthodes et fonctions SQLite natives. C'est une bonne manière de comprendre le fonctionnement de la base, mais cela altère néanmoins la portabilité du code.
Deux couches d'abstraction majeures existent pour PHP. Au sein de Pear tout d'abord, avec Pear DB. Sous forme de bibliothèque pour ADOdb, également.
Dans les deux cas, il s'agit de permettre d'exécuter les mêmes instructions quelles que soient les bases à interroger. Ce n'est pas toujours possible et on aura intérêt à rester sur du SQL très standard.
À ce jour, ADOdb semble offrir une robustesse et une richesse dans le support des SBGD des plus importantes. En outre, le support de PHP 5 est bien intégré, notamment avec la prise en compte de l'interface SPL, comme c'est le cas dans PHP Saloon.

- http://pear.php.net
- http://php.weblogs.com/ADODB

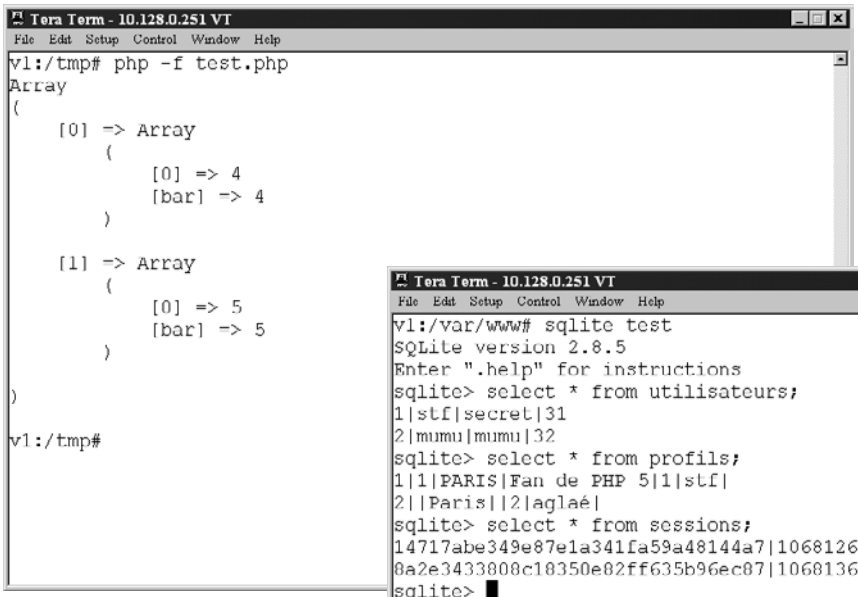


Figure 3–6
Résultat du programme de test

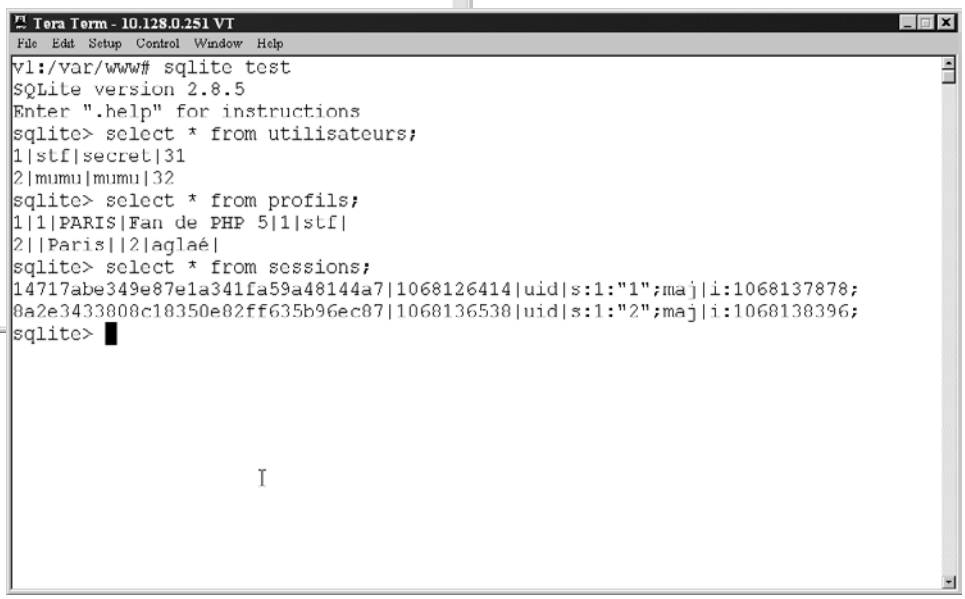


Figure 3–7 Exécution de requêtes en ligne de commande

SQLite et les transactions

Le programme client en ligne de commande `sqlite` est aussi un moyen simple de tester le fonctionnement du mécanisme des transactions.

Le principe d'une transaction est simple. La transaction agit comme un container qui obscurcit certaines des instructions SQL. Quel que soit le nombre d'instructions SQL dans une transaction, ceux qui se connectent à la base n'auront jamais la vision des étapes intermédiaires. Ils accéderont à la base telle qu'elle était auparavant si la transaction n'est pas encore terminée, ou en verront l'état final, à l'étape ultime de la transaction.

En outre, le contenu d'une transaction se déroule intégralement ou est annulé dans sa totalité. La base reste ainsi cohérente, quoiqu'il arrive.

Au sein d'une transaction, comme dans toute instruction SQL, le SGBD vérifie que les contraintes d'intégrité précisées dans le modèle sont respectées.

En SQL, le contenu d'une transaction est délimité par les instructions `begin` et `end`. L'instruction `rollback` permet d'annuler l'ensemble des opérations en cours de route. Il est possible de préciser dès le départ la conduite à tenir en cas d'erreur, le début de la transaction est alors :

```
begin on conflict reaction
```

Avec une réaction qui peut être :

- `rollback` : toute la transaction est annulée ;
- `abort` : la transaction est arrêtée, les changements de l'instruction fautive sont annulés, ceux effectués par les instructions précédentes demeurent ;
- `fail` : la transaction est arrêtée, les changements de l'instruction fautive réalisés jusqu'à l'apparition de l'erreur demeurent, tout comme les modifications des instructions précédentes ;
- `ignore` : la transaction est exécutée jusqu'à son terme, tous les changements qui ont pu avoir lieu demeurent, par exemple dans une instruction de mise à jour. Si la ligne a posé problème, toutes les autres sont néanmoins bien mises à jour ;
- `replace` : dans le cas d'une opération d'insertion, si un élément déclaré unique pré-existe, l'insertion est transformée en mise à jour.

Dans PHP Saloon, les transactions sont nécessaires. Certaines actions, par exemple l'inscription, comportent plusieurs phases. Celles-ci doivent être réalisées au sein d'une transaction pour éviter les interférences entre inscriptions simultanées. On évite ainsi de créer un profil sans finalement pouvoir l'associer à un utilisateur ou réciproquement.

Dans les deux séquences suivantes deux utilisateurs tentent leur inscription en parallèle et choisissent, hélas, le même pseudo pour leur premier profil. L'inscription de l'un des deux va être totalement annulée.

Session de Toto	Session de Titi
L'utilisateur Toto s'inscrit et termine sa transaction en premier.	L'utilisateur Titi voit sa transaction annulée à la fin, car comme Toto, il a voulu nommer son profil phpman.

```
xterm
sqlite> begin on conflict rollback;
sqlite> insert into
   ...>   utilisateurs ( id, pseudo, motdepasse, age )
   ...> values
   ...>   ( 1, 'toto', 'secret', 25 );
sqlite> insert into
   ...>   profils ( id, region, ville, cv, utilisateur, pseudo)
   ...> values
   ...>   ( 1, 1, 'Paris', 'Fan de PHP', 1 , 'phpman');
sqlite> end;
sqlite>
```

```
xterm
sqlite> begin on conflict rollback;
sqlite> insert into
   ...>   utilisateurs ( id, pseudo, motdepasse, age )
   ...> values
   ...>   ( 2, 'titi', 'mystere', 32);
sqlite> select * from utilisateurs;
1|toto|secret|25
2|titi|mystere|32
sqlite> insert into
   ...>   profils ( id, region, ville, cv, utilisateur, pseudo)
   ...> values
   ...>   ( 2, 1, 'Lyon', 'Autre Fan de PHP', 2, 'phpman');
SQL error: uniqueness constraint failed
sqlite> select * from utilisateurs;
1|toto|secret|25
sqlite>
```

Les transactions vont nous permettre de préserver l'état de cohérence dans notre base de données.

Création d'une vue CONNECTES

À ce stade, notre modèle de données et les outils nécessaires pour le manipuler en toute sécurité sont complets. Toutefois, SQL et SQLite permettent la création de vues, et ceci va nous simplifier encore le travail sur les données.

Même si la structure des requêtes SQL est simple, on comprend assez facilement que le regroupement de données éclatées dans plusieurs tables va donner lieu à des instructions `select` à rallonge. C'est notamment le cas lorsqu'il s'agit d'identifier les connectés. Il faut en effet réunir les données de trois tables, sans oublier naturellement les conditions qui portent sur les colonnes.

Si la requête ne doit être utilisée qu'une fois, cela ne porte pas à conséquence, si par contre, comme on l'imagine pour la liste des connectés, la requête fait l'objet de nombreux appels, alors il devient particulièrement ardu de reproduire l'intégralité du code SQL.

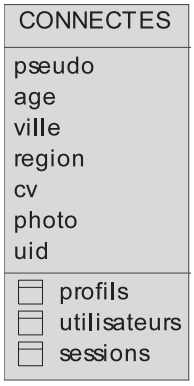


Figure 3–8 La vue CONNECTES

Le langage permet donc de définir des vues. Une vue est une sorte d'alias, qui crée une table virtuelle à partir du résultat d'une instruction `select`.

Pour PHP Saloon, nous allons définir une bonne fois pour toutes la requête indispensable pour obtenir les connectés et en faire une vue, que nous allons naturellement appeler `CONNECTES`.

```
create view
   CONNECTES
as // requête qui correspondra à connectes
   select
      profils.pseudo as pseudo,
             // as permet de renommer un champ dans le résultat final
      utilisateurs.age as age ,
      profils.ville as ville,
      profils.region as region,
      profils.cv as cv,
      profils.photo as photo,
      utilisateurs.id as uid
   from
      profils, utilisateurs, sessions
   where
      utilisateurs.id = profils.utilisateur and
      php('session::getSessionData', sessions.data, 'uid') =
             utilisateurs.id and
          ( php('time') -
            php('session::getSessionData',sessions.data, 'maj'))
         < 120;
```

Dans cette vue, nous avons utilisé la possibilité d'appeler des fonctions PHP depuis les requêtes SQL. Elles nous permettent de décoder le contenu des sessions (voir le chapitre suivant) ou d'obtenir le résultat de la fonction PHP `time`.

Par la suite, et pour toutes les requêtes, nous pourrons utiliser `connectes` comme une table classique. Pour sélectionner les connectés, il suffira donc d'exécuter :

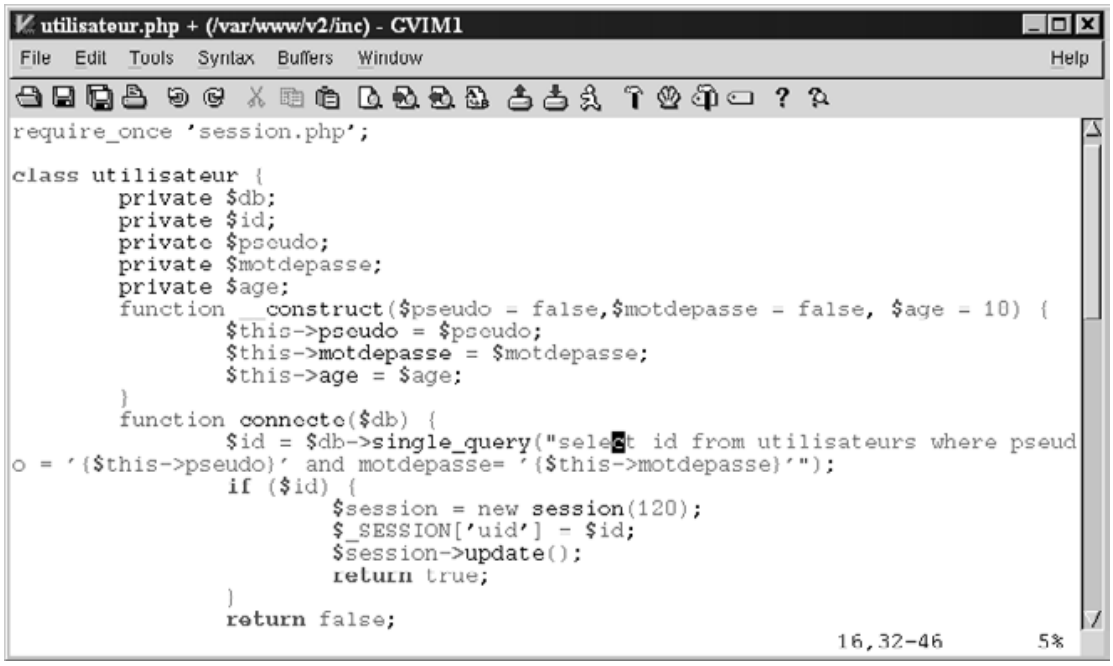
```
select * from connectes;
```

Cette manière de procéder concourt de même à mieux découper le travail. Dans un projet professionnel, les spécialistes du modèle de données construiront pour les créateurs de l'application les accès adaptés aux données, que ce soit sous forme de tables ou de vues.

En résumé…

Dans ce chapitre, nous avons résolu le problème de la persistance des informations manipulées en commençant par en faire le tour de manière précise avec le modèle conceptuel de données (MCD) de Merise, puis en utilisant SQLite pour tout stocker. Ce SGBD relationnel, à l'apparence anodine, voire simpliste, se révèle au final tout à fait sympathique. Puissant sans être contraignant, il permet de réaliser rapidement une application test sans pour autant hypothéquer un basculement vers des bases plus lourdes.

chapitre 4


utilisateur.php + (/var/www/v2/inc) - GVIM1
File Edit Tools Syntax Buffers Window Help
require_once 'session.php';
class utilisateur {
private $db;
private $id;
private $pseudo;
private $motdepasse;
private $age;
function __construct($pseudo = false,$motdepasse = false, $age = 10) {
$this->pseudo = $pseudo;
$this->motdepasse = $motdepasse;
$this->age = $age;
}
function connecte($db) {
$id = $db->single_query("select id from utilisateurs where pseudo = '{$this->pseudo}' and motdepasse= '{$this->motdepasse}'");
if ($id) {
$session = new session(120);
$_SESSION['uid'] = $id;
$session->update();
return true;
}
return false;
16,32-46 5%

Les objets dans PHP 5

Le développement objet dans PHP n'a jamais été en vogue. Et pour cause, les capacités du langage en la matière étaient tout simplement primitives. PHP 5 bouleverse cet état de fait, comme nous allons le voir avec la classe utilisateur de PHP Saloon. Protection des données, utilisation généralisée de références : PHP 5 prend les choses en main.

SOMMAIRE

MOTS-CLÉS

- Encapsulation
- Héritage
- Méthodes
- Attributs
- Référence

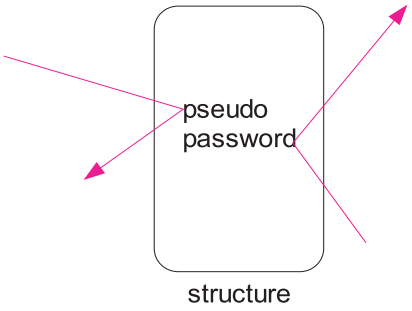
Avant de découvrir les choix effectués par les concepteurs de PHP 5, il est important de bien comprendre ce qu'est un objet. Historiquement parlant, la quasi-totalité des langages dispose de la notion de structure, ou de celle de *record* dans les langages dérivés de Pascal.

La structure permet d'agglomérer en un seul élément plusieurs données distinctes. On peut ainsi créer une structure pour manipuler la notion d'utilisateur dans PHP Saloon. Une telle structure grouperait le nom de code de l'utilisateur connecté et son mot de passe, tout cela en une seule entité, soit dans le langage C :

```
typedef struct {
   char *pseudo;
   char *motdepasse;
} utilisateur;
```

L'objet propose un modèle beaucoup plus élaboré que ce collage élémentaire en apportant trois niveaux d'amélioration :

- la protection des données de l'objet et la définition d'interfaces pour le manipuler ;
- la possibilité de définir des objets non seulement par composition mais aussi par extension ;
- le contrôle repoussé le plus tard possible de la nature exacte de l'objet afin de gagner en souplesse.



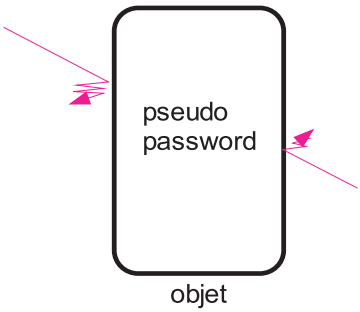


Figure 4–1
La protection des données par l'objet

Encapsulation et protection des données

Seul inconvénient à cette structure : les données agglomérées sont librement accessibles.

Reprenons l'exemple de notre structure `utilisateur`. Rien ne nous empêche d'aller modifier directement le nom de code au sein de cette dernière, ceci même si nous avons pris soin de créer une fonction `changepseudo` pour cela.

Naturellement, si par la suite nous devons modifier la manière dont est mémorisé ce nom de code (par exemple pour le compléter par le prénom), les problèmes surgiront de toutes parts. Chacun peut modifier ce que bon lui semble dans les variables de type `utilisateur`.

Une objection consiste à faire remarquer qu'il suffirait de rendre obligatoire l'utilisation de la fonction `changepseudo`. L'expérience montre hélas que l'être humain, et plus particulièrement l'informaticien, est globalement peu sensible à ce type de directives.

Avec l'objet, l'innovation porte sur la notion de protection des données. Au lieu de donner libre accès à l'information dans une sorte de structure poreuse, l'objet se définit comme une enceinte imperméable, même au développeur, qui va protéger les données contenues. On parle d'encapsulation des données et il s'agit probablement de la caractéristique la plus importante d'un objet.

En parallèle à cette protection, l'objet va disposer de méthodes. Ces méthodes sont des fonctions qui lui sont associées et elles vont servir d'interfaces entre le monde extérieur et les données contenues dans l'objet. L'objet fonctionne ainsi comme une boîte noire : on ne sait pas ce qu'il contient et comment il fonctionne, mais par chance nous disposons de méthodes pour le contrôler.

Dans notre exemple, la fonction `changepseudo` est une méthode de l'objet `utilisateur`. Elle permet en toute sécurité de modifier le pseudo. Si demain, la structure interne de notre objet est modifiée, il suffira d'adapter cette méthode et les utilisateurs de notre objet ne seront pas impactés.

Dans la pratique, on est souvent amené à manipuler des objets de même nature. Mais on ne souhaite pas avoir à redéfinir pour chaque objet ses attributs et ses méthodes. On utilise pour cela la notion de classe.

Une classe va servir à décrire un type d'objet et sera ensuite utilisée comme un moule, un modèle pour fabriquer à volonté autant d'objets désirés. Pour nos objets `utilisateur`, on peut ainsi définir une classe et les créer de cette façon :

```
class utilisateur {
   var $pseudo;
   var $password;
   function changepseudo($nouveau_pseudo) {
      $this->pseudo = $nouveau_pseudo;
   }
   function changepassword($nouveau_password) {
      $this->password = $nouveau_password;
   }
   function litpseudo() {
      return $pseudo;
   }
   function litpassword() {
      return $password;
   }
}
$u1 = new utilisateur();
$u2 = new utilisateur();
```

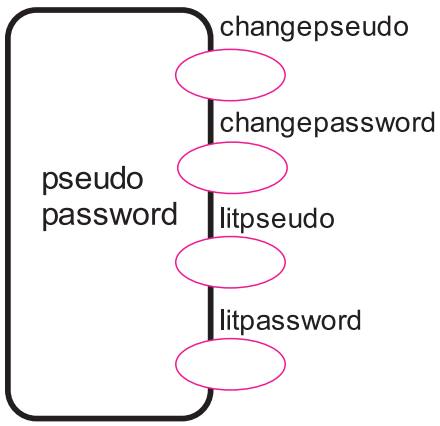


Figure 4–2 Attributs et méthodes d'un objet

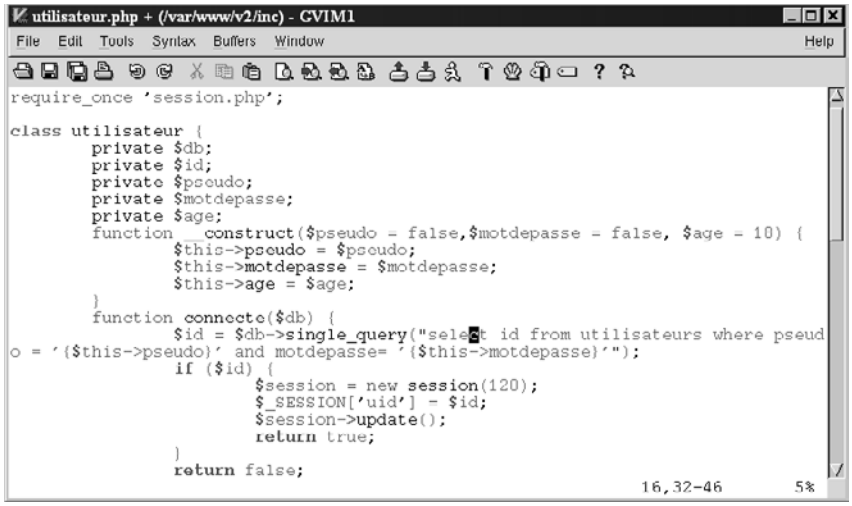


Figure 4–3 Première version de la classe utilisateur

Ce code est valide à la fois en PHP 4 et en PHP 5. L'instruction new est utilisée pour créer de nouveaux objets, comme dans une très large majorité de langages objet.

SYNTAXE **$this et l'opérateur ->**

PHP ne verse pas dans l'originalité pour ce qui est de son implantation du modèle objet. Comme pour new, la plupart des notations courantes ont été adoptées par PHP.

Ainsi la variable `$this`, disponible dans toutes les méthodes d'objet, désigne l'objet lui-même. C'est donc avec cette variable qu'on accède aux attributs et aux méthodes de l'objet courant dans une méthode.

Enfin, l'opérateur -> permet de désigner les méthodes ou les attributs manipulés :

```
$u1->pseudo = 'titi';
$u1->changepseudo('toto');
```

Contrairement à d'autres langages, PHP n'utilise pas la notation pointée. Deux notations n'auraient d'ailleurs pas de sens dans un langage sans pointeurs et où désormais tous les objets sont traités comme des références.

Protection des données : les 3 « P »

Si l'on retrouve bien la notion d'objet et de classe dans le code précédent et dans PHP 4, aucune protection des données n'est malheureusement réalisée, malgré ce qu'on pourrait supposer. Ainsi, il est tout à fait possible d'altérer la valeur de l'attribut `pseudo` sans passer par les méthodes définies :

```
$u1->pseudo = 'Pirate';
```

En PHP 4, il n'existe aucun moyen simple pour éviter cela. Cet état de fait traduit la réalité de l'implantation du modèle objet dans PHP 4. Les objets y sont construits comme de simples tableaux associatifs.

PHP 5 complète ce modèle objet initial en ajoutant la protection des données. La solution retenue est classique et ne se distingue en rien des langages objet courants. Celle-ci repose sur trois niveaux de protection :

- public ;
- protégé (protected) ;
- privé (private).

Le tableau ci-dessous résume l'impact de cette classification sur l'accès aux données (attributs ou méthodes).

Accès	Public	Protégé	Privé
À partir de la classe elle-même	✔	✔	✔
À partir de classes dérivées	✔	✔	✘
De l'extérieur	✔	✘	✘

Pour protéger nos attributs, le code précédent peut donc être réécrit :

```
class utilisateur {
   private $pseudo;
   private $password;
   function changepseudo($nouveau_pseudo) {
      $this->pseudo = $nouveau_pseudo;
   }
   function changepassword($nouveau_password) {
      $this->password = $nouveau_password;
   }
   function litpseudo() {
      return $pseudo;
   }
   function litpassword() {
      return $password;
   }
}
```

PHP 4 La compatibilité reste assurée

Par défaut ou si `var` est utilisé, les attributs sont considérés publics, ce qui garantit la compatibilité avec PHP 5 des objets et des classes définis dans PHP 4.

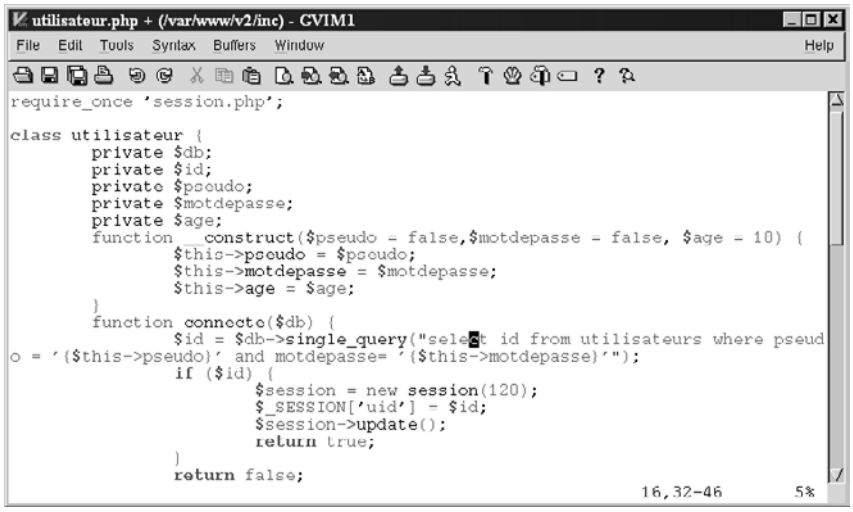


Figure 4–4 Classe utilisateur avec attributs privés

Les mots-clés `public`, `private` ou `protected` sont utilisés en lieu et place de `var`. Avec cette adaptation, il devient alors impossible d'accéder indûment aux attributs `pseudo` et `password` :

```
$u = new utilisateur();
$u->pseudo = 'Toto';
```

Le code provoque une erreur :

```
Fatal error: Cannot access private property utilisateur::$pseudo in 04-04.php on line 20
```

Ces deux notions réunies font de PHP 5 un véritable langage objet. D'autres facilités vont être disponibles, cependant le cœur du concept objet est désormais là : encapsulation et protection des données.

SYNTAXE Méthodes statiques

La plupart du temps, les méthodes s'appliquent sur un objet précis. Toutefois, il peut être utile de définir des méthodes transversales applicables hors du contexte particulier d'un objet.
PHP 5 propose donc la définition de méthodes, dites statiques. Ces méthodes vont être utilisables sans qu'un objet ne soit nécessaire. La syntaxe utilisée est alors :
`nomdelaclasse::nomdelamethode();`

Héritage

Parmi les facilités traditionnellement associées au modèle objet, l'héritage figure en général en bonne place. La notion de classe permet déjà de disposer d'un moyen pour recréer des objets d'un même type. L'héritage va apporter une simplification supplémentaire dans la définition des classes elles-mêmes en autorisant la programmation par extension.

Ainsi, avec l'héritage, une classe fille pourra être définie comme l'extension d'une classe parente. On peut alors créer une véritable hiérarchie de classes

dérivant les unes des autres. Dans PHP Saloon, on pourra par exemple étendre notre classe `utilisateur` ou, pour simplifier la mise en œuvre de l'architecture MVC, définir une classe racine contrôleur et étendre celle-ci avec les fonctions spécifiques liées à chacun des modules (identification, inscription...).

PHP utilise le mot-clé `extends` pour associer une classe fille à sa classe parente. Ainsi, pour notre classe `utilisateur` on pourrait définir :

```
class utilisateurabonne extends utilisateur {
   private $numero_client;
   function lit_numero_client() {
      return $this->numero_client;
   }
   function change_numero_client($nouveau_numero) {
      $this->numero_client = $nouveau_numero;
   }
}
```

Dans ce cas, un objet de type `utilisateurabonne` comportera trois attributs privés et six méthodes distinctes. L'héritage, qui pourrait être simplement vu comme une facilité syntaxique (en évitant de redéfinir des éléments de classe en classe), apporte donc avant tout une cohérence globale entre les classes et constitue un facteur de réutilisation et de qualité des développements.

Il faut noter qu'en matière d'héritage, PHP 4 et PHP 5 sont totalement compatibles. Les hiérarchies de classes conçues pour PHP 4 devraient donc être fonctionnelles, et ce sans difficultés, avec PHP 5. Naturellement, aucune protection des données ne sera assurée.

REGARD DU DÉVELOPPEUR **Étendre n'est pas composer**

Quand on découvre le modèle objet et l'héritage, la tentation est grande de voir des relations d'héritage entre toutes les classes. Cette surutilisation du concept aboutit en général à un usage déformé du modèle ou de la notion d'héritage, tout en produisant l'effet contraire à celui attendu en matière de lisibilité et de clarté des développements.

Plus particulièrement, il est essentiel de ne pas confondre l'extension d'une classe en une nouvelle classe qui la précise, et la composition, qui consiste à intégrer dans cette classe des objets d'autres types.

Ainsi, une classe `berline` peut logiquement étendre une classe `voiture`. Mais ce serait une erreur de vouloir étendre la classe `roue` en lui ajoutant la classe dérivée `voiture`. Une `voiture` pourra par contre comporter un attribut `roues` (par exemple, un tableau de 4 objets de classe `roue`).

Cet exemple est trivial et ne prête pas à confusion, mais dans la réalité des développements, il n'en va pas toujours de même. C'est pourquoi il convient de se souvenir à chaque instant, qu'objet ne veut pas dire héritage. Cette modération est particulièrement bienvenue dans un langage comme PHP (ou C++ par exemple) dont le fondement reste procédural.

Héritage et interfaces

La mécanique d'héritage disponible sur les objets est aussi disponible sur les interfaces (voir le chapitre 2). Il est alors possible de définir, en parallèle avec la hiérarchie des classes et des objets, une hiérarchie des interfaces. Le terme d'extension revêt alors tout son sens, puisqu'il s'agit de définir, via les règles d'héritage, des extensions d'API.

```
class utilisateurabonne extends utilisateur implements iutilisateur,
iabonne {
   // code de la classe.
}
```

On note cependant que si pour les interfaces, comme pour les classes, l'héritage multiple n'est pas pris en charge, une classe d'objets peut implémenter plusieurs interfaces. On peut voir ces implémentations multiples comme une composition au niveau des interfaces, comme il est possible de composer les objets au sein d'une classe.

REGARD DU DÉVELOPPEUR **Héritage multiple**

Un sujet de discorde permanent consiste à se demander s'il doit être permis de définir des classes dérivées de plusieurs classes parentes. C'est la notion d'héritages multiples.

Dans cette hypothèse, la hiérarchie des classes se transforme en graphes ou est reliée à plusieurs arbres *a priori* distincts.

On peut alors, pour reprendre notre exemple des voitures, définir `berline` comme à la fois une extension de la classe `voiture`, mais aussi de la classe `produit_hautdegamme`. Dans ce cas, on relie une hiérarchie qui définit la nature de l'objet à une hiérarchie qui définit les produits.

Plus communément, l'héritage multiple sera utilisé pour faire de `voiture` l'extension de `roue`, `moteur` et `autoradio` ! Ce qui est une catastrophe en termes de conception.

Plusieurs langages ont choisi de ne pas introduire cette notion d'héritage multiple ; c'est le cas de PHP 5.

Classes abstraites et finales

Même si les interfaces ne représentent pas conceptuellement des classes objet, on peut toutefois les rapprocher d'un autre raffinement du modèle objet compatible avec PHP 5 : les classes abstraites.

De telles classes ne peuvent être instanciées, c'est-à-dire que toute tentative de création d'un objet de ce type est impossible. Il s'agit de classes qui décrivent en général des éléments génériques dont seules les classes dérivées sont en réalité exploitables.

Dans PHP Saloon, on pourrait par exemple revoir notre hiérarchie de classe en définissant `utilisateur` comme une classe abstraite dont pourrait ensuite dériver une classe « normale » : `utilisateur_phpsaloon`. Tout en n'étant pas

instanciable, la classe `utilisateur` pourrait définir les bases imposées d'un type objet `utilisateur`, bases qui seraient alors reprises et complétées par héritage.

On le voit, il est possible d'aboutir à un résultat approchant en imposant une interface à la classe `utilisateur_phpsaloon`. C'est l'approche choisie pour toutes nos classes. Il faut néanmoins remarquer que dans le cadre des classes abstraites, il est possible d'imposer plus que l'API, puisqu'une classe abstraite peut définir des attributs.

Plusieurs manières de rendre une classe abstraite

PHP 5 permet de déclarer une classe abstraite de manière directe en utilisant le mot-clé `abstract` :

```
abstract class utilisateur {
}
```

Par ailleurs, différentes méthodes de la classe peuvent elles-mêmes être déclarées abstraites :

```
abstract class utilisateur {
   abstract function connecte() {
   }
}
```

Dans ce cas, il ne suffira pas de construire une classe dérivée, mais encore faudra-t-il que celle-ci implante une véritable méthode `connecte` pour que la création d'objets soit possible.

À l'opposé des classes abstraites, non instanciables mais extensibles, PHP 5 intègre aussi la notion de classe finale. Comme le laisse entendre ce terme, de telles classes ne pourront pas être étendues. Elles constitueront en quelque sorte des feuilles dans l'arbre des classes.

Une classe, comme une méthode, peut être déclarée finale. On pourra donc interdire au développeur, ou par exemple aux utilisateurs d'un composant, de redéfinir une méthode clé :

```
class utilisateur {
   final function connecte() {
      // ...
   }
}
```

Polymorphisme

Jusqu'à présent, l'héritage nous a permis d'étendre les classes en complétant les méthodes et les attributs disponibles. Cependant, PHP 5 autorise également les classes dérivées à redéfinir les méthodes existantes. C'est le polymorphisme. Ainsi, non seulement des objets dérivés disposeront de fonctionnalités additionnelles, mais encore pourront-ils prendre des formes différentes dans leur comportement.

> À RETENIR **Le polymorphisme n'est pas la surchage**
>
> On confond parfois la redéfinition de certaines méthodes dans les cas du polymorphisme et de la surchage de méthodes. La surcharge consiste à définir des méthodes distinctes en utilisant le même nom mais des prototypes différents. À ce jour, PHP 5 ne supporte pas la surcharge.
>
> Le polymorphisme consiste pour sa part à remplacer une méthode par une autre. Il s'agit donc de bien plus qu'une adaptation comportementale. Le prototype demeure inchangé.

Une méthode pourra réaliser un traitement adapté à chaque classe d'objets tout en conservant la même interface.

Dans PHP Saloon, on pourrait adapter les méthodes entre l'utilisateur abonné et l'utilisateur standard :

```
class utilisateur {
   function information() {
      return "Utilisateur invité";
   }
}
class utilisateur_abonne extends utilisateur {
   function information() {
      return "Votre êtes abonné(e) jusqu'au {$this->expiration}.";
   }
}
```

Quel que soit l'objet `utilisateur` créé, il devient possible d'utiliser la méthode `information()` sans se préoccuper de la nature exacte de l'objet disponible. La méthode adaptée est automatiquement utilisée.

Pour éviter les débordements, PHP permet toutefois de contrôler la nature des objets manipulés : on parle de « type hinting ». Il s'agit en réalité d'indiquer à PHP la nature du type attendu en paramètre par une fonction. Cette fonctionnalité, conjuguée au polymorphisme, donne un code puissant mais contenant des garde-fous.

En reprenant notre exemple précédent, on peut définir la fonction `detail_utilisateur()`. Celle-ci est à même d'afficher un récapitulatif des informations liées à l'utilisateur :

```
function detail_utilisateur($u) {
   // ...
   print $u->information();
   // ...
}
```

Une telle fonction ne posera aucun inconvénient tant que le paramètre requis est bien un utilisateur, abonné ou non. Les choses n'iront pas de même, si un entier est malencontreusement utilisé comme paramètre.

Avec le *type hinting*, PHP 5 va seul prendre en charge le contrôle de la nature de l'argument effectivement passé à la fonction. On peut alors améliorer la fonction `detail_utilisateur()` :

```
function detail_utilisateur(utilisateur $u) {
   // ...
   print $u->information();
   // ...
}
```

Désormais, tout élément non conforme sera rejeté par PHP lui-même. Cette notion complète parfaitement le polymorphisme puisque l'objet passé en paramètre n'est en aucun cas modifié. Ainsi, un utilisateur abonné validera la condition (il s'agit bien d'un utilisateur) sans pour autant être tronqué de ses propriétés étendues et l'appel de la fonction `detail_utilisateur()` affichera comme souhaité la date d'expiration de l'abonnement.

À RETENIR **Un mot sur la généricité et les classes templates**

Rappelons qu'il s'agit de définir des classes paramétrées par des données ou des types dont la nature précise ne sera connue qu'au moment de la création des objets. Le principal usage de ces classes génériques est d'implanter des algorithmes (opérations sur les piles et les files, ou encore matricielles).

PHP 5 ne propose rien en la matière. Il est vrai que PHP n'étant pas typé, l'intérêt de telles possibilités est moindre.

Constructeurs et destructeurs

Avec l'héritage, le polymorphisme et les classes abstraites PHP 5 permettent de définir des hiérarchies de classes et d'objets complexes. Cependant, pour le moment, l'ensemble des objets manipulés sont créés vierges de toute initialisation.

Il est néanmoins tout à fait possible de définir dans chaque classe une méthode ayant en charge des opérations d'initialisation. L'inconvénient majeur dans ce procédé est le côté manuel et besogneux. Pour chaque objet, et pour chaque classe à laquelle il appartient, il faudra prendre soin d'appeler les fonctions d'initialisation ad hoc.

Par chance, PHP 5, comme tous les langages objet, propose la notion de constructeur. Le constructeur joue très exactement le rôle de la fonction d'initialisation précédente. Mais à la différence de celle-ci, il sera automatiquement appelé par PHP.

Le nom de ce constructeur est normalisé par PHP et il s'agit :

- soit de `__construct`, notez les deux caractères soulignés (*underscore*) « _ » préfixant le mot `construct` (PHP 5) ;
- soit du nom de la classe elle-même (PHP 4/PHP 5).

Dans le cas de notre classe utilisateur, on pourrait par exemple écrire :

```
class utilisateur {
   private $pseudo;
   private $motdepasse;
   private $age;
   function __construct($pseudo = false, $motdepasse = false,
                        $age = 18) {
      $this->pseudo = $pseudo;
      $this->motdepasse = $motdepasse;
      $this->age = $age;
   }
   // ...
}
```

Lors de la création d'un objet, les différents attributs sont alors automatiquement initialisés, soit avec les valeurs par défaut, soit avec des valeurs fournies lors de la création :

```
$u = new utilisateur('PHPFan', 'Secret', 32);
```

Notre exemple est particulièrement simple, la classe `utilisateur` ne dérive d'aucune classe parente. Considérons maintenant notre classe `utilisateur_abonne`, il est tout aussi possible de définir un constructeur :

```
class utilisateur_abonne extends utilisateur {
   function __construct() {
      // ...
   }
   // ...
}
```

Dans cette situation, et cela peut surprendre de prime abord, la création d'un utilisateur abonné va provoquer :

- l'appel du constructeur de cette même classe ;
- mais non l'appel au constructeur de la classe utilisateur.

Pourquoi un tel comportement ? Pour une raison en réalité très simple. PHP ne peut savoir à la place du développeur comment appeler, et avec quels paramètres, le constructeur de la classe parente, à supposer d'ailleurs qu'il faille appeler celui-ci ! Il appartient donc au développeur de choisir explicitement d'appeler ou non le constructeur parent :

```
class utilisateur_abonne extends utilisateur {
   function __construct() {
      // ...
      parent::__construct();
   }
   // ...
}
```

La notation utilisée reprend la syntaxe adoptée pour les méthodes statiques, avec comme nom de classe, le mot parent.

La notion de constructeur peut naturellement être mise en parallèle avec le concept de destructeur. Alors que le constructeur est appelé lors de la création de l'objet, le destructeur est appelé, lui aussi automatiquement par PHP, lorsque l'objet doit être détruit.

Le nom du destructeur est lui aussi normalisé dans PHP 5 : `__destruct`. Il faut noter que la notion de destructeur n'est pas présente en PHP 4.

PHP gère lui-même la mémoire et dispose d'un ramasse-miettes (*garbage collector*) intégré ; l'utilisation d'un destructeur est moins courante que celle d'un constructeur. Cependant, certaines informations, notamment les ressources créées par des extensions, comme les images, doivent être libérées après utilisation. Le recours à un destructeur est alors indispensable.

PHP 4 Utilisation des constructeurs

La syntaxe désormais mise en avant par PHP 5 diffère de celle utilisée jusqu'à présent. Il est encore possible d'utiliser la notation en vigueur dans PHP 4, même si celle-ci est déconseillée. Les classes définies pour PHP 4 ne devraient donc pas poser problème.
À noter, il n'existe pas de destructeurs en PHP 4.

Utilisation des objets et références

Dans PHP, qu'il s'agisse de PHP 4 ou de PHP 5, les données sont copiées quand elles sont affectées ou passées en paramètres. Ce fonctionnement est naturel.

Toutefois, ce qui est naturel pour un tableau ou un entier, peut tourner au désastre pour un objet. D'abord en raison de la taille des objets, même si cet argument est tout autant opposable aux tableaux et aux chaînes de caractères, mais surtout en raison même de l'usage qu'on entend en général faire des objets.

Le plus souvent, on manipulera un objet comme un élément unique, sur la durée, avec en corollaire, la conservation d'un état des lieux propre. Cette continuité est naturellement brisée par les opérations de recopie. L'objet copié n'étant alors plus synchrone avec une copie qui, par ailleurs, ne disposera pas forcément de la même durée de vie.

Une solution consiste à demander explicitement à PHP de ne pas copier les données mais de transmettre directement une référence vers l'objet original. On utilise pour cela le symbole & (à la fois à l'affectation et lors de la définition de fonctions).

```
$reference =& $objet_original;
function foo(&$objet) { // le passage se fera par référence
   // ...
}
```

Ce procédé est très pénible et sujet à de multiples erreurs et oublis. PHP 5 adopte donc une autre philosophie, par ailleurs déjà adoptée par de très nombreux langages.

Dans PHP 5, lorsqu'un objet est créé, ce n'est plus l'objet lui-même qui est retourné, mais une référence vers celui-ci. Cela peut paraître complexe mais il n'en est rien. L'interpréteur PHP tient en effet à jour un entrepôt de tous les objets en cours d'utilisation et chaque objet se voit numéroté. La référence de l'objet est donc toute trouvée, il s'agit de ce numéro. C'est lui qui est manipulé partout.

Tout est alors plus simple. Alors qu'il fallait le plus souvent inutilement copier l'ensemble des propriétés d'un objet, on ne copie plus aujourd'hui que ce numéro de référence. Ce numéro est suffisant pour accéder si nécessaire aux informations et aux méthodes.

Ce numéro n'est pas un secret. Ainsi, en reprenant notre classe `utilisateur`, il est très simple de connaître la référence des objets créés avec un simple `print` :

```
$u = new utilisateur();
print $u;
```

On obtient :

```
Object id #4
```

Ceci nous confirme que nous manipulons l'objet numéro 4 (ce qui, manifestement, doit, à peu de choses près, signifier que `$u` est le quatrième objet créé).

Ce changement de mode de fonctionnement apporte de nombreux avantages, à la fois en termes de vitesse (moins de recopies) et de souplesse (moins de mémoire occupée), mais il ne sera pas sans poser difficulté aux utilisateurs de scripts et d'applications conçues pour PHP 4.

PHP 4 La grande différence

On le voit, le mécanisme mis en œuvre dans PHP 5 est à l'opposé du mode par copie disponible dans PHP 4. Même si tout est fait pour minimiser l'impact de ce changement de fond, certains objets ne fonctionneront pas avec PHP 5, notamment ceux qui reposaient (parfois sans le savoir) sur la copie implicite des objets lors de chaque opération (affectation, passage en paramètre).

Pour toucher du doigt la différence essentielle entre les deux versions du langage, considérons la classe (triviale) suivante :

```
class test {
   var $attribut;
}
```

Et une fonction tout aussi élémentaire qui modifie l'objet :

```
function foo($objet) {
   $objet->attribut = 'bar';
}
```

Dans PHP 4, la copie ou le passage en paramètre provoquent une copie de l'objet. Ainsi le code suivant :

```
$o1 = new test();
$o1->attribut = 'original';
$o2 = $o1;
$o2->attribut = 'bar';
print $o1->attribut;
$o1->attribut = 'original';
foo($o1);
print $o1->attribut;
```

Va afficher :

```
originaloriginal
```

Dans PHP 5, les objets ne sont manipulés que par référence, et cela par défaut. On obtient au contraire :

```
barbar
```

Cette différence a constitué un handicap majeur pour l'adoption des objets dans PHP 4. Dans une majorité de cas, il fallait user et abuser de l'opérateur & pour forcer l'utilisation de références. Ainsi, le code suivant se comporte en PHP 4 comme en PHP 5 :

```
<?
class test {
   var $attribut;
}
function foo(&$objet) {
// $objet sera passé par référence
   $objet->attribut = 'bar';
}
$o1 = new test();
$o1->attribut = 'original';
$o2 =& $o1;   // $o2 est une référence vers $o1
$o2->attribut = 'bar';
print $o1->attribut;
$o1->attribut = 'original';
foo($o1);
print $o1->attribut;
?>
```

Il est toutefois possible de placer PHP 5 en mode compatibilité, en configurant l'option `zend.implicit_clone` dans `php.ini`. Néanmoins, cette option entrave naturellement tout usage normal du modèle objet de PHP 5.

Enfin, il faut noter que l'utilisation systématique de références dans PHP 5 devrait permettre (outre les optimisations du moteur Zend) au code objet écrit pour PHP 4 de s'exécuter plus rapidement.

Autres facilités introduites par PHP 5

En complément du modèle objet, PHP 5 apporte un lot de fonctionnalités techniques visant à gagner en rapidité et à diminuer la quantité de code développé.

Méthodes et attributs dynamiques

Premier ensemble de fonctions : le support à la volée d'attributs et de méthodes. Traditionnellement, l'appel à une méthode inexistante, comme l'accès à des attributs non définis, provoque une erreur et termine l'exécution du script PHP.

PHP 5 offre un moyen de contourner ce problème en permettant de définir des *hooks* ou des *handlers* qui seront appelés en cas d'échec. Ces méthodes très particulières répondent à des signatures prédéfinies par le langage et peuvent être définies dans chaque classe :

```
function__call($fonction_appelée, $arguments);
function__set($attribut, $valeur);
function__get($attribut);
```

Ainsi, dans le cadre de la classe utilisateur, il est possible de remplacer les différentes méthodes par deux implantations adaptées de `__get` et `__set` :

```
<?
class utilisateur {
   private $pseudo;
   private $motdepasse;
   private $age;

   function __get($nom) {
      switch($nom) {
         case'login':
            return $this->pseudo;
         case'password':
            return $this->motdepasse;
      }
   }
   function __set($nom, $value) {
      switch($nom) {
         case'login':
            $this->pseudo = $value;
            break;
         case'password':
            $this->motdepasse = $value;
            break;
      }
   }
}
```

PHP 4 Un portage expérimental

Si PHP 4 sortait encore son épingle du jeu en matière d'héritage et d'encapsulation, les fonctionnalités suivantes sont clairement conçues pour PHP 5 même si un portage est disponible pour PHP 4. il faut dans ce cas explicitement les activer au cas par cas avec la méthode `overload()`.
Même dans cette hypothèse il faut s'attendre à quelques désagréments. D'une part la compatibilité n'est que partielle, il faudra notamment penser à retourner systématiquement la valeur true dans les fonctions `__get()` et `__set()`. D'autre part, l'expérience montre que la stabilité n'est pas toujours au rendez-vous.

```
$u = new utilisateur();
$u->login= 'PHPFan';
print $u->login;
?>
```

Il faut toutefois noter que ces différentes méthodes ne sont appelées que pour les attributs non existants. Ainsi, le code suivant va échouer :

```
$u->age = 32;
```

En effet, l'attribut `age` existe et a été déclaré privé.

Chargement automatisé des classes utilisées

Autre élément de simplification : `__autoload()`. Traditionnellement, et plus encore avec l'utilisation des objets, on est amené dans les développements à inclure des fichiers complémentaires. Par exemple, il est possible d'associer au code de chaque classe un fichier dédié.

Il devient alors indispensable de n'oublier aucune des directives `include` ou `require`. Cette obligation, simple en apparence, reste délicate en termes de maintenance et pendant la phase de développement.

Pour améliorer les choses, PHP 5 offre de définir une fonction, la fonction `__autoload()`, qui sera appelée chaque fois que la création d'un objet est impossible, parce que la classe demandée est introuvable. À charge de la fonction `__autoload()` d'agir en conséquence, en incluant le code requis la plupart du temps.

```
function __autoload($classe) {
   require_once'inc/'. $classe . '.php';
}
```

Ce mécanisme peut sensiblement simplifier la gestion de la modularité, mais attention, il requiert néanmoins un peu de rigueur dans l'organisation du code et des noms de fichiers. En effet, la fonction `__autoload()` ne dispose que du nom de la classe pour reconstituer le chemin des fichiers à inclure.

Clonage

Dernière facilité proposée par PHP 5 : le clonage. Il est toujours possible de créer un objet avec l'instruction `new`. Cette manière de faire ne permet pas toutefois d'obtenir deux objets identiques après coup. En effet, rien ne permet de tenir compte des évolutions subies par un objet, si ce n'est de les reproduire toutes, et dans les mêmes conditions. PHP 5 propose donc l'instruction `clone` :

```
$objet_clone = clone $objet_original;
```

REGARD DU DÉVELOPPEUR `__autoload()`, oui mais... sans espaces de noms

Les habitués du langage Java ou même de Perl resteront sans aucun doute sur leur faim avec `__autoload()` tant il est vrai que PHP 5, sur ce point, reste en retrait des attentes habituelles.
Sans altérer en rien la simplicité d'usage de PHP, on aurait aimé que PHP 5 soit l'occasion d'offrir un environnement non seulement adapté à la programmation objet (sur ce point l'objectif est atteint) mais aussi propice à la création de frameworks ou de composants, comme d'ailleurs Microsoft a su le faire avec Visual Basic en entourant un langage simplissime d'outils de compilation, et de toute la mécanique nécessaire pour attirer les développeurs de composants.
Si l'on en revient à Java, PHP 5 aurait notamment beaucoup gagné à supporter les namespaces et une instruction du type `import`. Ces apports faisaient d'emblée disparaître les conflits de nommage et permettaient d'envisager le développement de composants à grande échelle.

Pour prendre en charge de manière personnalisée la duplication d'objets, une classe devra définir une méthode `__clone()`. Cette méthode assurera la copie intelligente des éléments internes d'un objet.

Dans le cas de notre classe utilisateur, la méthode `__clone()` n'aurait qu'à copier l'ensemble des attributs et serait donc inutile. Dans la pratique, il faudrait aussi rouvrir des connexions vers les bases de données ou les fichiers...

```
<?
class utilisateur {
   private $pseudo;
   private $motdepasse;
   private $age;
   private $db;
   function __construct($pseudo, $motdepasse, $age = 33) {
// exemple d'une ressource non copiable simplement
      $this->db = new sqlite_db('test');
   }
   function __clone() {
      // les attributs ont déjà été copiés!
      $this->db = new sqlite_db('test');
   }
}
$o = new utilisateur('PHPFan', 'secret');
$o2 = clone $o;
?>
```

Notre exemple va permettre à chaque objet (celui cloné et l'original) de disposer de sa propre connexion à la base de données.

Les lecteurs les plus perspicaces l'auront probablement deviné, ce cas est quelque peu caricatural car PHP, gérant seul la durée de vie des objets, serait en mesure de constater si la connexion créée par le premier objet est en réalité également utilisée par le deuxième. On pourrait donc penser qu'une simple copie des attributs est suffisante, malgré tout.

À RETENIR Clonage implicite

Pour cloner un objet, il n'est pas forcément indispensable de définir sa propre fonction `__clone()`. Dans ce cas, PHP fait une copie à l'identique des attributs. Cette méthode brutale marchera tant que des éléments complexes, du type ressources ou descripteurs de fichiers, ne sont pas mis en œuvre.

PHP 4 Pas de clonage

Dans PHP 4, la notion de clonage n'est pas disponible, mais comme les objets sont copiés à chaque affectation (ou passage en paramètre), on peut considérer qu'un clone implicite est réalisé dans ces situations.
Ces copies laissent malheureusement à la charge du développeur le traitement des ressources (fichiers ou connexions), sur toute la hiérarchie des classes.

La classe utilisateur complète

La classe utilisateur finalement retenue pour PHP Saloon ne requiert naturellement pas la mise en pratique de l'ensemble des fonctionnalités désormais proposées par PHP 5. Trois méthodes sont disponibles :

- `connecte()` qui va activer une session pour l'utilisateur ;
- `nouveau()` qui va inscrire l'utilisateur au service ;
- `id()` qui permet d'obtenir l'identifiant de l'utilisateur déjà inscrit.

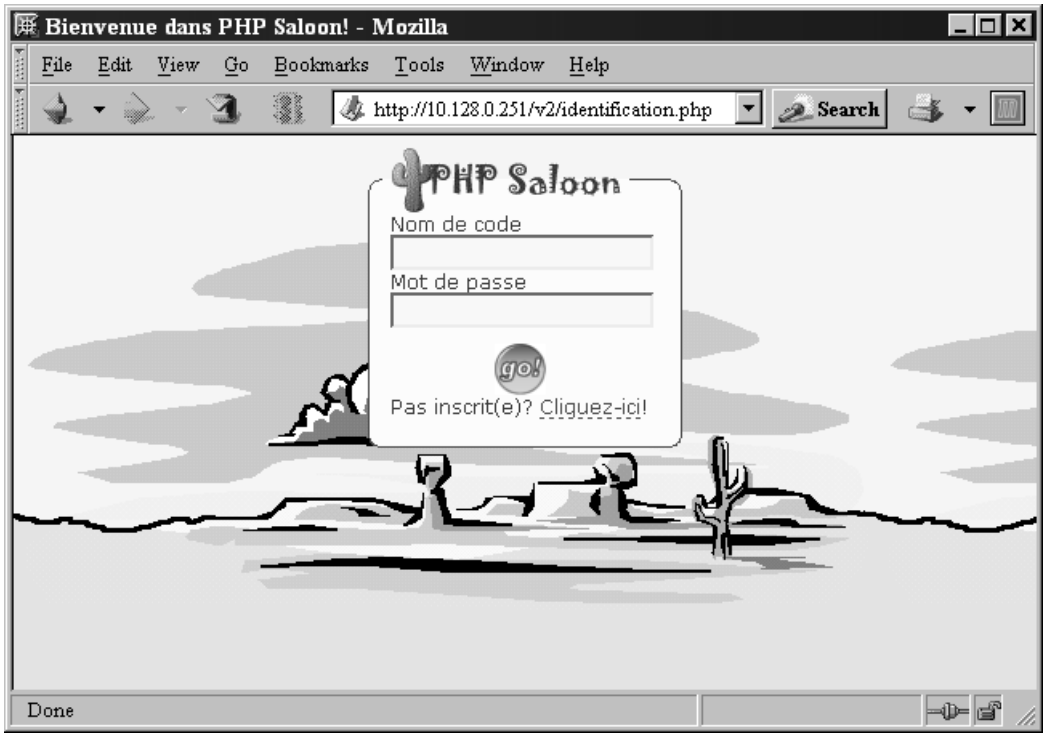
```
<?
require_once'session.php';
```

```
class utilisateur {
   private $db;
   private $id;
   private $pseudo;
   private $motdepasse;
   private $age;

   function __construct($pseudo = false,$motdepasse = false, $age = 18)
  {
      $this->pseudo = $pseudo;
      $this->motdepasse = $motdepasse;
      $this->age = $age;
   }
   function connecte($db) {
      // $db est une connexion déjà ouverte vers la base SQLite
      $id = $db->single_query("select id from utilisateurs
         where pseudo = '{$this->pseudo}' and
               motdepasse= '{$this->motdepasse}'");
      if ($id) {
         $session = new session(120);
         $_SESSION['uid'] = $id;
         $session->update();
         returntrue;
      }
      returnfalse;
   }
   function nouveau($db) {
      $resultat = $db->query($query = "insert into utilisateurs values
( NULL, '{$this->pseudo}', '{$this->motdepasse}', '{$this->age}')");
      if ($resultat)
         return $this->id = $db->last_insert_rowid();
      returnfalse;

   }
   function id() {
      if ( ! $this->id)
         $this->id = $_SESSION['uid'];
      return $this->id;
   }
}
?>
```

En résumé...

Alors que PHP 4 souffrait d'un support objet limité et poussif, PHP 5 apporte l'ensemble des éléments indispensables à la conception d'applications de grande envergure. De plus, et dans la logique de simplification constante du langage, PHP 5 complète les outils de base d'une multitude d'aménagements qui sont propres à rendre le développement encore plus simple et efficace.

chapitre 5


Bienvenue dans PHP Saloon! - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://10.128.0.251/v2/identification.php
Search
PHP Saloon
Nom de code
Mot de passe
go!
Pas inscrit(e)? Cliquez-ici!
Done

Sessions

La création de sessions utilisateur est essentielle car elle ouvre la voie à la personnalisation d'un site en fonction du visiteur en cours.
Après avoir démystifié les mécanismes mis en œuvre pour créer les sessions et étudié leur implantation dans PHP, nous expliquerons le code de la solution retenue pour PHP Saloon.

SOMMAIRE

MOTS-CLÉS

- Session
- Cookie
- Header
- HTTP

Incontournables sessions

Le visiteur d'un site se limite rarement à une page. Il navigue d'un endroit à un autre en suivant les liens proposés. Cet état de fait est particulièrement significatif pour l'accès à un service sur abonnement.

Le problème qui se pose alors est de savoir reconnaître le visiteur lors de son périple au sein du site. Cela peut paraître évident à première vue mais il n'en est rien. Pour s'en convaincre, il faut revenir aux échanges entre le navigateur du visiteur et le site.

Ces échanges sont pris en charge par le protocole HTTP (HyperText Transfer Protocol). Le navigateur va effectuer une demande en utilisant ce protocole pour tout élément des pages visitées (le texte, chaque image, les feuilles de styles ou encore les éventuelles applets Java).

D'une requête HTTP à l'autre, toutes les informations transmises par le navigateur sont oubliées. On parle de protocole sans état. L'avantage de tels protocoles est leur simplicité, ce qui les rend simples à implanter et donc moins coûteux à utiliser (ce qui explique leur succès dans le monde Internet).

Au niveau du serveur web (qui répond aux demandes HTTP), on ne peut donc pas savoir si plusieurs demandes émanent de la même personne. Il en va de même pour notre code PHP.

Tout l'objectif des sessions est donc de marquer le visiteur de manière unique lors de sa première demande et d'assurer que cette marque persiste pour les demandes suivantes. Il sera alors possible de sauvegarder les différentes données (on parle de profil) associées au visiteur et à partir de celles-ci, de personnaliser les pages proposées.

La période pendant laquelle le visiteur est suivi à la trace par ce marquage constitue une session. Bien entendu toute session expire après un moment d'inactivité car il serait impossible de conserver pour une durée illimitée les marquages de chaque visiteur rencontré.

Quand un visiteur revient après l'expiration de sa session ou s'il se présente lors d'une nouvelle visite sur le site, il est à nouveau marqué et bénéficie d'une nouvelle session. Il appartient alors au site de retrouver les données de la session précédente, faute de quoi le visiteur ne bénéficiera que des données de la nouvelle session.

Cette opération de restauration n'est possible que si l'utilisateur dispose d'une identification permanente auprès du site. C'est pourquoi l'utilisation des sessions (c'est-à-dire le marquage des visiteurs pendant une séance de navigation) est le plus souvent complétée par une inscription préalable qui permet une identification sur la durée. Les données de la session en cours sont alors sauvegardées et associées de manière permanente à ce visiteur, qui

ALTERNATIVE Identifier les visiteurs autrement

Si les sessions se sont imposées comme moyen d'identifier les visiteurs de manière unique, c'est qu'il existe peu d'alternatives aussi efficaces.
Ainsi, l'identification de l'utilisateur par son adresse IP se heurte aux pare-feux qui masquent en général, derrière une adresse commune, plusieurs utilisateurs.
L'authentification proposée par le protocole HTTP est pour sa part plus adaptée à la protection de répertoires entiers qu'au suivi des utilisateurs. De plus, la gestion en est laissée au navigateur, ce qui pose problème dans le cas des postes partagés ou en libre service.

Et dans PHP Saloon ?

PHP Saloon est un service accessible après inscription. L'utilisateur doit donc s'identifier et être reconnu pendant toute son utilisation du service. Les sessions sont donc impératives.

s'est identifié. Lors d'une seconde visite, il lui sera possible de réassocier les données sauvegardées à la nouvelle session créée.

ALTERNATIVE **Des sessions anonymes, sans inscription**

La plupart du temps, les sessions sont utilisées pour des services auxquels l'utilisateur s'inscrit régulièrement. Mais les sessions sont aussi un bon moyen de déterminer le profil comportemental du simple visiteur.

Dans cette optique, il n'est pas nécessaire que celui-ci soit un abonné au service (même si dans ce cas le croisement d'informations peut être plus fructueux).

De nombreuses sociétés proposent sur cette base d'analyser la fréquentation des sites en répondant à des questions comme : combien de temps le visiteur passe-t-il sur une page, quel est son cheminement au sein du site, quelles sont les pages les plus visitées et dans quel ordre...

Les outils proposés par PHP permettent de simplifier

On le voit, le support des sessions comporte plusieurs aspects bien distincts :

- le fait de « marquer » le visiteur ;
- la sauvegarde des données du visiteur (comportement, profil) pendant la durée de vie de la session ;
- la sauvegarde permanente intersessions du profil d'un visiteur.

PHP propose de simplifier la gestion des deux premiers points, le troisième restant à la charge du webmaster du site. Ce qui est naturel puisqu'il appartient au concepteur du site de déterminer comment un visiteur doit s'inscrire, s'authentifier et comment les données doivent être sauvegardées (fichier, base de données).

PHP 4 **Les sessions dans PHP 4 et PHP 5**

C'est la même chose ! Si vous avez développé vos propres fonctions d'authentification, ou si vous utilisez un script PHP conçu à l'origine pour PHP 4, vous n'aurez donc rien à changer.

Création et maintien de la session

Pour suivre le visiteur d'un site, PHP propose d'associer un identifiant unique à sa session. Celui-ci sera associé au visiteur pendant tout son parcours jusqu'à l'expiration de la session. PHP garantit que cet identifiant sera unique.

En pratique, le suivi de session doit être activé avec l'instruction : `session_start()`. À compter de cette activation, l'identifiant de session est disponible soit en utilisant la fonction `session_id()`, soit via la constante `SID`.

CONFIGURATION **Activation automatique des sessions**

Si votre site fait un large usage des sessions, il peut devenir pénible de devoir ajouter l'instruction `session_start()` dans chaque page.
PHP peut démarrer la session pour vous automatiquement. Cette possibilité est contrôlée via une directive spécifique dans le fichier de configuration de PHP (`php.ini`) :

```
session.auto_start = on;
```

Cette option est désactivée par défaut.

La génération par PHP d'un identifiant unique est cruciale – le serveur répondant à de nombreuses requêtes simultanées, la tâche est en effet complexe. Voyons comment assurer son maintien tout au long de la session.

PHP met en place les mécanismes pour assurer cette transmission de page en page de manière (presque) totalement transparente pour le développeur du site. Deux techniques sont utilisées :

- l'utilisation d'un cookie ;
- la réécriture automatique des liens et des paramètres de formulaires.

Dans la plupart des cas, PHP utilise un cookie pour stocker l'identifiant de session directement sur l'ordinateur du visiteur. Le cookie ainsi stocké par le navigateur sera représenté au serveur web pour toute demande ultérieure. Cette méthode est préférable pour mettre en œuvre les sessions. Hélas, elle peut échouer si le navigateur ne supporte pas les cookies ou si ceux-ci ont été désactivés pour des raisons de sécurité.

DANS LA VRAIE VIE **Cas des fermes de serveurs ou de répartition sur plusieurs serveurs**

La création d'un identifiant unique est déjà une opération complexe pour un serveur, la chose se complique lorsqu'il s'agit d'assurer qu'un tel identifiant sera unique et accessible sur plusieurs serveurs.

Cette situation est notamment celle rencontrée lorsqu'on répartit la charge des requêtes sur plusieurs serveurs et, par extension, dans les fermes de serveurs.

Le gestionnaire de sessions standard proposé par PHP n'est pas utilisable dans ces configurations (chaque serveur va retourner un identifiant différent pour des requêtes potentiellement associées au même visiteur. Cependant plusieurs extensions sont disponibles qui permettent de résoudre le problème (on pourra citer l'extension `msession`).

COMPRENDRE **Les cookies**

Le cookie est en quelque sorte une variable (un nom associé à une valeur) qui va être échangée entre le navigateur et le serveur (et donc aussi PHP) lors du parcours de pages web. Pour mieux comprendre le fonctionnement des cookies, rappelons très brièvement le fonctionnement du protocole HTTP, utilisé pour accéder aux sites et aux pages web.

Le protocole HTTP spécifie que le navigateur (client) souhaitant accéder à un élément (page, image), établit une connexion avec le serveur du site visité et lui envoie une requête pour obtenir du serveur l'information demandée. Chaque élément qui compose une page fait ainsi l'objet d'une séquence demande/réponse .

À ce stade, il n'est donc pas encore question de cookie. Cependant, outre la demande et la réponse, le navigateur et le serveur vont s'échanger des informations complémentaires. On parle de *head*er (en-tête) car dans le cas du serveur, ces informations sont présentées avant le contenu de la réponse (une image par exemple).

Le navigateur peut ainsi préciser au serveur qu'il préfère les documents en français, qu'il sait ou non interpréter les images JPEG et GIF, qu'il dispose ou non d'un plug-in Flash et naturellement indiquer sa nature (Mozilla, Safari ou Internet Explorer). Ces informations précieuses seront exploitées par le serveur pour délivrer le contenu le plus adapté.

De son côté, le serveur délivre des en-têtes au navigateur, ne serait-ce que pour indiquer le type du document ou préciser si celui-ci peut être conservé dans un cache.

Que vient faire le cookie dans ces échanges ? Il s'agit tout simplement d'un en-tête supplémentaire que le serveur ajoute et qui sera présent dans les échanges entre le navigateur et le serveur. Ce cookie est sauvegardé par le navigateur qui l'ajoute ensuite à ses propres en-têtes chaque fois qu'une demande concerne le site émetteur du cookie (un cookie n'est disponible que pour le site qui l'a créé). Le cookie ainsi retransmis peut être mis à jour par le serveur.

Notons enfin, que si le cookie peut être grossièrement comparé à une variable partagée, il dispose de propriétés supplémentaires qui permettent de déterminer son expiration ou sa portée.

PHP propose donc une méthode alternative qui repose sur une modification à la volée du code produit par les pages PHP. Il va notamment ajouter une variable supplémentaire – `PHPSESSID` – aux adresses utilisées dans les liens ou dans les formulaires. La variable ainsi ajoutée va transporter l'identifiant de session. Cette opération est transparente, PHP modifie le contenu de la page automatiquement avant que celle-ci ne soit envoyée au navigateur du client.

Il s'agit naturellement d'une opération plutôt lourde et moins sûre que l'utilisation des cookies, puisque l'identifiant de session est accessible et peut être transmis par erreur à des tiers par l'utilisateur lui-même. Cependant, cette méthode fonctionne dans tous les cas et avec tous les navigateurs.

Dans les versions récentes de PHP, cette option est désactivée par défaut. Vous pouvez la réactiver en modifiant les directives suivantes dans le fichier de configuration de PHP :

```
session.use_only_cookies = no
session.use_trans_sid    = yes
```

MÉTHODE **Contraintes d'utilisation pour `session_start()`**

On le voit, les cookies sont la méthode universellement utilisée pour stocker l'identifiant de session. Ceci ne va pas sans contrainte. En effet, cette méthode impose d'activer les sessions avant la production de la page. La valeur du cookie doit être transmise au navigateur avec tous les en-têtes du protocole HTTP, c'est-à-dire avant le contenu de la page.

De fait, on doit toujours placer l'instruction `session_start()` avant tout le reste dans une page, en prenant soin de vérifier qu'aucun espace ne s'est glissé dans le code HTML environnant, notamment si des fichiers sont inclus.

Voici un exemple classique où l'appel à l'instruction `session_start()` va échouer. Le code de la page PHP est particulièrement simple :

```
X
<? session_start(); ?>
<html>
<head><title>Page de test</title></head>
<body>
Une page classique
</body>
</html>
```

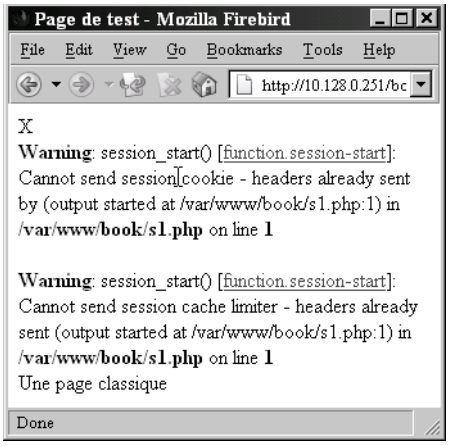


Figure 5–1 Caractères parasites causant l'échec de l'activation d'une session

Dans le cas présent, l'erreur est aisément détectable. Cependant, les espaces indésirables sont bien moins faciles à repérer, surtout si l'appel à `session_start()` est réalisé dans des fichiers inclus.

Il n'est pas difficile d'expérimenter les sessions. C'est ce que nous allons faire avec un site élémentaire, constitué uniquement de deux pages : `p1.php` et `p2.php`. Cette expérimentation nous permettra de maîtriser la situation pour PHP Saloon.

Figure 5–2 Activation de la session sur la première page

p1.php

```
<? session_start(); ?>
<html>
<head>
  <title>Page 1</title>
<head>
<body>
  <p> Nous avons obtenu notre identifiant de session :
      <? print session_id(); ?>
  </p>
  <ul>
   <li><a href="p2.php">Lien vers la page 2</a></li>
  </ul>
</body>
</html>
```

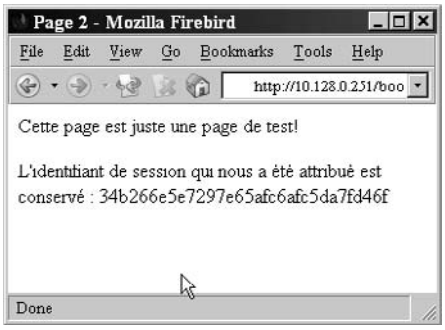


Figure 5–3 Transmission de l'identifiant à la deuxième page

p2.php

```
<? session_start(); ?>
<html>
<head>
  <title>Page 2</title>
<head>
<body>
  <p>Cette page est juste une page de test!</p>
  <p>L'identifiant de session qui nous a été
  attribué est conservé : <? print session_id(); ?>
  </p>
</body>
</html>
```

La page `p1.php` comporte un lien vers `p2.php` et toutes les deux activent le support des sessions. Vous pourrez vérifier que l'identifiant de session est bien transmis d'une page à l'autre et un cookie créé (cookie dont le détail du contenu est observable via l'interface ad hoc proposée par votre navigateur).

Et dans PHP Saloon ?

Notre exemple est effectivement très simple, mais PHP Saloon reposera sur les mêmes mécanismes. Il nous faudra maintenir la session entre la page listant les connectés, celle listant les amis et enfin celle permettant de lire les messages reçus.

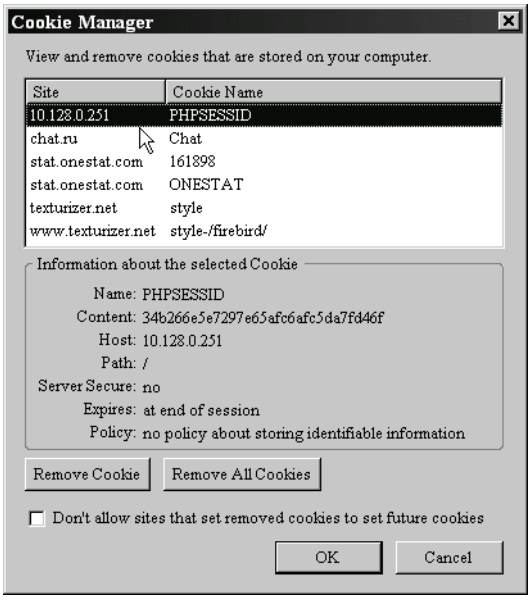


Figure 5–4 Visualisation du cookie via l'interface du navigateur

Cette transmission sera encore plus évidente après avoir désactivé – pour quelques instants – le support des cookies dans votre navigateur. PHP ajoute de manière transparente une variable supplémentaire aux liens vers `p2.php`.

Figure 5–5

B.A.-BA Contrôler les cookies dans son navigateur

Chaque navigateur dispose de sa propre interface pour gérer les cookies. Voici celles disponibles sous Microsoft Internet Explorer et sous Mozilla (ou Netscape).

Avec Microsoft Internet Explorer

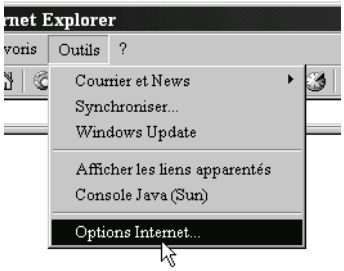
Toutes les options de configuration d'Internet Explorer sont accessibles à partir du menu Outils, sous-menu Options Internet 1.

À partir de l'onglet Général, il est possible d'effacer la totalité des cookies stockés et même d'accéder au répertoire de stockage (via le bouton Paramètres) 2.

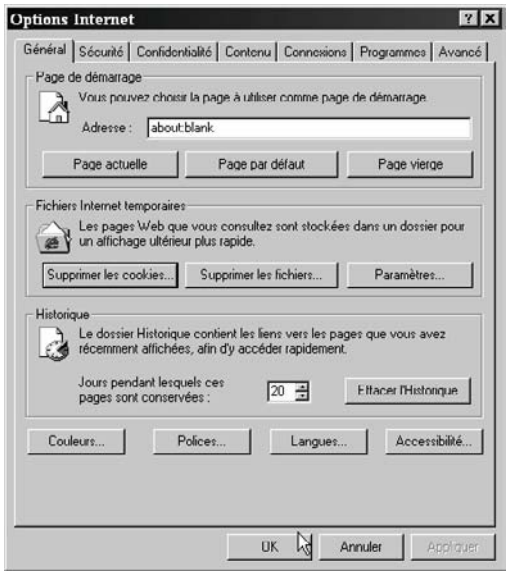
La configuration de la politique de sécurité appliquée au cookie est accessible via l'onglet Confidentialité 3.

Il est alors possible de choisir un niveau pré-établi ou de préciser de manière plus radicale ses choix (bouton Avancé) 4.

1



2



3

4

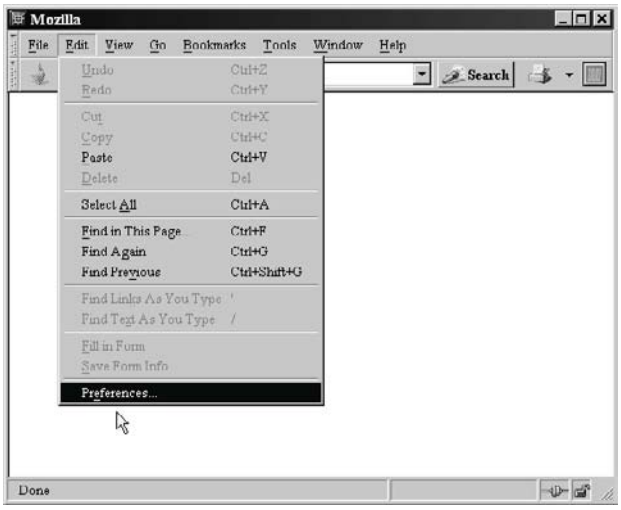
Avec Mozilla et Netscape

Toutes les préférences de Mozilla sont accessibles depuis le menu Édition, sous-menu Préférences 1.

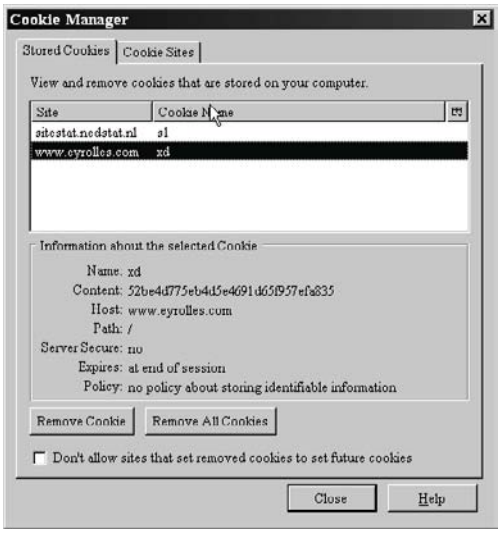
La gestion des cookies est traitée dans la section Sécurité. Il est possible de déterminer simplement le refus ou non des cookies, de l'afficher éventuellement pour certains sites, voire, comme pour Internet Explorer, de définir des politiques par zone et niveau de sécurité 2.

Enfin, Mozilla offre la possibilité d'explorer efficacement le contenu des cookies et de supprimer sélectivement certains d'entre eux 3.

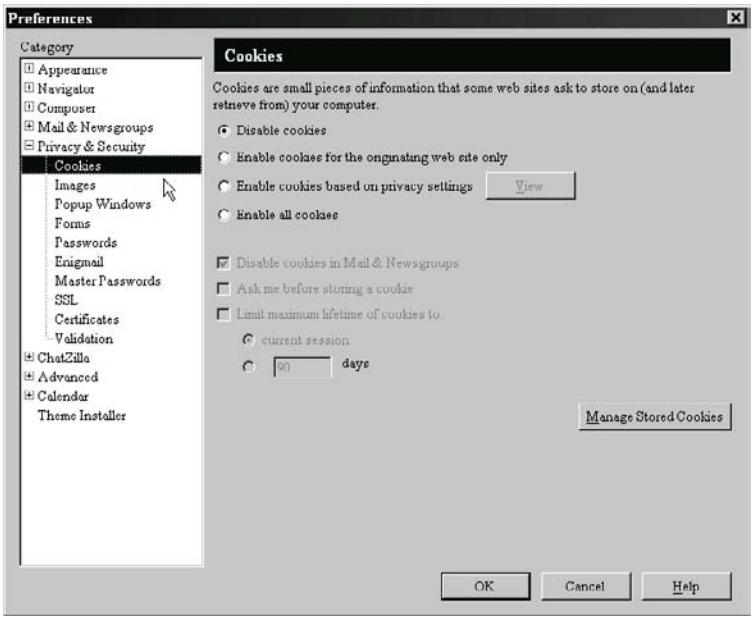
1



3



2



Sauvegarde des données de session

L'instruction `session_start()` permet donc d'activer le suivi des visiteurs. Cependant, pour le moment, hormis l'identifiant, le profil d'un visiteur ne comporte encore aucune donnée. Pour résoudre ce manque, PHP permet d'associer certaines données à la session. Celles-ci seront alors préservées pendant toute la durée de vie de la session.

Pour ce faire, PHP met à notre disposition le tableau superglobal `_SESSION`. Ce tableau, en permanence accessible, est automatiquement initialisé avec les données de session. De même, chaque élément ajouté à ce tableau sera automatiquement ajouté aux données de la session.

À titre d'expérimentation, nous allons reprendre notre site élémentaire, toujours constitué des deux pages `p1.php` et `p2.php`. Dans la première page, nous allons associer une variable à la session avant de constater si celle-ci est bien conservée dans la deuxième page. La plupart des sites disposant de sessions reposent sur ce même mécanisme.

Et dans PHP Saloon ?

Pour PHP Saloon, la solution retenue consiste à :

- activer explicitement les sessions avec `session_start()` ;
- utiliser directement le tableau `_SESSION`.

Ces options permettent de ne pas avoir à modifier la configuration de PHP, configuration qui n'est pas toujours accessible, notamment dans le cas d'hébergements mutualisés.

p1.php

```
<? session_start(); ?>
<html>
<head>
  <title>Page 1</title>
<head>
<body>
  <p> Nous avons obtenu notre identifiant de session :
      <? print session_id(); ?><br/>
      Et enregistré la variable phpsaloon qui prend la
      valeur 3.
      <? $_SESSION['phpsaloon'] = 3; ?>
  </p>
  <ul>
   <li><a href="p2.php">Lien vers la page 2</a></li>
  </ul>
</body>
</html>
```

p2.php

```
<? session_start(); ?>
<html>
<head>
  <title>Page 2</title>
<head>
<body>
```

```
  <p>Cette page est juste une page de test!</p>
  <p>L'identifiant de session qui nous a été
  attribué est conservé : <? print session_id(); ?></br>
  Et mieux, le contenu de la variable phpsaloon aussi (il
  s'agit de : <? print $_SESSION['phpsaloon']; ?>).
  </p>
</body>
</html>
```

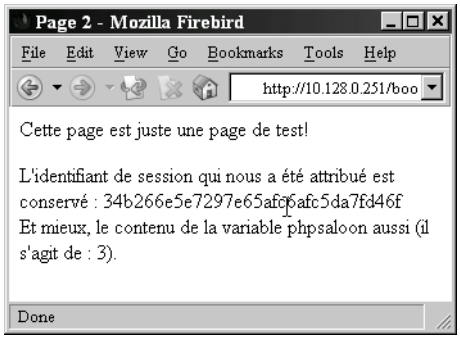


Figure 5–6
Persistance du tableau _SESSION au sein d'une session

ALTERNATIVE `session_register()`

Au lieu de manipuler directement le tableau `_SESSION`, il est possible de travailler sur ses propres variables.

L'instruction `session_register()` permet de désigner les variables qui constituent le profil du visiteur et qui doivent, à ce titre, être préservées d'une page à l'autre.

Avec l'appel à `session_register()`, PHP gère automatiquement la sauvegarde des variables désignées et leur restauration dès l'utilisation de `session_start()`.

L'utilisation de `session_register()` peut donc paraître plus simple, cependant les pièges sont nombreux. Tout d'abord, l'apparente simplification qui résulte de l'utilisation des variables courantes aboutit en réalité à un code difficile à interpréter. En effet, sauf si vous utilisez une charte de nommage très stricte, rien ne distingue une variable classique d'une variable dont la valeur sera préservée durant la session.

Par ailleurs, la restauration automatique des variables est subordonnée à l'activation automatique de l'option `registrer_globals` dans le fichier de configuration PHP. Une option, qui permet aussi la création automatique de variables à partir des paramètres de l'adresse web ou des formulaires, est désormais désactivée par défaut, compte tenu des risques d'utilisations malicieuses des pages développées.

Enfin, l'utilisation simultanée du tableau `_SESSION` et de `session_register()` est impossible (soit PHP gère ce tableau, soit le développeur y accède, mais jamais les deux).

Première implantation de la classe session

Au chapitre 2, nous avons défini l'interface à respecter :

```
<?
interface iSession {
   function update();
   function active();
}
?>
```

Nous allons donc créer la classe `session` en conformité avec cette interface en prévoyant :

- un constructeur ;
- les deux fonctions `update()` et `active()`.

La date de dernière mise à jour sera stockée dans le tableau `_SESSION`. Cette première implantation est particulièrement simple :

```
<?
require_once'isession.php';
class session implements iSession {
   static private $db;
   static private $gestionnaire;
   function __construct($timeout = 120) {
      $this->timeout = $timeout;
      session_start();
   }
   function update() {
      $_SESSION['maj'] = time();
   }
   function active() {
      if ( (time() - $_SESSION['maj']) > $this->timeout)
         return false;
      return true;
   }
}
?>
```

Notre classe `session` peut alors être utilisée dans tous les fichiers pour vérifier l'existence d'une session active :

```
<?
require_once'session.php';
  $session = new session(320);
if ($session->active())
     $session->update();
else {
```

```
// réagir car la session n'existe pas ou a expiré.

die("Il faut réagir!");
  }
?>
<html>
<head><title>Page protégée</title></head>
<body>
Vous voyez cette page car votre session
est active! Hourra!
</body>
</html>
```

REGARD DU DÉVELOPPEUR **Cookies et date d'expiration**

Chaque cookie dispose d'une date d'expiration, il peut donc paraître étrange de gérer nous-mêmes l'expiration quand il serait tellement simple de laisser le cookie expirer tout seul. Ceci d'autant que PHP permet d'agir sur la date d'expiration avec la fonction : `session_set_cookie_params()`.

Hélas, le diable se retrouve, comme souvent, dans les détails. En effet, il appartient au serveur de transmettre la date d'expiration. Cette date est calculée et déterminée en fonction de la date du serveur lui-même. Or, les ordinateurs sont rarement totalement synchrones, et le plus souvent l'ordinateur du visiteur et le serveur ne sont tout simplement pas exactement à l'heure. L'expiration devient alors un jeu aléatoire, d'autant que certains navigateurs vont voir d'un mauvais œil les dates d'expiration dans le passé.

La solution retenue consiste à utiliser un cookie de session. Ce type de cookie ne précise pas de date d'expiration et le navigateur va les traiter spécifiquement. De tels cookies resteront valides tant que le navigateur est ouvert et seront détruits à la fermeture de celui-ci.

Cette situation nous convient parfaitement, même s'il faut en contrepartie gérer nous-mêmes le cas des utilisateurs qui laissent leur navigateur ouvert inactif.

Pilote de sauvegarde pour les sessions

Toutefois, cette première implantation fonctionnelle pose deux problèmes :

- la sécurité ;
- le couplage avec notre modèle de données.

La sécurité tout d'abord. En effet, par défaut, les données de session sont sauvegardées par PHP dans un fichier. Il existe un seul fichier par session et tous les fichiers se trouvent dans le même répertoire. Ce procédé appelle plusieurs réflexions :

1. Les fichiers sont stockés dans un répertoire accessible à tous (notamment dans le cas d'un système Unix) et au mieux dans un répertoire accessible à l'utilisateur pour le serveur web.

CONFIGURATION Chemin de sauvegarde pour les données de session

Par défaut, sous Unix, le chemin de sauvegarde est `/tmp`, le répertoire temporaire traditionnel. Il est possible de modifier ce chemin en utilisant la directive : `session.save_path = "/tmp"`.
ou encore la fonction `session_save_path()`.
Changer le chemin peut améliorer la sécurité des données stockées quand le serveur n'est pas mutualisé.

2. Toutes les sessions sont stockées au même endroit, donc en cas d'hébergement mutualisé et si rien n'est fait, les données des différents clients sont mélangées.
3. Avec ce mélange, un client mal intentionné de l'hébergeur peut accéder aux données des autres clients.

Par chance, PHP permet de remplacer ce mécanisme par celui de son choix. La méthode peut paraître complexe de prime abord mais il n'en est rien. Pour sauvegarder les données de session, PHP fait nécessairement appel à quatre fonctions principales :

- une fonction chargée d'ouvrir le média (par défaut un fichier) dans lequel les données seront stockées ;
- une fonction chargée de lire les données dans le média ;
- une fonction chargée d'écrire les données ;
- une fonction chargée de terminer l'accès au média.

À ceci, s'ajoutent deux fonctions utilitaires :

- une fonction appelée pour détruire une session ;
- une fonction jouant le rôle de *garbage collector* (cette dernière fera l'objet d'une explication spécifique).

Ce sont ces fonctions que PHP nous permet de remplacer avec l'instruction `session_set_save_handler()`. Dans PHP Saloon, nous disposons déjà d'un média bien adapté : notre base de données SQLite !

B.A.-BA Handle, callback, handler & hook

Ces mots sont très souvent utilisés lorsqu'il s'agit de transmettre des fonctions en paramètre ou de manipuler des données dont le contenu nous est caché (données opaques). Leur traduction en français est plutôt hasardeuse et ne constitue pas vraiment un facteur de lisibilité.

Le terme *handler* désigne le plus souvent une fonction, un objet qui joue un rôle de gestionnaire pour un type de données ou une situation. On parlera de *string handler* par exemple et dans notre cas de *save handler*.

Lorsque le handler est destiné à modifier ou à remplacer un élément interne à un logiciel, on parle de *hook* (crochet). Il existe des hooks permettant d'étendre le moteur PHP ou le noyau GNU/Linux, par exemple.

Une *callback* est une fonction, comme peut l'être un handler mais il s'agit en général d'une fonction qui sera déclenchée de manière asynchrone dans une logique événementielle. Les callbacks sont très présentes quand on parle d'interface graphique. On définit ainsi des callbacks pour les boutons, celles-ci seront appelées si l'on clique dessus.

Enfin, le *handle* (poignée) désigne une bribe d'information permettant d'accéder à l'information principale. Ainsi, dans PHP 5, on ne manipule plus les objects directement mais un *object handle* (en fait, le numéro de série de l'objet dans la base des objets existants).

Pilote de session SQLite pour PHP Saloon

Il nous faut donc créer six fonctions et modifier le constructeur de notre classe `session` afin d'activer notre mécanisme de sauvegarde.

Comme nous avons pris le parti d'utiliser les nouvelles fonctionnalités objet de PHP, nous allons réunir ces quatre fonctions au sein d'une même classe qui devra valider l'interface suivante :

```
<?
interface iGestionnaireSession {
   static function open();
   static function close();
   static function read($cle);
   static function write($cle, $valeur);
   static function destroy($cle);
   static function gc($duree);
}
?>
```

L'ensemble des fonctions sont déclarées statiques car en réalité nous n'instancierons pas d'objet et le changement de gestionnaire de session sera réalisé par la fonction `activer()`.

La table sessions

Avant d'activer ce gestionnaire, il convient de nous assurer que la base de données comporte bien une table pour le stockage.

Cette table est simple, nous l'avons déjà décrite dans le chapitre précédent ; elle ne comporte que deux colonnes, une pour l'identifiant de session, une pour les données.

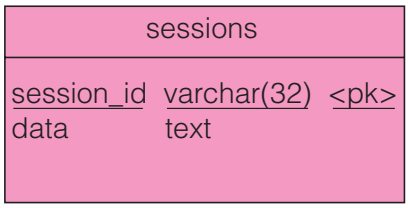


Figure 5–7
Structure de la table `sessions`

Les fonctions du gestionnaire de session interrogeront cette table pour y lire ou y mettre à jour les données de session.

La classe gestionnaireSession

Le code de la classe `session` est relativement simple. Dans cette première version, nous laisserons de côté l'aspect nettoyage en proposant une fonction `garbageCollector()` vide.

```
<?
require_once'igestionnairesession.php';
class gestionnaireSqlite implements iGestionnaireSession {
   static private $db;
   static private $actif;
   static function open() {
      if (gestionnaireSqlite::$db = new sqlite_db('test'))
         return true;
      return false;
   }
   static function close() {
      // laissons PHP tout fermer
      return true;
   }
   static function read($cle) {
      return gestionnaireSqlite::$db->single_query("select data from
sessions where session_id = '$cle'");
   }

   static function write($cle, $valeur) {
      if (! @gestionnaireSqlite::$db->query("insert into sessions
values ('$cle', ".time().", '$valeur')")) {
         if (gestionnaireSqlite::$db->query("update sessions set data =
'$valeur' where session_id = '$cle'"))
            return true;
         return false;
      }
      return true;
   }

   static function destroy($cle) {
      if (gestionnaireSqlite::$db->query("delete from sessions where
sessions_id = '$cle'"))
         return true;
      return false;
   }
   static function gc($duree) {
      return true;
   }
```

MÉTHODE Déboguer les gestionnaires de session

Les fonctions d'un gestionnaire de session peuvent être appelées hors du contexte des scripts PHP (par exemple après son exécution) ; de fait, le débogage est parfois complexe. L'affichage de messages d'informations est le plus souvent inefficace, car la page est déjà créée au moment où l'affichage est demandé.

Pour suivre plus efficacement le fonctionnement d'un gestionnaire, une solution possible est d'écrire les messages de débogage dans un fichier, par exemple avec la fonction `errorlog()`.

```
    static functionactiver() {
        if (! gestionnaireSqlite::$actif) {
            session_set_save_handler(array('gestionnaireSqlite', 'open'),
                    array('gestionnaireSqlite', 'close'),
                    array('gestionnaireSqlite', 'read'),
                    array('gestionnaireSqlite', 'write'),
                    array('gestionnaireSqlite', 'destroy'),
                    array('gestionnaireSqlite', 'gc'));
            gestionnaireSqlite::$actif = true;
        }
    }
}
?>
```

SYNTAXE Accès à des attributs et méthodes statiques

PHP adopte parfois une syntaxe étrange, ainsi l'accès à un attribut statique se fait selon la syntaxe suivante :

```
classe::$attribut
```

et non :

```
$classe::attribut
```

Ce qui est plutôt logique en réalité.

Moins logique, une méthode statique ne peut être désignée par son nom sous la forme :

```
classe::fonction
```

Au contraire, il est nécessaire de construire un tableau, le premier élément désignant la classe, le second le nom de la méthode.

```
array( 'classe', 'methode')
```

À noter que cette syntaxe s'étend aux méthodes objet. Dans ce cas, le premier élément du tableau est l'objet lui-même.

Il faut noter que le gestionnaire de session utilisera sa propre connexion à la base. En effet, les fonctions du gestionnaire peuvent être appelées alors même que l'exécution du script PHP est terminée, en particulier les fonctions d'écriture et de clôture. Il est donc impossible d'utiliser une connexion partagée par tout le script et qui pourrait être close par mégarde avant les derniers appels au gestionnaire.

Garbage collector

Dans l'implantation précédente, nous avons laissé de côté l'aspect nettoyage. Ce rôle est tenu par la fonction `garbageCollector()`, ramasse-miettes en français.

Pourquoi faire le ménage dans nos sessions ? Tout simplement pour éviter l'accumulation de sessions qui ont expiré et sont donc inutiles. Une option possible pour notre fonction `garbageCollector()` peut être de les effacer sans distinction.

Il va de soi que cette fonction ne doit pas être appelée pour chaque script PHP exécuté ! PHP permet donc de définir une probabilité d'appel.

CONFIGURATION **L'appel du garbage collector du gestionnaire de session**

La probabilité d'appel est définie dans le fichier de configuration PHP avec la directive :

```
session.gc_probabilility = 1
```

Cette directive précise le pourcentage des scripts PHP qui verront la fonction de `garbageCollector()` appelée. Par défaut, il s'agit de 1.

Enfin, PHP permet de préciser un délai de péremption (par défaut 1440 secondes). Ce délai est passé en paramètre à la fonction de nettoyage et est configurable avec la directive :

```
session_gc_maxlifetime = 1440
```

Dans le cas de PHP Saloon, ce délai est ignoré au profit du délai d'expiration fixé pour les sessions.

Implantation retenue

Avec notre gestionnaire de session, nous pouvons améliorer notre classe `session` pour aboutir à un code plus satisfaisant.

```
<?
require_once'gestionnairesqlite.php';
require_once'unserialize.php';
require_once'isession.php';
class session implements iSession {
   function __construct($timeout) {
      $this->timeout = $timeout;
      gestionnaireSqlite::activer();
      session_start();
   }
   function update() {
      $_SESSION['maj'] = time();
   }
   function active() {
      if ( (time() - $_SESSION['maj']) > $this->timeout)
         return false;
      return true;
   }
```

```
   static function getSessionData($data, $nom) {
      $a = unserialize_session($data);
      return $a[$nom];
   }
}
?>
```

Cette nouvelle classe `session` sauvegarde les données dans la base de données SQLite utilisée pour tout PHP Saloon. Il va donc nous être possible de tirer directement profit des données de session dans nos futures requêtes SQL.

Décodage des données de session

Hélas, là encore, le diable se retrouve dans les détails. En effet, pour sauvegarder les données de session (en réalité le tableau `_SESSION`), PHP réalise une opération de sérialisation. Cette opération consiste à transformer un objet stocké en mémoire (ici, notre tableau `_SESSION`) en une chaîne de caractères facile à sauvegarder. Une opération inverse (désérialisation) permet de recréer l'objet en mémoire à partir de la chaîne de caractères.

Cet encodage en un seul agrégat de données est souvent pénible à gérer, surtout si l'on dispose en parallèle d'un modèle de données complet. Pour PHP Saloon, les choses restent relativement simples, notre profil de session ne comportera que deux informations dont la date de mise à jour.

Cependant, dans la plupart des cas, et notamment dans les applications de commerce électronique, ce stockage en retrait du modèle de données est tout simplement impensable. Aussi, nombre de services se contenteront de confier à PHP la gestion de l'identifiant et des cookies et réaliseront eux-mêmes le stockage de chaque élément manipulé avec des requêtes dans la base de données.

Par ailleurs, si PHP dispose de deux fonctions : `serialize()` et `unserialize()` qui fonctionnent sur toutes les données manipulées dans PHP, celles-ci ne sont pas celles utilisées pour encoder les données de session. Pire, la fonction de décodage nécessaire n'est pas disponible en standard dans PHP.

Il va donc falloir user d'un peu de sorcellerie et surtout expliquer une fonction PHP spécifique présentée en annexe : `unserialize_session()`. Celle-ci servira à décoder les données stockées dans notre table sessions et va nous permettre d'ajouter une fonction d'accès aux données de session dans notre classe `session` :

```
<?
static function getSessionData($data, $nom) {
   $a = unserialize_session($data);
   return $a[$nom];
}
?>
```

ATTENTION session_decode()

La fonction `session_encode` retourne bien une chaîne de caractères représentant le contenu du tableau `_SESSION`, mais malheureusement la fonction `session_decode()` ne retourne pas assez logiquement un objet PHP mais modifie directement le tableau `_SESSION`. Impossible donc, par exemple, de consulter les données de la table `sessions` de notre base SQLite sans démolir le tableau `_SESSION` courant !

La fonction `getSessionData()` est déclarée statique car elle ne dépend pas d'un objet mais est utilisable sans contrainte particulière. Nous l'utiliserons notamment dans les requêtes SQLite.

```
<?
function unserialize_session($val)
{
// traiter le cas des pipes dans les chaines
   $val = preg_replace('/"([^"]*)";/e','\'"\'. urlencode("$1") .
                \'";\'',$val);
// séparer nom et valeur
   $tmp = preg_split('/([^;}][A-Za-z0-9_]+)\|/', $val, -1,
                PREG_SPLIT_DELIM_CAPTURE);
// reconstruire le tableau
for(array_shift($tmp);
       $nom = array_shift($tmp);
       $result[$nom] = unserialize(urldecode(array_shift($tmp))));
return $result;
}
?>
```

COMPRENDRE **Le codage des données de session**

Nous l'avons vu, PHP encode les données de session de manière privée (il est toutefois possible d'opter pour le format WDDX sous réserve de disposer de l'extension).

Ainsi, pour un tableau de deux éléments :

```
array( 'premier' => 2, 'deuxieme' => "foo")
```

PHP encode comme suit :

```
premier|i:2;deuxieme|s:3:"foo";
```

Ce codage simple d'apparence nécessite un peu d'effort en cas de données plus complexes. Voici donc un peu de magie avec la fonction `unserialize_session()` (voir ci-dessus). Celle-ci réalise ce travail au mieux, sans toutefois être parfaite. Elle est suffisante pour PHP Saloon.

Extension de la classe session

La classe `session` créée précédemment est suffisante. Pour éviter néanmoins de répéter sur chaque page le mécanisme de test de validité et le traitement en cas d'expiration, il peut être judicieux d'étendre la classe `session` en définissant une nouvelle classe : `sessionValide`.

Les objets ainsi définis imposeront la présence d'une session valide, ou provoqueront la redirection vers la page d'identification.

```
<?
require_once'session.php';
class sessionValide extends session {
   function __construct($redirection, $frame = false ) {
      parent::__construct(120);
      if ($this->active()) {
         $this->update();
      } else {
         if (! $frame)
            header('Location: ' . $redirection);
         else
            print <<<EOF <html>
  <head><title>Redirection !</title></head>
  <script>
window.parent.location = "$redirection";
  </script>
  <body>
  Vous devriez être redirigé(e) automatiquement. Si ce n'est pas le
  cas, <a href="$redirection">cliquez-ici</a>.
  </body>
</html>
EOF;      exit();
      }
   }
}
?>
```

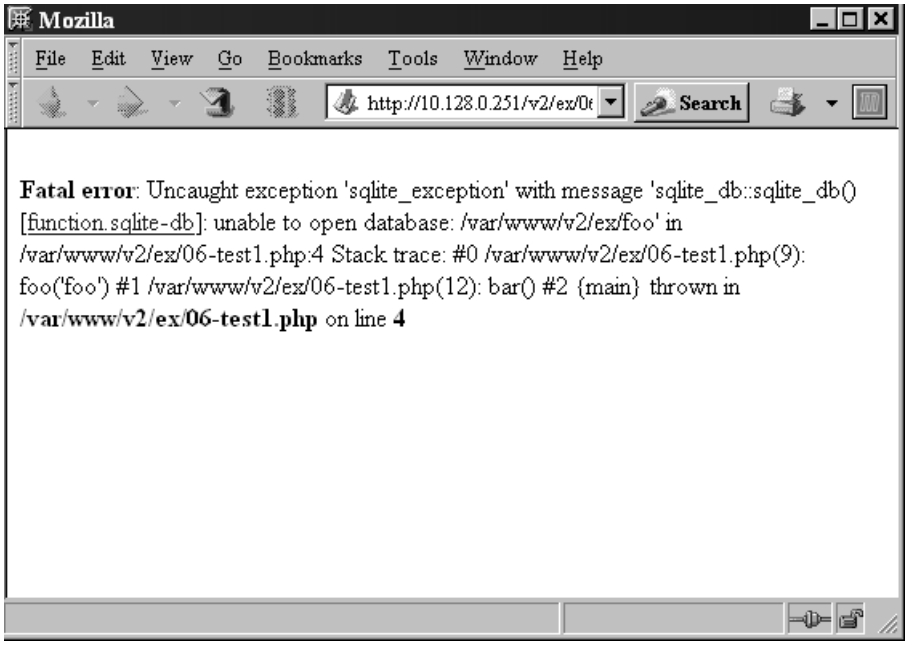
Cette nouvelle classe prend en compte la redirection, même quand l'expiration se produit au sein d'un frame ; c'est alors la totalité du document parent qui est rechargée.

En résumé...

PHP offre une multitude de fonctionnalités pour gérer les sessions. Certaines permettent de disposer de sessions de manière presque totalement transparente. Cependant, comme nous venons de le constater, il est parfois plus sage de mettre les mains dans le cambouis et de savoir avec précision ce qu'il en est des données manipulées pour pouvoir en faire le meilleur usage.

Pour PHP Saloon, nous venons d'adopter une position médiane en confiant à PHP la gestion des aspects les plus ardus (comme les cookies ou la création de l'identifiant de session) tout en conservant le contrôle sur le stockage des données.

chapitre 6


Mozilla
File Edit View Go Bookmarks Tools Window Help
http://10.128.0.251/v2/ex/0
Search
Fatal error: Uncaught exception 'sqlite_exception' with message 'sqlite_db::sqlite_db() [function.sqlite-db]: unable to open database: /var/www/v2/ex/foo' in /var/www/v2/ex/06-test1.php:4 Stack trace: #0 /var/www/v2/ex/06-test1.php(9): foo('foo') #1 /var/www/v2/ex/06-test1.php(12): bar() #2 {main} thrown in /var/www/v2/ex/06-test1.php on line 4

Gérer les erreurs grâce aux exceptions

Jusqu'à cette nouvelle version, PHP est resté très conventionnel dans la gestion des erreurs. Or, nous verrons que le nouveau système d'exceptions de PHP 5 peut apporter une amélioration sensible de la qualité des développements. Pour autant, la gestion historique des erreurs ne doit pas disparaître. Tout est question d'équilibre.

SOMMAIRE

MOTS-CLÉS

- try
- catch
- throw
- error handler

Le traitement classique des erreurs dans PHP Saloon

Dans un programme, chaque appel de fonction ou presque, est susceptible d'échouer, parce que des paramètres incorrects sont mis en jeu ou simplement parce que l'environnement d'exécution n'est pas celui escompté.

Un principe élémentaire

Le traitement classique de la situation s'articule en deux temps :

- tester à chaque étape la réussite de l'action à mener ;
- en cas d'échec, retourner un code d'erreur.

Ce modèle de fonctionnement est particulièrement simple. Cette simplicité conceptuelle est pourtant à l'origine de l'inconvénient majeur qui pénalise cette méthode. Dans la pratique, il vous faudra effectuer des dizaines de tests qui complexifient le code développé, sans parler des codes d'erreurs et de leurs messages associés qui doivent cheminer entre les appels de fonctions pour être exploitables.

Considérons la classe `utilisateur` décrite au chapitre 4. Sa méthode `connecte()` illustre parfaitement cette manière de faire :

```
class utilisateur {
// ...
   function connecte($db) {
      // $db est une connexion déjà ouverte vers la base SQLite
      $id = $db->single_query("select id from utilisateurs
            ➥ where pseudo = '{$this->pseudo}' and
            ➥ motdepasse= '{$this->motdepasse}'");
      if ($id) {
         $session = new session(120);
         $_SESSION['uid'] = $id;
         $session->update();
         return true;
      }
      return false;
   }
   // ...
}
```

Un test est réalisé pour s'assurer que la requête s'est correctement déroulée. En fonction du résultat, un code de retour est renvoyé comme résultat de l'appel à `connecte()`. Tout devrait donc être pour le mieux. Mais il n'en est rien.

Une réalité plus complexe

Première erreur, la signification de la valeur `$id` est biaisée. Rien ne permet, à première vue, de distinguer l'échec de la requête, par exemple si le code SQL est incor-

PHP 4 Objet ou pas, le traitement des erreurs reste une priorité

Dans PHP 4, aucun choix possible ; seul le traitement classique des erreurs est disponible et il demeure impératif pour assurer à l'utilisateur la meilleure qualité de service possible, mais aussi pour protéger les données manipulées.

rect, d'un identifiant qui vaudrait zéro. Certes, on suppose dans ce cas que l'identifiant d'un utilisateur n'est jamais nul, mais dans la pratique, qui pourrait le jurer ?

Il faut donc examiner de plus près le fonctionnement de la méthode `single_query()` dans cinq situations précises :

- La requête est erronée.
- La requête est valide mais aucune ligne ne correspond.
- La requête est valide, une ligne correspond et la valeur est 0.
- La requête est valide, une ligne correspond et la valeur est non nulle.
- La requête est valide, plusieurs lignes correspondent.

Nous le voyons d'emblée, d'une situation apparemment élémentaire, on bascule dans la gestion de cas multiples et parfois complexes dans leurs implications. La documentation de la classe `sqlite_db` nous permet de construire le tableau suivant :

	Requête erronée	Pas de résultat	Résultat unique nul	Résultat unique	Résultat multiple
Valeur de retour	Bool(false)	NULL	String(1) "0"	String	Array()
Evalué à false avec ==	✔	✔	✔		
À exclure	✔	✔			✔

Si l'on n'y prend pas garde, ce sont donc seulement trois cas sur cinq qui sont évalués comme équivalents au booléen `false`, et pas forcément ceux à exclure.

La simple comparaison réalisée dans notre code, comme d'ailleurs, l'utilisation de l'opérateur de comparaison usuel == ne suffit pas. PHP dispose par chance de l'opérateur ===. Contrairement à l'opérateur classique == la comparaison va porter aussi sur le type de la donnée, telle que stockée dans l'interpréteur PHP.

Nous pouvons donc modifier notre méthode de la façon suivante :

```
class utilisateur {
   // ...
   function connecte($db) {
      // $db est une connexion déjà ouverte vers la base SQLite
      $id = $db->single_query("select id from utilisateurs
           ➥ where pseudo = '{$this->pseudo}' and
           ➥ motdepasse= '{$this->motdepasse}'");
      if ( ($id === false) || ($id === NULL) || is_array($id) )
         return false;
      $session = new session(120);
      $_SESSION['uid'] = $id;
      $session->update();
      return true;
   }
   // ...
}
```

COMPRENDRE La comparaison dans les langages peu typés

Le comportement de l'opérateur == peut paraître bien étrange dans la situation présentée. Mais il s'agit en réalité du fonctionnement classique adopté dans l'ensemble des langages non typés et même, dans une moindre mesure, en C.
Cette approche permissive qui veut que globalement tous les éléments neutres des types de données soient évalués à `false`, est le plus souvent simplificatrice.
Le revers est alors de ne pas détecter les situations pour lesquelles cette version étendue de la comparaison biaise le comportement attendu.

Cette version n'assure encore qu'une prise en compte partielle des erreurs.

Il faudrait probablement retourner une valeur distincte de `false` pour distinguer les erreurs manifestes, soit dans le code SQL, soit dans la cohérence de la base, du simple fait que l'utilisateur n'existe pas ou a donné un mot de passe erroné.

On pourrait choisir les codes de retour suivants (assez classiques au demeurant) :

- -1 : erreur dans l'application ;
- 0 : erreur dans les données utilisateur (login ou mot de passe) ;
- 1 : utilisateur connecté.

Soit une petite modification dans le code :

```
class utilisateur {
   // ...
   function connecte($db) {
      // $db est une connexion déjà ouverte vers la base SQLite
      $id = $db->single_query("select id from utilisateurs
           where pseudo = '{$this->pseudo}' and
           motdepasse= '{$this->motdepasse}'");
      if ( ($id === false) || is_array($id) )
         return -1;
      if ($id === NULL)
         return false;
      $session = new session(120);
      $_SESSION['uid'] = $id;
      $session->update();
      return true;
   }
   // ...
}
```

Dans la pratique, il reste encore un bon nombre d'erreurs qui ne sont pas prises en compte. Ainsi, la création d'une nouvelle session pourrait échouer, comme sa mise à jour avec la méthode `update()`.

Notre idée de départ, extrêmement simple, peut très vite virer au cauchemar, avec le risque d'oublier ici et là des situations d'échec.

Un risque additionnel pour les applications web

Ce risque est d'autant plus grand que les applications web doivent tenir compte, et ce, avec la plus grande rigueur, d'une vulnérabilité classique mais bien plus dangereuse à l'échelle d'Internet : le *cross site scripting* ou XSS.

PHP Saloon, comme la plupart des applications web dynamiques, interagit avec l'utilisateur, qui lui transmet des informations. Dès lors, l'application manipule des contenus, stockés le plus souvent dans des variables (dans

l'exemple précédent le mot de passe et le pseudo) dont la nature est par essence, incontrôlable a priori.

Le XSS consiste donc à véroler le contenu de ces variables, directement via un formulaire, ou en modifiant les URL. L'idée est alors de placer du code malicieux en lieu et place des données attendues.

Dans une application comme PHP Saloon, ce code malicieux peut revêtir au moins deux formes : du code SQL, susceptible de corrompre, voire de détruire notre base de données, ou encore du code PHP qui accédera alors à la totalité des fonctionnalités offertes par PHP côté serveur.

Reconsidérons notre requête, par exemple avec `PHPFan` comme pseudo et `secret` comme mot de passe. Dans ce cas, tout va pour le mieux, la requête exécutée est :

```
select id from utilisateurs where pseudo = 'PHPFan' and
motdepasse= 'secret'
```

Que se passe-t-il si la valeur du mot de passe est désormais `secret'; delete from utilisateurs` ? La requête exécutée s'écrit à présent :

```
select id from utilisateurs where pseudo = 'PHPFan' and
motdepasse= 'secret'; delete from utilisateurs
```

Faute de précautions suffisantes, nous venons de perdre l'intégralité de nos utilisateurs.

Il convient donc de modifier notre code pour ajouter les vérifications indispensables, vérifications qui peuvent aboutir à de nouvelles erreurs ou un rejet des données, d'où un nouveau cycle dans le traitement des erreurs.

ATTENTION Magic quotes

PHP essaie le plus possible de fournir des automatismes pour éviter les incidents liés à l'insertion de code malicieux dans les variables. C'est notamment le cas des « Magic quotes ». Cette fonctionnalité, configurable dans le fichier php.ini, permet à PHP de résoudre le cas épineux des guillemets et de l'apostrophe, notamment pour les bases de données et les formulaires web.
Cependant, dans une très large majorité de cas, ce traitement sympathique sera insuffisant et peut aller jusqu'à compliquer davantage le traitement de l'information.
On peut alors désactiver ces transformations et faire appel à des fonctions plus puissantes :

- `htmlentities();`
- `addslashes();`
- `escapeshellcmd()`.

Les exceptions, comme alternative

La chasse aux erreurs risque de transformer rapidement ce qui devait être une classe toute simple en un morceau de code spaghetti quasi indéchiffrable. Pour éviter ce genre de dérive, les exceptions vont mettre en œuvre une tout autre philosophie.

Le concept

L'idée principale est de détacher le code correspondant au fonctionnement normal, du code « exceptionnel » qui peut entrer en jeu de manière ponctuelle pour considérer une situation particulière, souvent une erreur.

Syntaxe des exceptions en PHP

```
try {
   // algorithme général
}
catch(PHPSaloon_Exception $e) {
   // situation exceptionnelle 1
}
catch(Exception $e) {
   // situation exceptionnelle 2
}
```

L'intérêt de ce procédé est de préserver le maximum de lisibilité en séparant l'algorithme et le mécanisme mis en jeu, mais aussi les traitements conceptuellement moins essentiels, indépendamment de leur nécessité.

La structure syntaxique adoptée dans PHP (et par la plupart des autres langages) reprend cette séparation avec d'une part un bloc `try`, dévolu au cadre général, et un ou plusieurs blocs `catch` qui traiteront les cas exceptionnels.

Le fonctionnement dans PHP

Au-delà de ce découpage, il reste à déterminer les conditions du basculement entre le cadre général (`try`) et les situations exceptionnelles (`catch`). Pour comprendre le fonctionnement adopté, on peut associer l'exception à la notion de signal.

Au sein du bloc `try`, l'exécution d'une instruction erronée va déclencher l'émission d'un signal : notre exception. Ce signal va interrompre l'exécution de tout le bloc.

Ensuite, en fonction de la nature du signal, l'interpréteur PHP exécutera le bloc `catch` approprié. Dans cette hypothèse, l'exécution du bloc `try` n'est jamais terminée.

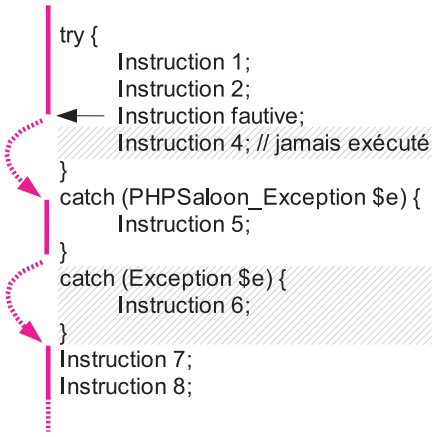


Figure 6–1 Interception d'une exception par un bloc `try/catch`

Comme PHP 5 dispose d'un bon modèle objet, l'implantation choisie pour les exceptions ne se résume pas à un simple signal, mais définit les exceptions comme des objets de la classe `exception` ou de tout autre classe dérivée.

Le choix de la section `catch` appropriée est alors fonction de la classe de l'objet. À charge aux développeurs de créer leurs propres classes d'exceptions.

Du point de vue pratique, l'interpréteur et les extensions PHP peuvent provoquer des exceptions, mais cela vaut aussi pour le code PHP lui-même via l'instruction `throw` :

```
throw new PHPSaloon_exception();
```

Pour PHP Saloon, nous pouvons donc définir notre propre classe `exception`. Il n'est pas nécessaire d'ajouter quoi que ce soit. Ce qui nous intéresse ici est de pouvoir différencier nos exceptions de celles susceptibles d'être déclenchées par ailleurs :

```
   class PHPSaloon_Exception extends Exception {
}
```

EN COULISSE **Les propriétés de la classe** `exception`

La classe `exception` est imposée par PHP comme classe parente de toute classe utilisable avec `catch`. Cette classe est naturellement vouée à être étendue sous forme de classes dérivées. Celle-ci dispose en l'état d'un certain nombre de méthodes (les propriétés sont privées, ce qui est naturel) :

- `getMessage()`;
- `getCode()`;
- `getLine()`;
- `getFile()`;
- `getTrace()`;
- `getTraceAsString()`.

L'ensemble de ces méthodes permet d'accéder à la totalité des informations clés : message d'erreur, pile des fonctions appelées, localisation dans le code.

Cette richesse permet de réaliser, notamment pour le débogage, des affichages plus complets et plus utiles à la manière de Java, ce que permet l'accès à la pile des appels (fonction `getTrace()`).

Cette définition nous permettra alors de traiter séparément et au mieux les exceptions en fonction de leur origine, par exemple SQLite ou DOM (voir le chapitre suivant) :

```
try {
   // ...
   $db = new sqlite_db('test');
   // ...
   if (! $x) {
      // ...
   throw new PHPSaloon_Exception("Utilisateur inconnu");
   }
}
catch (SQLite_Exception $e) {
   // pas récupérable, il faut tout arrêter
   throw $e ;
}
catch (PHPSaloon_Exception $e) {
   // afficher un message et c'est tout.
}
```

Dans cet exemple, chaque bloc `catch` traitera les exceptions pour lesquelles il est prévu (en fonction de la classe attendue). L'irruption d'une extension générique de classe `Exception` ne sera pas, cependant, prise en compte par ces blocs. Cette exception se propagera alors des blocs `catch` vers un bloc `try` englobant, s'il existe, et à défaut, le programme sera interrompu.

Par précaution, il est donc possible de placer un bloc `catch` général paramétré par la classe `exception`, à la suite de traitements plus précis. Nous aurons alors la garantie qu'aucune exception, incluant celles non prévues lors des développements, ne passera à travers les mailles du filet.

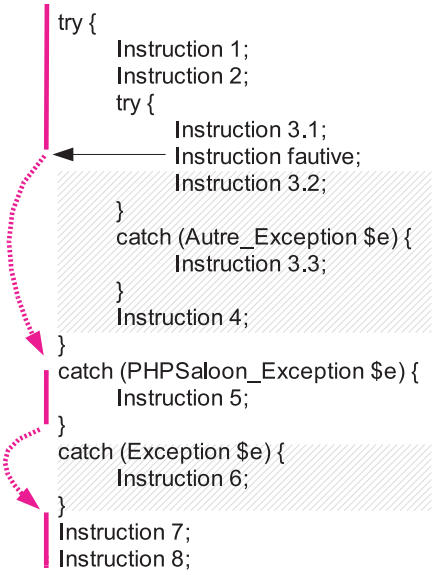


Figure 6–2 Propagation d'une exception dans une hiérarchie de blocs `try`

PHP VS JAVA `finally()`

Contrairement à Java, PHP ne complète pas `try` et `catch` par un bloc `finally`. Ce bloc est normalement exécuté, quoi qu'il advienne, à compter du moment où une exception a été déclenchée et ceci, juste avant de poursuivre l'exécution du programme (en tout dernier, donc).
Dans la pratique, il est toujours possible de se passer d'un tel bloc, son absence ne devrait donc pas être pénalisante pour PHP 5.

On pourrait alors penser qu'il suffit de placer un seul bloc `catch` de ce type pour éviter toute difficulté. Dans ce cas, c'est l'identification de la nature exacte de l'erreur qui pose problème. Il ne reste, dans cette optique, plus que la comparaison des messages d'erreurs pour déterminer les actions à mener.

Quels coûts pour les exceptions ?

Si la prise en compte rigoureuse des erreurs au moyen de codes de retour est fastidieuse, son coût est globalement assez faible. Pour les exceptions, la mécanique mise en œuvre au niveau de l'interpréteur est d'une tout autre complexité.

Pour chaque bloc `try`, il convient en effet de stocker l'état des informations locales pour pouvoir les reconstituer le cas échéant, par exemple dans le cadre d'un bloc `catch`. Ceci prend du temps, et de la mémoire. Ces photographies du système à un instant donné devront par ailleurs être conservées pour chaque hiérarchie de bloc `try`.

Ainsi, plus les imbrications sont nombreuses, plus le coût est important. Étant donné la jeunesse de PHP 5, il n'existe encore aucune mesure concernant le coût exact du mécanisme. Cependant quelques mesures élémentaires tendent à montrer qu'il peut rapidement devenir pénalisant, d'autant que l'utilisation du modèle objet est propice à l'utilisation des exceptions et peut résulter d'un niveau d'imbrication non négligeable.

Exceptions ou erreurs : une question d'équilibre

Alors, exceptions ou erreurs ? La querelle ne semble pas avoir de solution et déchaîne les passions entre ceux qui assimilent les exceptions au pire (le `goto`) et ceux qui voudraient en voir partout. La plupart des langages n'ont pas tranché, à l'exception du langage Ada qui en fait un modèle de développement.

Faut-il alors utiliser ces exceptions ? Oui, mais avec parcimonie. Il est toujours tentant, comme pour l'héritage, de voir des exceptions à tous les niveaux. En réalité, dans la plupart des cas, il est possible de traiter sans trop de complexité une partie des erreurs, tout en réservant aux exceptions les dysfonctionnements ou les événements qui relèvent d'une signification particulière au niveau de l'application.

Dans PHP, ce juste milieu est d'autant plus important que l'introduction des exceptions est récente. Toutes les exceptions n'intègrent donc pas forcément ce mécanisme. Il faudra apprendre à vivre avec les erreurs.

De plus, même pour une extension « compatible », comme SQLite, les choses ne sont pas nécessairement aussi simples. C'est le cas de l'instruction suivante :

```
$db = new sqlite_db('foo');
```

Elle est susceptible de déclencher une exception si pour une raison ou une autre (comme un problème de droit), il est impossible d'ouvrir la base « foo ».

Prenons le cas de cette autre instruction :

```
$result = $db->query('select * from');
```

Elle semble manifestement incorrecte (la table manque) et ne déclenchera pas d'exception. En effet, les développeurs ont considéré que la syntaxe incorrecte d'une requête SQL relevait du *warning* et non de l'erreur. `$result` prend donc la valeur `false` sans déclencher d'exception. Dans la pratique l'utilisation des extensions est donc réservée à une situation d'échec fatal dans l'extension SQLite.

Parmi les solutions envisageables, on peut toujours déterminer les situations exceptionnelles importantes, et ce, dès la phase de conception, de manière à déterminer (avec plus ou moins de précision) le périmètre des incidents qui rentrent dans le cadre des exceptions.

Dans PHP Saloon, on peut faire usage des exceptions au niveau des constructeurs. En effet, des erreurs majeures sont à même de survenir au moment de l'initialisation des objets. Le constructeur ne disposant pas de valeur de retour, l'utilisation d'exception est alors tout à fait adaptée.

MÉTHODE **N'utiliser que des exceptions**

Dans un langage comme Ada, toute erreur se matérialise sous forme d'exception. Il en résulte un modèle de programmation original. Il est tentant de chercher à reproduire cette approche en PHP.

Pour le moment il n'existe malheureusement aucune méthode simple pour transformer d'emblée les erreurs et *warnings* PHP en exception.

Il est possible toutefois de parvenir à un résultat approchant en définissant son propre gestionnaire d'erreurs avec la fonction `set_error_handler()`. Le gestionnaire défini est alors exécuté lors de chaque erreur, ce qui donne la possibilité d'intercepter les erreurs et de déclencher soi-même une exception.

Dans notre application, une telle utilisation est d'autant plus simple que les principales extensions mises en jeu (SQLite et DOM) définissent chacune leurs exceptions de référence. Elles sont susceptibles de les déclencher au moment de l'initialisation des objets (connexion ou document) :

```
class connecte {
   function __construct() {
      try {
         // ...
      }
      catch(SQLite_Exception $e) {
         // ...
      }
      catch(DOM_Exception $e) {
         //
      }
      catch(Exception $e) {
         //
      }
   }
   // ...
}
```

Ouverture de documents XML, connexion à la base SQLite

Accès à la base compromis

Problème XML

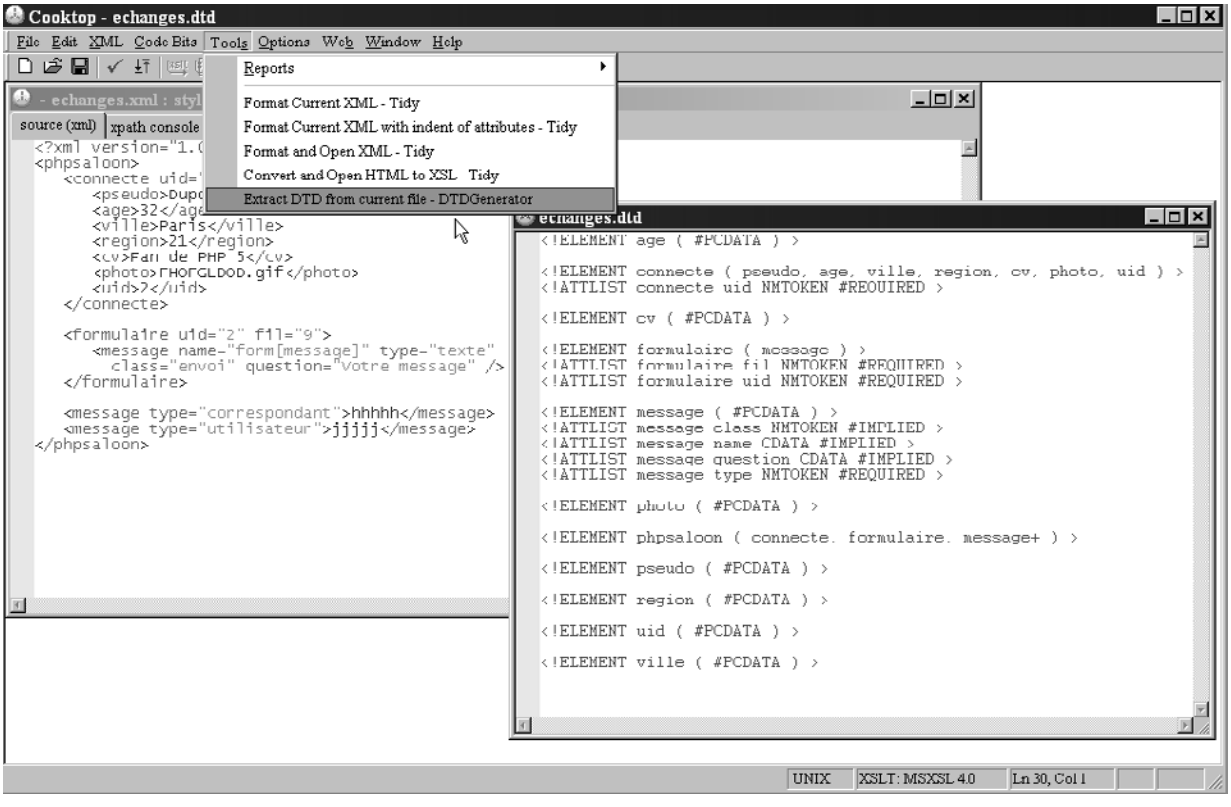
Traitement par défaut

En résumé…

Les exceptions constituent un mécanisme supplémentaire pour aboutir à du code de bonne qualité. En la matière, PHP 5 propose une version pleinement satisfaisante. Cependant, il faudra du temps pour que les multiples extensions de PHP tirent le meilleur profit de cette nouveauté.

Par ailleurs, et même si cela peut paraître séduisant, les exceptions restent une possibilité dont il faudra apprendre à user sans en abuser. Contrairement à des langages tout objet, PHP reste un langage fondamentalement procédural et il serait illusoire de croire que l'intégralité des fonctionnalités du langage se calquera sur le modèle des exceptions.

chapitre 7


Cooktop - echanges.dtd
File Edit XML Code Bits Tools Options Web Window Help
Reports
Format Current XML - Tidy
Format Current XML with indent of attributes - Tidy
Format and Open XML - Tidy
Convert and Open HTML to XSL Tidy
Extract DTD from current file - DTDGenerator
source (xml)
xpath console
<?xml version="1.
<phpsaloon>
<connecte uid="
<pseudo>Dupo
<age>32</age
<ville>Paris</ville>
<region>21</region>
<cv>Fan de PHP 5</cv>
<photo>THORGLDOD.gif</photo>
<uid>7</uid>
</connecte>
<formulaire uid="2" fil="9">
<message name="form[message]" type="texte"
class="envoi" question="Votre message" />
</formulaire>
<message type="correspondant">hhhhh</message>
<message type="utilisateur">jjjjj</message>
</phpsaloon>
echanges.dtd
<!ELEMENT age (#PCDATA) >
<!ELEMENT connecte (pseudo, age, ville, region, cv, photo, uid) >
<!ATTLIST connecte uid NMTOKEN #REQUIRED >
<!ELEMENT cv (#PCDATA) >
<!ELEMENT formulaire (message) >
<!ATTLIST formulaire fil NMTOKEN #REQUIRED >
<!ATTLIST formulaire uid NMTOKEN #REQUIRED >
<!ELEMENT message (#PCDATA) >
<!ATTLIST message class NMTOKEN #IMPLIED >
<!ATTLIST message name CDATA #IMPLIED >
<!ATTLIST message question CDATA #IMPLIED >
<!ATTLIST message type NMTOKEN #REQUIRED >
<!ELEMENT photo (#PCDATA) >
<!ELEMENT phpsaloon (connecte, formulaire, message+) >
<!ELEMENT pseudo (#PCDATA) >
<!ELEMENT region (#PCDATA) >
<!ELEMENT uid (#PCDATA) >
<!ELEMENT ville (#PCDATA) >
UNIX
XSLT: MSXSL 4.0
Ln 30, Col 1

Échanges et contenus XML avec DOM

Certaines applications web sont encore développées en HTML. Ce chapitre est l'occasion de tourner la page pour aborder XML, qui est le langage de référence pour les services web et les applications de nouvelle génération. D'autant que PHP 5 met en œuvre des fonctionnalités novatrices en la matière.

SOMMAIRE

MOTS-CLÉS

- XML
- DOM
- libXML
- namespace
- DTD

Pourquoi adopter XML ?

Personne n'a pu y échapper ; XML est au cœur des préoccupations en matière de développement. Avant d'entrer dans le vif du sujet, il est important de comprendre ce que recouvre ce terme qui cache une réalité plus simple qu'il n'y paraît.

Tour d'horizon

Première étape. Quid du vocable XML ? La forme développée de cette abréviation est : eXtensible Markup Language. XML serait donc un langage.

En réalité, XML est plutôt une syntaxe, et c'est là toute sa force. En effet, XML a su s'imposer comme une syntaxe universelle pour échanger et décrire l'information, sans se préoccuper de la réalité des informations échangées.

Cette syntaxe est par ailleurs excessivement simple (ce qui est probablement à l'origine de son succès). Elle reprend le principe des balises HTML en le généralisant et en le rendant cohérent.

Un document XML est donc un simple document texte dans lequel l'information est structurée par des balises :

Exemple de document XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<recette nom="Salade de fruits">
   <ingrédients>
      <ingrédient>Pommes</ingrédient>
      <ingrédient>Ananas</ingrédient>
    </ingrédients>
   <explications>
Éplucher, couper en dés, mélanger à un sirop.
   </explications>
</recette>
```

Dans ce document, nous avons proposé nos propres balises. XML ne nous limite pas en la matière, ce n'est tout simplement pas son objet. Il nous dit juste comment structurer l'information avec des balises, par exemple en imposant que chaque balise soit correctement refermée (ce qui n'est pas le cas en HTML, où il n'est pas rare de voir des balises ou <br> sans leurs contreparties et </br>).

La quasi-totalité des navigateurs sait afficher correctement les documents XML en distinguant balises et contenus. C'est le cas de Mozilla et d'Internet Explorer.

À RETENIR XML et l'encodage des caractères

Les documents XML sont des documents texte. Par défaut, l'encodage utilisé est UTF-8. Il est naturellement possible de préciser un encodage différent. Traditionnellement, en France, on choisira Iso-8859-15 ou à défaut Iso-8859-1 si le symbole euro n'est pas requis ou l'encodage n'est pas disponible.

À RETENIR Syntaxe raccourcie

Lorsqu'une balise n'englobe aucun contenu, il est possible d'opter pour une syntaxe plus courte. Ainsi :

```
<br></br>
```

Pourra être noté comme suit :

```
<br/>
```

Cette notation n'empêche en rien la présence d'attributs :

```
<br attribut="foo" />
```

Figure 7–1 Affichage d'un document XML par un navigateur

Les langages XML et la DTD de PHP Saloon

À ce stade, vous pouvez légitimement vous interroger face aux déclarations dithyrambiques « d'experts » qui « font du XML ». Ceci est globalement aussi pertinent que de prétendre écrire avec des mots, des espaces, des virgules et des points.

Quitte à nous répéter, XML est juste une syntaxe. Chacun est donc libre de définir ses balises, et leur ordonnancement, tout comme chaque langue possède ses mots, et définit leur organisation (en français, par exemple : sujet, verbe et complément). Ceci est d'autant plus facile en XML que l'interaction entre tous ces vocabulaires est simplifiée par l'utilisation d'espaces de nommage.

De fait, de nombreux organismes métier définissent leurs langages propres en utilisant XML et le W3C, à l'origine d'XML, qui a naturellement proposé des outils et des langages pour décrire précisément ceux construits sur la base d'XML.

C'est le cas de la DTD (Document Type Definition) qui décrit tout simplement la grammaire d'un langage XML en listant pour chaque balise les attributs autorisés, et pour les balises (on parle d'éléments), les sous-balises autorisées.

Cette description supplémentaire est indispensable. En son absence, il devient presque impossible de comprendre un document XML, faute de connaître le rôle et l'organisation des balises entre elles. Pour continuer notre analogie, il vous faut un dictionnaire et un précis de grammaire en français si vous voulez interpréter correctement les phrases, faute de quoi, le flot des mots et de la ponctuation vous sera incompréhensible.

COMPRENDRE **Espace de nommage (namespace)**

Dans le langage naturel, il est rare de parler plusieurs langues à la fois. D'ailleurs, rien n'est prévu pour mélanger dans un même document ou un même discours, des éléments de langues distinctes.

En XML, c'est tout le contraire. Chaque élément peut disposer d'un préfixe qui va préciser son langage de référence. On peut ainsi créer des documents composites.

Les namespaces doivent être déclarés avant leur première utilisation et un document dispose d'un espace de nommage par défaut (ce qui évitera de répéter le préfixe). Par ailleurs, sauf indication contraire, les nœuds enfants héritent de l'espace de nommage de leur père.

La syntaxe retenue consiste à séparer le nom de l'élément de son préfixe par « : » :

```
<phpsaloon:connecte>
   <phpsaloon:description>
Je suis un fan de PHP, ma page :
      <html:a href="...">sur phpfan.com</html:a>
   </phpsaloon:description>
</phpsaloon:connecte>
```

RAPPEL Comment fonctionne une DTD

La DTD décrit la structure d'un document XML (une sorte de grammaire donc). Cette description est assez élémentaire, elle précise pour chaque élément (les balises) :

- la nature des attributs autorisés pour un élément ;
- les sous-éléments autorisés pour ce même élément.

La syntaxe, quoique légèrement exotique, n'est pas bien difficile. Considérons par exemple le document XML suivant :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<a>
   <b>Hello</b>
   <c type="toto">
      <d>World!</d>
   </c>
</a>
```

Il est possible de définir l'élément a dans une DTD comme suit :

```
<!ELEMENT a (b,c)>
```

Ceci signifie que l'élément a est composé d'un élément b, puis d'un élément c. Le rôle de la virgule est d'articuler une séquence. La définition suivante décrit quant à elle un élément a qui ne pourrait contenir que b ou c, mais pas les deux en même temps :

```
<!ELEMENT a (b|c)>
```

On le voit, tout cela est proche des expressions régulières, d'autant que la syntaxe des DTD reprend, elle aussi, les symboles classiques :

- ? pour indiquer que l'expression précédente est optionnelle ;
- + pour indiquer que l'expression est présente au moins une fois ;
- * pour indiquer que l'expression peut être soit absente soit répétée autant de fois que souhaité.

Ainsi la définition suivante définit a comme susceptible de contenir autant de b ou de c nécessaires, et ce, dans n'importe quel ordre :

```
<!ELEMENT a (b|c)*>
```

Lorsqu'un élément peut contenir du texte, on utilise le symbole PCDATA, ainsi pour notre élément d, on peut écrire :

```
<!ELEMENT d (#PCDATA)>
```

Il faut noter que les parenthèses de premier niveau sont indispensables, même lorsqu'un seul symbole est utilisé.

Enfin, deux cas particuliers sont pris en compte :

- Le cas des éléments qui ne comportent pas de contenu, on utilise alors EMPTY comme description.
- Les éléments dont le contenu est laissé libre avec le mot-clé ANY.

La description des attributs est encore plus simple, ainsi pour notre élément c, on peut écrire :

```
<!ATTLIST c type CDATA #REQUIRED>
```

La syntaxe ne réserve pas de surprise, on précise l'élément concerné, l'attribut, le type donné à celui-ci et enfin des options. Ici, le mot-clé CDATA signifie que la valeur est du texte, et #REQUIRED que cet attribut est obligatoire. #IMPLIED signifierait optionnel.

En plus de CDATA, il est possible de définir une énumération, par exemple :

```
<!ATTLIST c type (toto|titi) #IMPLIED>
```

Dans ce cas, l'attribut type est optionnel et peut prendre soit la valeur toto, soit la valeur titi.

On le voit, les choses peuvent devenir rapidement malaisées si le langage XML à décrire est vaste. Dans le cas de notre exemple, une DTD validant notre document serait :

```
<?xml version=1.0 encoding=iso-8859-1 ?>
<!ELEMENT a (b,c) >
<!ELEMENT b (#PCDATA) >
<!ELEMENT c (d) >
<!ELEMENT d (#PCDATA) >
<!ATTLIST c type CDATA #REQUIRED >
```

Il faut toutefois avoir en mémoire qu'il s'agit d'une DTD parmi d'autres ; un document donné ne suffit pas à décrire un langage. Il convient donc de bien définir la nature des documents qui seront manipulés avant d'écrire la DTD.

Une autre DTD acceptable pour notre document est par exemple :

```
<!ELEMENT a (b,c*) >
<!ELEMENT b (#PCDATA|d) >
<!ELEMENT c (#PCDATA|d*) >
<!ELEMENT d (#PCDATA|EMPTY) >
<!ATTLIST c type CDATA #IMPLIED >
```

Dans ce cas, d'autres profils de documents vont être conformes à cette nouvelle DTD, documents qui ne sont peut-être pas autorisés, d'un point de vue applicatif.

La syntaxe précise d'une DTD dans la recommandation du W3C :

- http://www.w3.org/TR/2004/REC-xml-20040204/

Dans le cas de PHP Saloon, nous allons ainsi manipuler des listes de connectés (« connecte » sera donc un des éléments présent dans nos documents XML). Voici un exemple :

Document XML représentant une liste de connectés

```
<?xml version="1.0"?>
<phpsaloon>
  <connectes>
    <connecte uid="1">
      <pseudo>foo</pseudo>
      <age>32</age>
      <ville>Paris</ville>
      <cv>fan de php5</cv>
    </connecte>
    <connecte uid="45">
      <pseudo>bar</pseudo>
      <age>25</age>
      <ville>Rennes</ville>
      <cv>fan de Perl</cv>
      <photo>GO405K40GK40.png</photo>
    </connecte>
  </connectes>
</phpsaloon>
```

Dans une DTD, on pourrait alors indiquer :

DTD validant la liste des connectés

```
<!ELEMENT phpsaloon (connectes)>
<!ELEMENT connectes (connecte)*>
<!ELEMENT connecte (pseudo, age?, ville?, cv?, photo?)>
<!ATTLIST connecte uid CDATA #REQUIRED>
<!ELEMENT pseudo (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
```

Cette obstination à vouloir décrire très exhaustivement les langages créés peut paraître excessive. Après tout, nul ne connaît le dictionnaire par cœur et certaines constructions syntaxiques laissent parfois rêveur.

Cependant, l'homme n'est pas un ordinateur et il ne faut pas oublier que, derrière les échanges XML, nous trouverons de plus en plus d'éléments critiques : financiers ou contractuels. Éléments dont il sera fondamental de s'assurer de la conformité, au risque, sinon, de voir des facturations bloquées, des remboursements mal effectués pour de simples erreurs d'interprétation. Avec XML, c'est toute une mécanique de suivi et de contrôle de conformité qui doit le plus souvent s'appliquer (notamment dans le cadre des services web).

Dans PHP Saloon, nous allons éviter ces aspects en nous contentant de produire les documents XML pour leur rendu sur des clients web, en dehors de

tout échange. La validation revêt néanmoins un intérêt dans PHP Saloon : s'assurer que le code généré dynamiquement ne comporte pas d'erreur.

Naturellement, ce genre de vérification coûte cher en temps, et en CPU. C'est vrai. Mais en phase de test, de telles vérifications vont nous permettre de détecter d'éventuelles erreurs dans la génération de nos documents XML.

Outre les procédures d'identification et d'inscription, PHP Saloon peut se décomposer en trois types de documents XML :

- les listes de connectés (utilisées tant pour avoir la liste effective des connectés présents que pour celle des amis d'un connecté donné) ;
- le compteur des messages en attente de lecture ;
- les fils des discussions et les messages échangés.

Pour chaque cas, il est possible de définir un document XML type, dont la grammaire exacte sera ensuite précisée au moyen d'une DTD. Ainsi, par exemple, pour le cas de l'échange de message :

Document XML utilisé pour l'échange de messages

```
<?xml version="1.0" encoding="iso-8859-15" ?>
<phpsaloon>
   <connecte uid="2">
      <pseudo>Dupont</pseudo>
      <age>32</age>
      <ville>Paris</ville>
      <region>21</region>
      <cv>Fan de PHP 5</cv>
      <photo>FHOFGLDOD.gif</photo>
      <uid>2</uid>
   </connecte>

   <formulaire uid="2" fil="9">
      <message name="form[message]" type="texte"
               class="envoi" question="Votre message" />
   </formulaire>

   <message type="correspondant">hhhhh</message>
   <message type="utilisateur">jjjjj</message>
</phpsaloon>
```

Ce document XML se décompose en trois zones :

- la description de l'interlocuteur dans la discussion (un connecté) ;
- le formulaire de réponse ;
- l'historique du fil de la discussion.

Cette décomposition se retrouve au final dans l'application, après mise en œuvre des feuilles de style (voir figure 7-2).

Nous avons repris à l'identique le modèle d'élément `connecte` présenté pour notre liste de connectés (voir « Document XML représentant une liste de

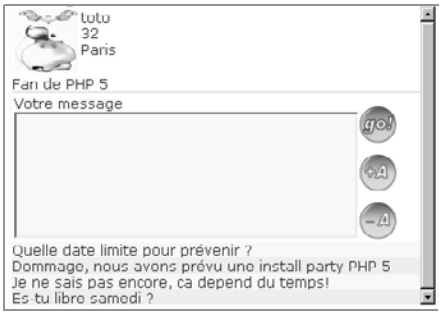


Figure 7–2 Le rendu final du document XML lié aux échanges de messages

connectés », page 125). Cela va nous permettre de simplifier sensiblement la grammaire de nos documents XML tout en améliorant la cohérence d'ensemble.

À partir de ce document type, nous pouvons reprendre la DTD originale présentée précédemment, en la complétant comme suit :

Première version de la DTD du langage XML de PHP Saloon

```
<!ELEMENT phpsaloon ( connectes| (connecte?, formulaire, message*) ) >
<!ELEMENT connectes (connecte)+ >
<!ELEMENT connecte ( pseudo, age?, ville?, region?, cv?, photo?, uid? ) >
<!ATTLIST connecte uid NMTOKEN #REQUIRED >
<!ELEMENT formulaire ( message ) >
<!ATTLIST formulaire fil NMTOKEN #REQUIRED >
<!ATTLIST formulaire uid NMTOKEN #REQUIRED >
<!ELEMENT message ( #PCDATA ) >
<!ATTLIST message class NMTOKEN #IMPLIED >
<!ATTLIST message name CDATA #IMPLIED >
<!ATTLIST message question CDATA #IMPLIED >
<!ATTLIST message type NMTOKEN #REQUIRED >
<!ELEMENT age ( #PCDATA ) >
<!ELEMENT cv ( #PCDATA ) >
<!ELEMENT photo ( #PCDATA ) >
<!ELEMENT pseudo ( #PCDATA ) >
<!ELEMENT region ( #PCDATA ) >
<!ELEMENT uid ( #PCDATA ) >
<!ELEMENT ville ( #PCDATA ) >
```

OUTIL **Générer automatiquement une DTD avec Cooktop**

Cooktop est un outil gratuit vraiment très intéressant, nous en reparlerons d'ailleurs dans le prochain chapitre.

Cet outil propose, comme bien d'autres, de générer une DTD automatiquement à partir d'un document XML. Le résultat est toujours décevant. Mais il faudrait pour cela, partir non pas d'un mais de plusieurs documents, judicieusement choisis de surcroît.

Néanmoins, la DTD ainsi générée peut servir de base de départ. En outre, si la syntaxe précise des DTD ne vous est pas encore familière, cela aura le mérite de proposer un exemple tout prêt.

- http://www.xmlcooktop.com/

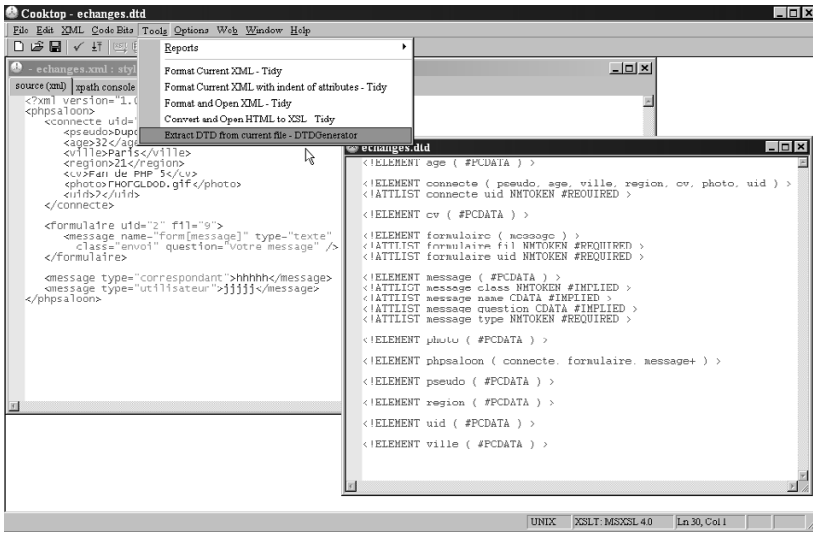


Figure 7–3 Génération automatique d'une DTD avec Cooktop

Au final, en tenant compte de l'ensemble des modules de PHP Saloon (y compris les procédures d'inscription et d'identification), on aboutit à une DTD qui permettra de valider l'ensemble de nos documents :

Version finale de la DTD de PHP Saloon

```
<!ELEMENT phpsaloon ( connectes | nofriend |
                      nomessage | nouveaumessage |
                      (connecte?, formulaire,
                       (info|message*)) ) >

<!ELEMENT connectes (connecte)+ >

<!ELEMENT connecte ( pseudo, motdepasse?, age?,
                     ville?, region?, cv?,
                     photo?, uid? ) >
<!ATTLIST connecte uid NMTOKEN #IMPLIED >
<!ATTLIST connecte titre CDATA #IMPLIED >
<!ATTLIST connecte type CDATA #IMPLIED >

<!ELEMENT formulaire ( message|(connecte+) ) >
<!ATTLIST formulaire fil NMTOKEN #IMPLIED >
<!ATTLIST formulaire uid NMTOKEN #IMPLIED >

<!ELEMENT message ( #PCDATA ) >
<!ATTLIST message name CDATA #IMPLIED >
<!ATTLIST message question CDATA #IMPLIED >
<!ATTLIST message type NMTOKEN #REQUIRED >

<!ELEMENT nofriend EMPTY >

<!ELEMENT nomessage EMPTY >

<!ELEMENT nouveaumessage EMPTY >
<!ATTLIST nouveaumessage nombre NMTOKEN #REQUIRED >

<!ELEMENT age ( #PCDATA ) >
<!ATTLIST age name CDATA #IMPLIED >
<!ATTLIST age question CDATA #IMPLIED >
<!ATTLIST age type NMTOKEN #IMPLIED >

<!ELEMENT cv ( #PCDATA ) >
<!ATTLIST cv name CDATA #IMPLIED >
<!ATTLIST cv question CDATA #IMPLIED >
<!ATTLIST cv type NMTOKEN #IMPLIED >

<!ELEMENT photo ( #PCDATA ) >
<!ATTLIST photo name CDATA #IMPLIED >
<!ATTLIST photo question CDATA #IMPLIED >
<!ATTLIST photo type NMTOKEN #IMPLIED >

<!ELEMENT pseudo ( #PCDATA ) >
<!ATTLIST pseudo name CDATA #IMPLIED >
<!ATTLIST pseudo question CDATA #IMPLIED >
<!ATTLIST pseudo type NMTOKEN #IMPLIED >

<!ELEMENT motdepasse ( #PCDATA ) >
<!ATTLIST motdepasse name CDATA #REQUIRED >
<!ATTLIST motdepasse question CDATA #REQUIRED >
<!ATTLIST motdepasse type NMTOKEN #REQUIRED >

<!ELEMENT info (#PCDATA|a)* >
<!ATTLIST info type (message|error) #REQUIRED >

<!ELEMENT a (#PCDATA) >
<!ATTLIST a href CDATA #REQUIRED >

<!ELEMENT region (#PCDATA|option)* >
<!ATTLIST region name CDATA #IMPLIED >
<!ATTLIST region question CDATA #IMPLIED >
<!ATTLIST region type NMTOKEN #IMPLIED >

<!ELEMENT option (#PCDATA) >
<!ATTLIST option value CDATA #REQUIRED >

<!ELEMENT uid ( #PCDATA ) >

<!ELEMENT ville ( #PCDATA ) >
<!ATTLIST ville name CDATA #IMPLIED >
<!ATTLIST ville question CDATA #IMPLIED >
<!ATTLIST ville type NMTOKEN #IMPLIED >
```

Cette DTD peut être représentée graphiquement de manière un peu plus lisible. La plupart des outils commerciaux d'édition XML le permettent, plus ou moins heureusement (figure 7-4).

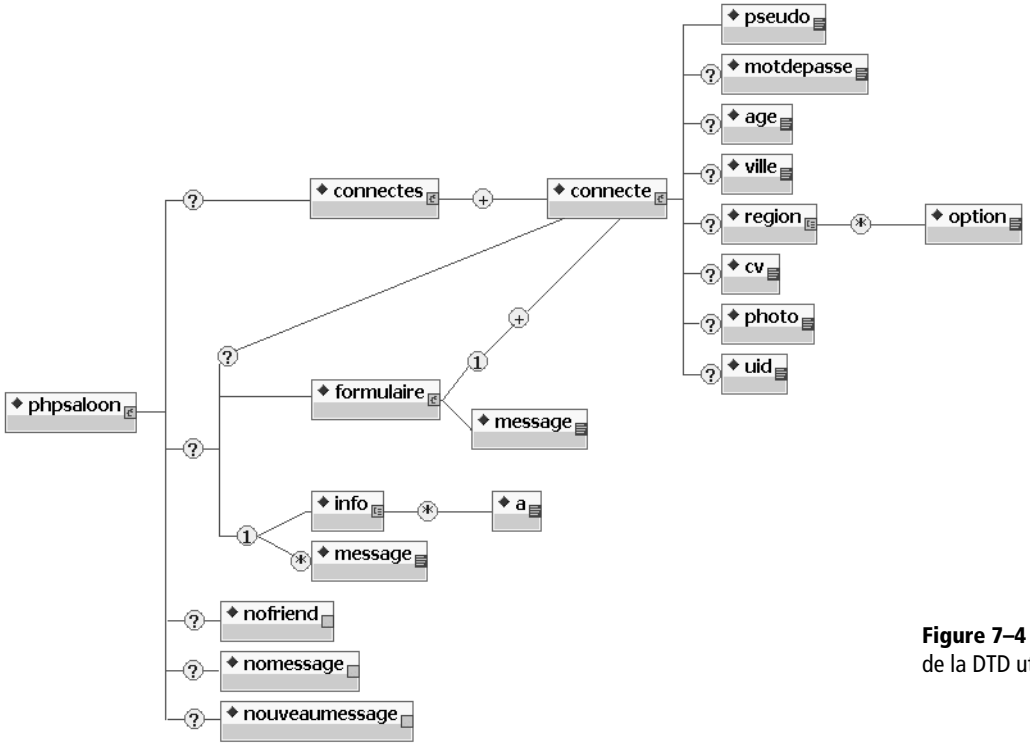


Figure 7–4 Représentation arborescente de la DTD utilisée pour PHP Saloon

NORMES **OASIS, RosettaNet, des organismes pour des langages XML métier**

Si chacun peut définir son langage XML, il va de soi, qu'en pratique, il est indispensable de définir des langages communs, sinon comment échanger ?

C'est pourquoi plusieurs organismes se sont penchés sur la définition de langages XML liés aux besoins des entreprises et de certains métiers. Les deux efforts les plus importants sont portés par deux consortiums :

- OASIS ;
- RosettaNet.

Ces deux consortiums internationaux travaillent désormais ensemble.

- http://www.rosettanet.org
- http://www.oasis-open.org
- http://www.ebxml.org

ALTERNATIVE **Schémas XML et Relax NG**

Utiliser une DTD est un premier pas pour décrire un langage XML. Cette approche demeure quand même un peu simpliste. Aucune méthode de description ne sera jamais parfaite, mais les DTD sont néanmoins assez grossières. Le W3C s'est donc attelé à une méthode plus fine, il s'agit des schémas XML.

De son côté, l'OASIS a défini Relax NG, une autre méthode pour décrire les choses. Actuellement, et probablement pour des raisons historiques, les descriptions avec DTD sont les plus courantes. Cependant, l'utilisation des schémas et de Relax NG est en progression constante.

PHP5 et son extension DOM sont très éclectiques en la matière.

XML oui, mais pourquoi faire dans PHP Saloon ?

Nous avons défini la structure des documents XML qui seront manipulés ou créés dans PHP Saloon. Cependant, sauf extensions particulières de notre système de chat, nous n'échangerons aucune information avec qui que ce soit. Dans ce cas, n'est-il pas un peu excessif de vouloir s'imposer XML ?

Si l'on se restreint à la seule notion d'échange, il est clair que PHP Saloon ne requiert en rien XML mais son utilisation le dotera néanmoins de cordes supplémentaires.

1. En décrivant le contenu avec XML, nous le séparons de manière claire de la forme (c'est-à-dire des opérations de mise en page).
2. Cette séparation permet de proposer plusieurs interfaces, chacune adaptée à un média (téléphone, navigateur classique, navigateur riche).

Les chapitres 9 et 10 montrent des versions XUL/Mozilla et i-mode de PHP Saloon.

Ces deux aspects sont essentiels. Au fil du temps, le langage HTML a été détourné de son objectif et est aujourd'hui principalement utilisé comme langage de mise en page tandis qu'à l'origine, il s'agissait de présenter une information structurée simplement dont la mise en forme devait être confiée à des feuilles de style CSS.

Aujourd'hui, la création de sites dynamiques aboutit donc à un joyeux maelström dans lequel se mélangent l'accès aux données, le code (PHP, Perl...) et la présentation. Le résultat est une maintenance délicate quand elle n'est pas impossible et une adaptation laborieuse aux différents modes de consultation disponibles aujourd'hui.

REGARD DU DÉVELOPPEUR **Web et accessibilité**

Les premières versions du langage HTML s'intéressaient principalement au contenu, et à sa structure (de manière simplifiée), en proposant pour un document la définition d'un titre, de chapitres ou de paragraphes.

Cette ambition originale a été détournée pour rendre le Web plus « avenant ». Au fil du temps, notamment de par la course entre éditeurs de navigateurs, on a augmenté le langage HTML en introduisant des éléments de mise en page, ou en détournant l'utilisation de certains éléments aux fins de mise en page (par exemple, les tableaux).

Avec HTML 4.01 et XHTML, le W3C revient aux sources. Il recentre HTML et surtout XHTML sur la description du contenu et confie les détails de mise en page à des feuilles de style (les fameuses CSS).

Avec cette méthode, c'est toute l'accessibilité du Web qui s'en trouve améliorée. Il devient possible de proposer des styles différents, pour s'adapter à l'outil de navigation, mais aussi et surtout pour prendre en compte les besoins de personnes mal-voyantes, par exemple.

Le W3C accompagne ce recentrage de nouvelles fonctionnalités pour permettre une meilleure navigabilité sans la souris (organisation des formulaires, accesskeys). C'est un pas important pour revenir sur l'exclusion de populations entières, du fait même de ce détournement du langage HTML.

- http://www.w3.org/

Il suffit pour s'en convaincre, d'observer les difficultés avec lesquelles les sites modernes s'adaptent aux deux navigateurs les plus répandus : Microsoft Internet Explorer et Mozilla.

Au final, cette complexité nuit à l'accessibilité du Web. Cela vaut naturellement pour les utilisateurs classiques, qui sont confrontés à des sites plus ou moins récalcitrants en fonction du navigateur utilisé, mais aussi, et surtout, cet état de fait constitue une véritable barrière à l'utilisation du Web par les utilisateurs victimes d'un handicap (difficultés motrices, mal-voyants).

En utilisant XML, et en le transformant dynamiquement pour obtenir le site le plus adapté au visiteur, PHP Saloon va donc répondre au mieux à cette notion d'accessibilité tout en bénéficiant de la conception simplifiée qui résulte d'une meilleure organisation des différentes composantes : données, traitements et présentation.

Document Object Model : une interface disponible dans PHP

Le W3C aurait pu s'arrêter à la définition d'XML. Cependant, un document XML en tant que tel ne sert à rien. Il doit être analysé pour que l'information contenue soit exploitée. Faute de standard en la matière, il y a fort à parier que chacun y serait allé de son propre système d'analyse.

Le W3C a donc défini une modélisation pour les documents XML, et surtout une interface pour manipuler le modèle, et ainsi modifier, lire, détruire les informations des documents XML. Ce standard est DOM (Document Object Model).

Dans DOM, tout document XML est décrit comme un arbre. Chaque élément, ou balise, est représenté sous forme de nœud, le contenu, comme des feuilles.

Les fonctions de l'interface DOM sont donc beaucoup plus simples qu'il n'y paraît. Sous leurs airs rébarbatifs, il ne s'agit en réalité que d'arboriculture : on enlève des branches, on en greffe, on élimine des feuilles pour se tailler un document sur mesure.

Dans PHP Saloon, nous allons très exactement procéder de la sorte, tantôt en modifiant un document XML de référence, pour y placer les informations d'un connecté, tantôt en créant un document à partir de rien et en y ajoutant branches et feuilles.

Qu'il s'agisse de PHP 5 ou de PHP 4, la manière la plus simple, et celle conseillée, pour manipuler un document XML avec DOM consiste à utiliser les objets, même si l'ensemble des fonctions DOM reste disponible pour des raisons de compatibilité.

PHP 4 Attention aux noms des fonctions DOM

Dans PHP 4, l'extension DOM a été largement victime de sa jeunesse. En réalité, elle a été intégrée et rendue disponible rapidement dans PHP, alors même que le nom des fonctions n'était pas stabilisé.
Il faut savoir que le W3C, avec DOM, ne se contente pas de définir une modélisation pour les documents XML et HTML, mais définit aussi une API de manière très précise.
Au fil du temps, les développeurs de PHP ont donc rectifié les erreurs de l'extension DOM pour adopter les noms imposés par le W3C.
Certains anciens noms restent disponibles pour des raisons de compatibilité mais rien ne garantit leur maintien, alors même que l'interface privilégiée pour manipuler les documents XML reste l'interface objet...
La documentation PHP de l'extension DOM liste les changements de noms de manière précise.

DOM, premier contact avec le formulaire d'identification

Pour explorer plus en détail l'utilisation de DOM dans PHP, nous allons définir de A à Z la procédure d'identification à PHP Saloon.

Cette procédure doit :

- proposer le code XML correspondant au formulaire de connexion à PHP Saloon ;
- gérer le traitement de la réponse.

Un des moyens de simplifier cette première étape est de partir sur la base d'un document XML pré-existant qui va nous servir de modèle. Ce document est bien évidemment conforme à la DTD qui vient d'être établie.

Il faut aussi noter que ce document type pourra servir pour la mise au point des transformations XSLT lorsqu'il s'agira de donner une apparence à tout cela (car pour le moment, nous nous limitons à créer le code XML, aucune esthétique, donc).

Le modèle XML de l'identification à PHP Saloon

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<phpsaloon>
  <formulaire>
    <connecte>
      <pseudo name="form[pseudo]" question="Nom de code" type="string">
        <![CDATA[foo]]>
      </pseudo>
      <motdepasse name="form[motdepasse]" question="Mot de passe"
                  type="secretstring">
        <![CDATA[bar]]>
      </motdepasse>
    </connecte>
  </formulaire>
  <info type="message">
    Pas inscrit(e)? <a href="inscription.php">Cliquez-ici</a>!
  </info>
</phpsaloon>
```

Ce document peut être visualisé dans un navigateur (voir figure 7-5) ou dans tout autre outil de création XML.

Comme nous pouvons le constater, la nature arborescente du document est immédiatement mise en évidence dans le navigateur. Le document tel que modélisé dans DOM n'est en rien différent. C'est ce type d'arbre qu'il faut avoir en tête quand il s'agit d'appliquer les méthodes de l'API proposées par DOM.

Pour cette première confrontation avec DOM, nous allons effectuer des manipulations simples, comme le remplacement des valeurs par défaut présentes dans notre modèle `identification.xml` :

B.A.-BA **Majuscules ? Minuscules ?**

Faut-il utiliser des majuscules ou des minuscules pour le nom des fonctions et des variables en PHP ? Peu importe, PHP n'est pas sensible à la casse.

Attention toutefois, quand vous interrogerez PHP sur le nom de variables ou de fonctions, celui-ci les retournera toujours en minuscules. C'est le cas par exemple avec les fonctions :

`get_declared_class()` ou `get_object_methods()`

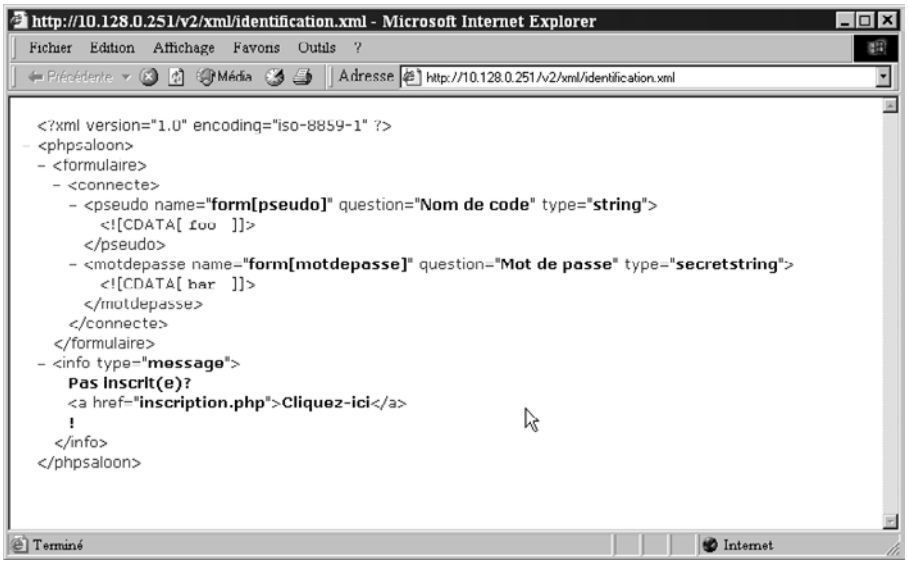


Figure 7–5
Identification.xml dans un navigateur

1. charger le document XML (ce qui revient à dire que ce document sera analysé et transformé en arbre DOM) ;
2. explorer l'arbre correspondant ;
3. modifier la valeur texte du nœud pseudo.

Commençons par un exemple simple :

Premier contact avec un document DOM

```
<?
$tree = new domDocument();
$tree->load('identification.xml');
echo <<<EOF
L'élément racine est : {$tree->documentElement->nodeName},
il comporte {$tree->documentElement->childNodes->length}
élément(s) fils.<br>
EOF;
?>
```

Dans cet exemple, nous créons un objet de la classe `domDocument` avant d'utiliser la méthode `load` pour charger le document en mémoire. Mais cette apparente simplicité masque toute une mécanique complexe : le document XML a été lu, analysé, et transformé en arbre. Un arbre dont l'API DOM nous permet de récupérer la racine (attribut `documentElement`) et le nombre de fils. Avec le document XML précédent, nous obtenons :

```
L'élément racine est : phpsaloon, il comporte 5 élément(s) fils.
```

À RETENIR Comment connaître les méthodes et les attributs disponibles ?

Dans notre premier exemple, nous avons utilisé des attributs (cela vaut pour les méthodes). Les plus curieux l'ont peut-être déjà remarqué, le manuel de PHP manque de précision à ce sujet.
En réalité, la documentation de PHP va progressivement intégrer ces éléments, mais il est peut-être encore plus simple d'aller chercher l'information à la source : dans la recommandation du W3C.
On peut ainsi lire, dans la description de l'objet `Document` que :

- `documentElement` retourne un objet de type `Element`, et qu'il s'agit d'un attribut public en lecture seule.
- l'objet `Element` dispose pour sa part d'un attribut public : `nodeName` lui aussi en lecture seule et permettant d'accéder au nom de la balise.

Une copie de la recommandation du W3C est présentée en annexe de cet ouvrage.

Naturellement, il nous faut aller plus loin et explorer complètement notre document type, puisqu'au final il nous faudra remplacer les pseudos et mots de passe fictifs par ceux du connecté. Cette exploration peut être réalisée en quelques lignes avec une fonction récursive (voir ci-après).

Exploration d'un document XML

```
<?
$tree = new domDocument();
$tree->load('identification.xml');
function domExplore($node, $root = true) {
  if ($root) print "<ol>\n";
  print "<li> {$node->nodeName}<br>\n";
  if ($node->childNodes->length) print "<ol>\n";
  for($i = 0; $i < $node->childNodes->length; $i++)
     domExplore($node->childNodes->item($i), false);
  if ($node->childNodes->length) print "</ol>\n";
  if ($root)   print "</ol>\n";
}
domExplore($tree->documentElement);
?>
```

La fonction `domExplore` n'est guère plus compliquée que le code du premier exemple, elle utilise l'attribut `childNodes` d'un élément pour obtenir une liste des nœuds enfants (`nodeList`). Ce nouvel objet comporte un attribut `length` qui donne le nombre de nœuds et une méthode, `item()`, qui va permettre d'accéder isolément à chacun d'entre eux. Là encore, la recommandation du W3C est d'un précieux secours.

Le résultat de l'exécution est présenté en figure 7-6.

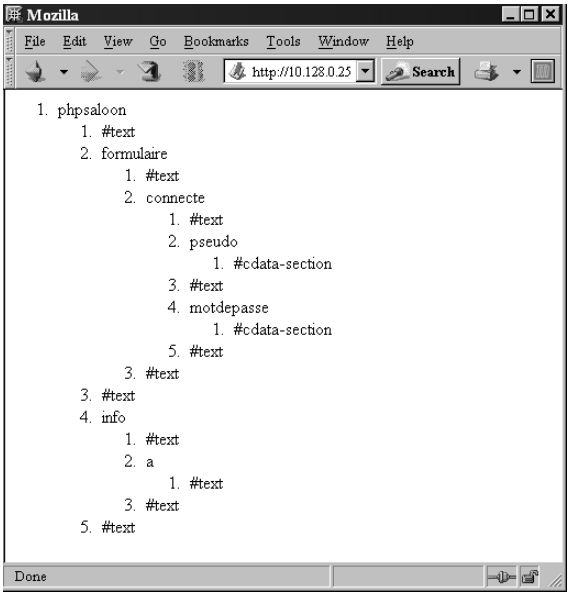


Figure 7–6
Parcours récursif d'un document XML avec `domExplore()`

Il réserve une surprise. On aurait pu logiquement s'attendre à ce que l'arbre exploré soit identique à celui affiché par les navigateurs (voir figure 7-5).

L'origine de cette différence est simple : DOM ne se contente pas de modéliser la structure d'un document XML, il se doit d'en préserver la totalité des informations, ce qui inclut les espaces et les retours à la ligne, notamment entre les éléments. Ainsi, il y a une totale bijection entre le document XML et son modèle DOM.

La recherche des nœuds qui nous intéressent directement en devient plus aléatoire. Par chance, DOM met à notre disposition des méthodes complémentaires, et notamment, `getElementsByTagName` :

Recherche d'éléments particuliers dans un document XML

```
<?
$tree = new domDocument();
$tree->load('identification.xml');
$pseudos = $tree->getElementsByTagName('pseudo');
print $pseudos->item(0)->nodeValue;
?>
```

Dans cet exemple, la fonction `getElementsByTagName` retourne une liste de nœuds réduite au seul élément qui nous importe, l'élément `pseudo` dont le contenu sera affiché (en l'occurrence : « foo »).

Il est alors possible de modifier ce contenu pour le remplacer par le pseudo réel d'un connecté en modifiant la valeur de l'attribut `nodeValue`, modification aisément vérifiable en provocant l'affichage du document modifié :

Modification dynamique d'un document XML

```
<?
$tree = new domDocument();
$tree->load('identification.xml');
$pseudos = $tree->getElementsByTagName('pseudo');
$pseudo = $pseudos->item(0);
$pseudo->nodeValue = 'PHPFan';
print $tree->saveXML();
?>
```

On peut donc imaginer pouvoir procéder de la sorte pour l'ensemble des éléments à mettre à jour dans notre modèle XML.

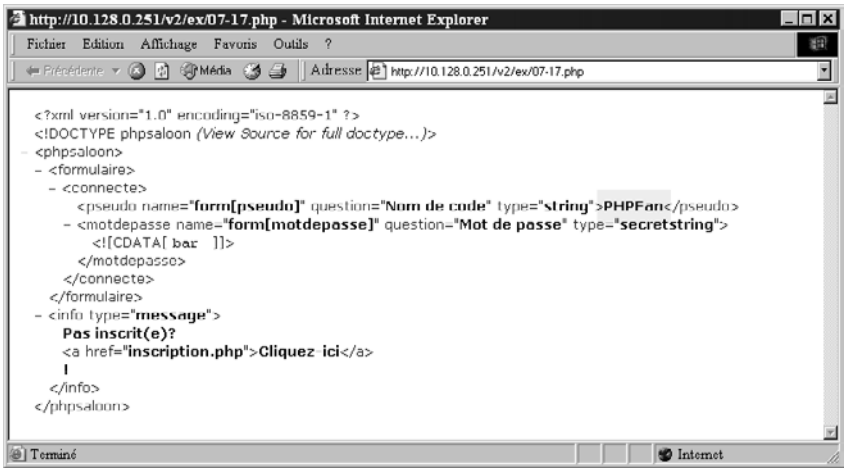


Figure 7–7 Affichage du document XML modifié

PERSPECTIVES

L'implantation DOM de PHP 5 sait aussi interpréter les documents HTML

Dans nos exemples, nous allons exclusivement travailler sur des documents XML qui sont directement manipulés dans PHP Saloon, mais le W3C a défini DOM aussi pour les documents HTML.

L'extension DOM de PHP 5 ne s'arrête donc pas aux seuls documents XML mais est en mesure de charger les documents HTML :

```
<?
$dom = new domDocument();
@$dom->load('http://www.w3.org');
$liste = $dom->getElementsByTagName('title');
print $liste->item(0)->nodeValue;
?>
```

Cette facilité est cependant parfois mise à mal par la très mauvaise qualité des pages HTML. Par ailleurs, il est assez courant de ne pas spécifier l'encodage de ces pages en supposant par défaut un mode (iso-8859-1) qui n'est pas celui de l'analyseur XML (utf-8).

XPath, recherche avancée dans les documents XML

Cette manière de procéder fonctionne, mais elle est à vrai dire ardue et risquée. Difficile de modifier un tant soit peu le document XML sans provoquer de catastrophe, même un espace risquerait de perturber le bon fonctionnement de l'ensemble, ne serait-ce qu'en parcourant l'arborescence à la manière de notre fonction `domExplore`.

On a vu précédemment qu'il était possible de retrouver un élément en le recherchant par son nom. Cette méthode apporte un plus mais reste très insuffisante. Par chance, le W3C a complété XML avec le langage XPath. XPath va nous permettre de rechercher les éléments en fixant des contraintes

beaucoup plus précises portant principalement sur le chemin entre la racine du document et l'élément concerné.

Avec XPath, on pourra, entre autres choses, désigner en une seule opération de recherche l'ensemble des éléments descendants d'un formulaire, nommés `pseudo` et disposant d'un attribut `type`.

Dans cet ouvrage, nous n'aborderons XPath que de manière limitée. Le langage en lui-même est très riche et la recommandation du W3C en est une référence indispensable. Les différentes utilisations possibles dans PHP Saloon donneront un bon aperçu du sujet.

Premières expressions XPath

Nous allons reprendre la situation précédente en tirant profit d'XPath pour ne pas avoir à parcourir notre document `identification.xml`, ni même à rechercher par nom d'élément, et ainsi d'éviter d'en obtenir plusieurs sans pouvoir affiner notre recherche.

Recherche d'un élément (ou d'une liste d'éléments) avec XPath

```
<?
$tree = new domDocument();
$tree->load('identification.xml');
$xpathcontext = new domXPath($tree);
$liste = $xpathcontext->query('formulaire/*/pseudo');
if ($liste->length)
   $liste->item(0)->nodeValue = 'PHPFan';
print $tree->saveXML();
?>
```

Pour ce faire, PHP met à notre disposition la classe `domXPath`,. Celle-ci définit un contexte XPath, sorte d'environnement où vont être évaluées les futures expressions XPath. Ici, nous avons initialisé le contexte avec l'arbre DOM de notre document.

Il suffit alors de demander à obtenir les nœuds désignés par l'expression XPath à l'aide de la méthode `query`. Celle-ci fonctionne sur le même principe qu'une requête SQL sauf que le langage d'interrogation est XPath. Là où une requête SQL retournera des lignes, une expression XPath permettra de désigner un ensemble de nœuds.

Ici, l'expression XPath `formulaire/*/pseudo` désigne l'ensemble des éléments `pseudo` qui sont aussi des « petits-fils » de l'élément `formulaire` et ceci peut importe l'élément intermédiaire.

Les règles les plus simples d'XPath sont donc très proches des expressions régulières utilisées sur les fichiers, le séparateur / lui-même est traditionnellement utilisé dans les chemins de fichiers Unix/Linux (utilisateurs de Windows remplacez / par \ dans votre tête).

À RETENIR XPath et XSLT

Nous explorerons plus en détail les expressions XPath dans le cadre des transformations XSL pour le rendu des documents XML, dès le chapitre suivant. Cependant, XPath mériterait un ouvrage à lui seul.

📖 *XSLT Fondamental*, Philippe Drix, Eyrolles.

Nous sommes donc en mesure de proposer une première version du contrôleur utilisé pour l'identification. Comme l'ensemble des contrôleurs est chargé de créer des document XML, une possibilité intéressante offerte par le modèle objet est de créer une classe contrôleur dérivant de la classe `domDocument`.

Contrôleur pour l'identification, première version

```
<?
require_once'icontroleur.php';
require_once'utilisateur.php';
class controleur extends domDocument implements icontroleur {
  function __construct() {
    parent::__construct();                                          ◂ Initialisation du document DOM.

    $this->load('identification.xml');                              ◂ Chargement du patron XML.

    $this->db = new sqlite_db('phpsaloon');                         ◂ Connexion à la base.

    $context = new domXPath($this);                                 ◂ Construction du contexte XPath.

    if ($_POST['form']) {                                           ◂ Si le formulaire a été validé.

      $this->user = new utilisateur($_POST['form']['pseudo'],       ◂ Création d'un utilisateur.
                  $_POST['form']['motdepasse']);

      if ($this->user->connecte($this->db) {                        ◂ Tentative de connexion.

        header('Location: /site.php');                              ◂ Redirection vers le chat si ok.
        exit(0);
      }
      // une erreur s'est produite, connexion impossible
      $list = $context->query('phpsaloon/formulaire');
      $message = new domElement('info');
      $message->setAttribute('type', 'erreur');
      $message->nodeValue = 'Impossible de vous connecter';
      if (count($list->item(0)->childNodes))
        $list->item(0)->insertBefore($message,
                  $list->item(0)->childNodes[0]);
      else
        $list->item(0)->appendChild($message);
    } else

      $this->user = new utilisateur('','');                         ◂ Création d'un utilisateur par defaut.

    $list = $context->query('formulaire/*/pseudo');                 ◂ Remplacement des éléments du modèle.
    $list->item(0)->nodeValue = $this->user->pseudo;
    $list = $context->query('formulaire/*/motdepasse');
    $list->item(0)->nodeValue = $this->user->motdepasse;
  }
}
?>
```

Le fonctionnement d'une bonne partie des classes `controleur` utilisées dans PHP Saloon sera identique :

1. charger un modèle XML ;
2. tenir compte de la validation d'un formulaire si nécessaire ;
3. rechercher les nœuds dont le contenu est à modifier ;
4. altérer leur contenu.

Notre classe `controleur` est parfaitement conforme à ce séquencement. Elle pourrait d'ailleurs être utilisée en l'état. Mais puisque la plupart des classes `controleur` vont mettre en œuvre les mêmes actions, il est judicieux de créer une classe `controleur` générique qui inclura par exemple :

- une méthode pour ajouter ou modifier un message d'information ;
- une méthode pour simplifier le remplacement du contenu d'un nœud donné.

COMPRENDRE **À propos des sections CDATA**

Quand du code manifestement non conforme à la syntaxe d'un document XML (par exemple un morceau de code C++, Java), doit être néanmoins inclus dans un élément, il n'est pas toujours pertinent de tout encoder brutalement, comme c'est le cas pour l'affectation à l'attribut `nodeValue`.

Le standard XML permet d'insérer des données en l'état dans un document XML avec l'utilisation de sections dites CDATA. Une section CDATA est délimitée par deux marqueurs, un au début, un à la fin, qui vont délimiter une zone au contenu libre :

```
<?xml version="1.0" ?>
<code langage="C">
<[CDATA[
int main(int ac, char *av) {
   int i;
   if (ac < 2)
      exit(1);
   for(i = 1 ; i < ac ; i++)
      printf("Argument %d : %s\n", i, av[i]);
   return 0;
}
]]>
</code>
```

Dans le cas de PHP Saloon, on pourrait choisir de créer pour les différents champs (comme celui de description), une section CDATA. Le code de la fonction `replaceValue()` pourrait alors s'écrire :

```
protected function replaceCDATA($element,
            $nouveautexte = '') {
      // effacer tout ce qui pourrait
      // se trouver là

      foreach($element->childNodes as $child)
         $element->removeChild($child);

      if ($nouveautexte) {
         // créer le noeud CDATA

         $cdata = new domCDATASection(
                  utf8_encode($nouveautexte));
         $element->appendChild($cdata);
      }
}
```

On obtient alors une classe `controleur` :

Contrôleur générique

```
<?
require_once'icontroleur.php';
class controleur extends domdocument implements icontroleur {
   protected $form;
   function __construct() {
      parent::__construct();
      $this->form = $_POST['form'];
   }
   protected function insertMessage($path, $message_texte) {
      $xpath = new domXPath($this);
      $a = $xpath->query($path);
      $msg = new domelement('info');
      $text = new domText();
      $a = $a->item(0);
      $text->appendData(utf8_encode($message_texte));
      $a->insertBefore($msg,$a->childNodes->item(0));
      $msg->appendChild($text);
      $msg->setAttribute("type","erreur");
   }
   protected function replaceCDATA($element, $nouveautexte = '') {
      for($i = 0; $i < $element->childNodes->length; $i++)
         $element->removeChild($element->childNodes->item(0));
      if ($nouveautexte)
         $element->nodeValue = $nouveautexte;
   }
}
?>
```

Au final, notre classe identification peut donc être définie de manière simplifiée comme suit

Contrôleur pour l'identification, version finale

```
<?
require'utilisateur.php';
require'controleur.php';
class identification extends controleur {
   function __construct() {
      parent::__construct();
      $this->load('xml/identification.xml');
      $xpath = new domXPath($this);
      $db = new sqlite_db('test');
      if ($this->form) {
         $user = new utilisateur($this->form['pseudo'],
                                 $this->form['motdepasse']);
         if ($user->connecte($db)) {
            header('Location: site.php');
            exit();
         }
```

```
            $this->insertMessage('/phpsaloon/formulaire',
            'Votre connexion a échoué. Merci de réessayer
             ultérieurement.');
        }
        $a = $xpath->query('formulaire/*/pseudo');
        $this->replaceCDATA($a->item(0),$this->form['pseudo']);
        $a = $xpath->query('formulaire/*/motdepasse');
        $this->replaceCDATA($a->item(0),$this->form['motdepasse']);
    }
}
?>
```

Construction de document XML à partir de zéro

Pour tout le code étudié jusqu'à présent, nous avons utilisé un modèle XML et travaillé directement sur les nœuds du patron. Dans certains cas, il est plus efficace de construire le document final à partir de rien, en ajoutant les nœuds du document les uns après les autres. Dans PHP Saloon, il en va ainsi pour les listes de connectés ou pour le document indiquant à chaque instant le nombre de messages restant à lire.

Pour ce dernier cas, deux types de documents XML peuvent être produits :

```
<phpsaloon><nouveaumessage nombre="3" /></phpsaloon>
```

lorsqu'au moins un message est en attente pour être lu, ou quand aucun message n'est en attente :

```
<phpsaloon></nomessage></phpsaloon>
```

Ces documents simples peuvent être créés nœud par nœud, comme l'ont déjà montré les fonctions `insertMessage()` et `replaceCDATA()`.

En reprenant notre classe `controleur` de référence, on peut donc définir un contrôleur spécifique pour ce petit document :

Contrôleur pour les messages en attente

```
<?
require_once'icontroleur.php';
require_once'controleur.php';
class messages extends controleur {
    function __construct() {
        parent::__construct();
        $phpsaloon = new domelement('phpsaloon');
        $this->appendChild($phpsaloon);
        $db = new sqlite_db('test');
        // recherche des messages non lus (statut = 0)
```

```
      $nb = $db->single_query('select count(*) from messages
                               where destinataire = '
                              . $_SESSION['uid'] . ' and statut = 0');
      if ($nb) {
         $newmessage = new domelement('nouveaumessage');
         $phpsaloon->appendChild($newmessage);
         $newmessage->setAttribute('nombre' , $nb);
      } else {
         $nomessage = new domelement('nomessage');
         $phpsaloon->appendChild($nomessage);
      }
   }

  }
?>
```

Validation des documents créés

L'inconvénient avec les documents dynamiques, comme celui qui va être produit par la classe `identification`, est que, par essence, on ne peut totalement garantir la conformité du document généré à un modèle de départ. Plus précisément, les multiples manipulations qui sont réalisées sur l'arbre peuvent altérer le format du document ; autant d'incertitudes qui n'ont pas de raison d'être pour un document statique.

Par chance, nous avons défini la DTD des documents manipulés dans PHP Saloon. Dynamiques ou pas, ceux-ci doivent s'y conformer et PHP permet de s'en assurer.

Tous les processus de validation ont été intégrés à la classe `domDocument`, y compris la validation à base de DTD et nous disposons de la méthode `validate()`.

Dans une phase de test, le code de la page d'identification pourrait donc prendre la forme suivante :

```
require_once'identification.php';
$document = new identification();
$document->validate('phpsaloon.dtd') or die("Oops! Le document généré
est non conforme.");
// suite du code effectuant la transformation XSL.
```

Cette validation provisoire permettra l'identification des défauts dans l'implantation des documents et pourra être simplement commentée en production.

MÉTHODE Les tests de conformité sont cruciaux

On peut être tenté de se dire : à quoi bon tous ces tests. Cela prend du temps, coûte du CPU... Toutes ces objections sont vraies et la validation à la volée d'un document dynamique représente tout simplement un gouffre en termes de ressources. Cependant, il convient de replacer les échanges XML dans un contexte plus large. Ceux-ci vont prendre une place considérable en entreprise, ils feront notamment l'objet de facturations. La vérification de conformité aura alors un impact financier (pourquoi payer une information qui ne respecte pas les critères prévus), et d'autres documents transporteront des informations vitales.
Dans PHP Saloon, les choses ne sont pas trop graves, au pire les feuilles de style XSL du chapitre suivant seront incapables de transformer le flux XML en page HTML compréhensible. Rien de bien méchant, mais tout n'est pas toujours aussi inoffensif.

ALTERNATIVE **D'autres outils de validation**

Si la définition de DTD est un élément déjà très utile pour s'assurer de la conformité des documents XML à une semantique donnée, la méthode reste plutôt primitive et très largement insuffisante dans un contexte plus exigeant.

Le W3C a donc proposé un outil de description alternatif : les schémas, plus complexes à mettre en œuvre, ils décrivent avec plus de finesse les langages XML. PHP 5 permet la validation avec de tels schémas via la méthode `schemaValidate()`.

Par ailleurs, dans le même temps, l'OASIS a défini Relax NG comme un autre langage de description sémantique. Dans PHP, la méthode `relaxNGValidate()` est disponible.

Les deux approches, Relax NG et Schemas, sont puissantes, le choix est en ce domaine sujet à querelles de chapelles. Pour plus d'informations, on pourra consulter le site xmlfr.org qui propose une introduction à Relax NG et naturellement le site du W3C pour la recommandation sur les schémas XML.

- http://xmlfr.org/actualites/xmlfr/040116-0001
- 🕮 *Schémas XML*, Jean-Jacques Thomasson, Eyrolles
- 🕮 *Relax NG*, Éric Van der Vlist, O'Reilly

SimpleXML, une alternative très séduisante

DOM, nous le voyons, est d'une grande puissance, chaque nœud d'un document XML peut être altéré et modelé précisément. Par ailleurs, derrière une apparente complexité, DOM a l'avantage de suivre rigoureusement une logique cohérente. De plus, l'apprentissage de DOM peut constituer un bon investissement car cette interface est désormais le moyen privilégié d'interagir avec les navigateurs modernes. DOM dispose pour cela d'une interface JavaScript en tout point identique à celle disponible pour PHP.

Cependant, il est vrai que la manipulation quelque peu tortueuse de l'arbre des nœuds peut paraître insurmontable. PHP dispose dans ce cas d'une autre extension, promise à un grand avenir, eu égard à sa simplicité : SimpleXML.

Avec SimpleXML, PHP va construire non pas une arborescence de nœuds DOM mais, plus directement, une arborescence d'objets PHP prêts à l'emploi. Il devient alors élémentaire de travailler sur les documents XML.

La valeur de chaque nœud est rendue accessible sous forme d'attributs des objets et les attributs de ces derniers sont disponibles en utilisant la syntaxe des tableaux.

PHP 4 **SimpleXML est aussi disponible**

Les dernières versions de PHP 4 disposent elles aussi de l'extension SimpleXML. On peut donc sans difficulté adapter des développements existants.

Reprenons le modèle qui nous a servi pour l'identification (voir « Le modèle XML de l'identification à PHP Saloon », page 132). Le code suivant va nous permettre de réaliser en un tour de main les mêmes opérations sans utiliser DOM :

Manipulation d'un document XML avec SimpleXML

```
<?
$racine = simplexml_load_file('identification.xml');
$racine->formulaire->connecte->pseudo = 'PHPFan';
$racine->formulaire->connecte->motdepasse = 'secret';
$racine->formulaire->connecte->pseudo['question'] = 'Pseudo?';
print $racine->asXML();
?>
```

Difficile de faire plus simple, d'autant que pour des manipulations plus complexes, SimpleXML prend également en charge les requêtes XPath :

```
$tableau = $racine->xsearch('formulaire/*/pseudo');
```

Dès lors, on peut se demander pourquoi s'encombrer de DOM ? Deux raisons à cela.

Premier point, s'il est possible de transformer un document DOM en objets SimpleXML, l'inverse n'est pas possible. Cette bijection peut paraître superflue, après tout SimpleXML se suffit à lui-même ! Oui... et non. Avec SimpleXML, il est impossible de réaliser les transformations XSL et plus généralement impossible, par exemple en matière de validation, d'accéder à la totalité des capacités de DOM.

Second point, il est sympathique de créer cette avalanche d'objets, mais pour les documents légèrement volumineux, les choses vont terriblement se corser. Le modèle objet n'est pas des plus véloces et par ailleurs, il est relativement vorace en mémoire : autant de ressources, mémoire et CPU qui ne sont pas infinies...

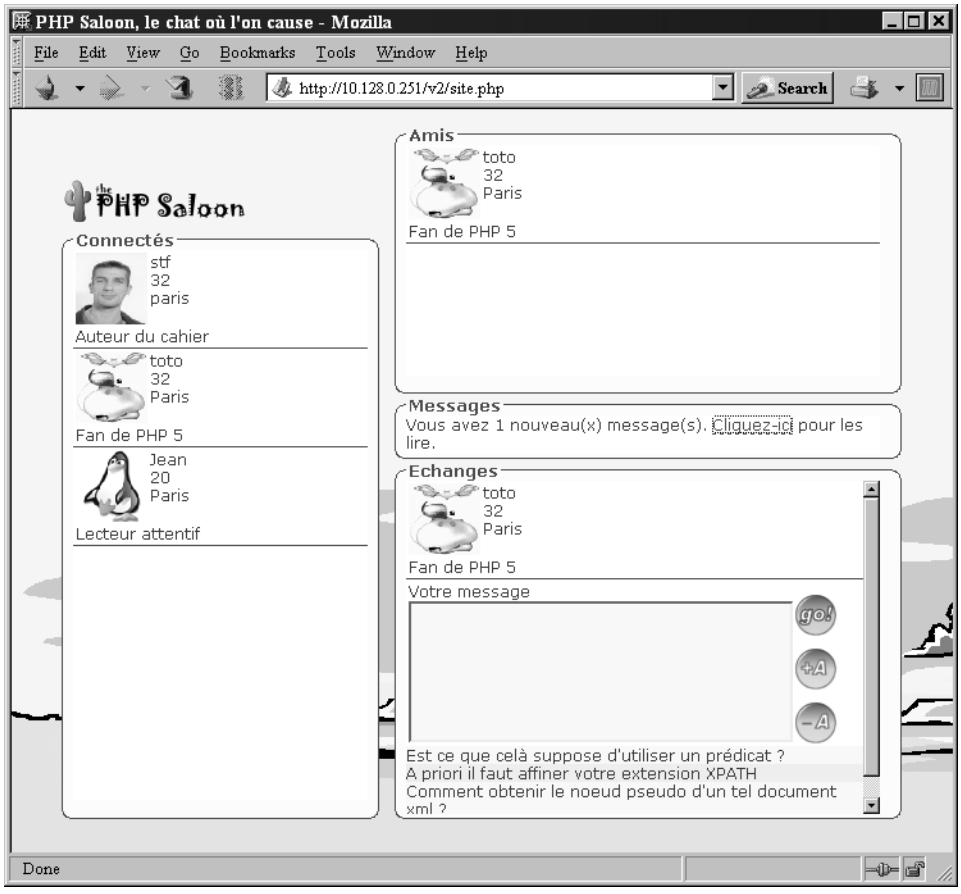
SimpleXML est une alternative intéressante pour manipuler des documents de taille raisonnable dès lors qu'aucune interaction trop complexe n'est requise. Ceci correspond tout à fait à la nature des échanges dans le cadre des services web (avec SOAP, XMLRPC).

Pour PHP Saloon, outre l'aspect didactique, la nécessité de transformer nos documents plaçait DOM en position de choix.

En résumé...

Dans ce chapitre, nous avons vu comment construire avec XML un contrôleur propre à chaque situation. Pour une application comme PHP Saloon, il aurait été possible de mélanger les données de présentation avec la logique applicative. Cela nous aurait permis d'échapper à la relative complexité de DOM et du langage XML. Cependant, et c'est l'objet du chapitre à suivre, il nous aurait fallu alors récrire l'application pour chaque cible (mobile, navigateur...).

chapitre 8


PHP Saloon, le chat où l'on cause - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://10.128.0.251/v2/site.php
Search
the PHP Saloon
Connectés
stf
32
paris
Auteur du cahier
toto
32
Paris
Fan de PHP 5
Jean
20
Paris
Lecteur attentif
Amis
toto
32
Paris
Fan de PHP 5
Messages
Vous avez 1 nouveau(x) message(s). Cliquez-ici pour les lire.
Echanges
toto
32
Paris
Fan de PHP 5
Votre message
go!
+A
-A
Est ce que celà suppose d'utiliser un prédicat ?
A priori il faut affiner votre extension XPATH
Comment obtenir le noeud pseudo d'un tel document xml ?
Done

Affichage sur mesure avec XSLT

Le bénéfice apporté par l'utilisation d'XML pourrait sembler marginal s'il se limitait à l'obtention d'un découpage élégant de PHP Saloon. Or, notre XML peut être transformé pour générer plusieurs types de rendus.

Comme nous le verrons, XML et les transformations XSL ouvrent la voie vers un Web accessible et adapté à chacun.

SOMMAIRE

MOTS-CLÉS

- XPath
- template
- libXSLT
- FO

Principe général

Dans le chapitre précédent, nous nous sommes attachés à produire des documents XML (par ailleurs, conformes à notre DTD). De fait, nous disposons d'un langage complet et de documents grammaticalement corrects.

Comme leur nom l'indique, les transformations XSL vont transformer ces documents en de nouveaux documents selon un patron et une feuille de style qu'il nous faudra au préalable définir.

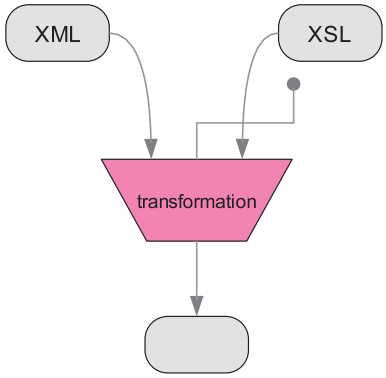


Figure 8–1 Le processus de transformation

COMPRENDRE **Créer des documents binaires avec FO et FOP**

Comme les feuilles de style sont elles-mêmes décrites sous forme de langage XML, il est impossible de produire autre chose que des documents texte avec les transformations XSL. L'exemple le plus fâcheux est l'impossibilité de créer des documents PDF, ou PostScript pour l'impression.

Le W3C apporte une solution à ce problème avec FO (Formatting Objects). FO est un langage de mise en page XML. Au lieu de produire directement un document PDF ou HTML, notre transformation XSL va créer une description en FO de la mise en page et du contenu du document. Cette description très précise sera ensuite confiée à un convertisseur spécifique (un par format). L'outil le plus connu dans ce domaine est FOP, proposé par l'Apache Foundation.

On pourrait donc imaginer pouvoir toujours utiliser FO, même pour HTML. La réalité est autre. FO offre une finesse adaptée pour les documents imprimés mais inutile dans les cas de documents web, d'autant que ce système de double transformation diminue naturellement les performances et la réactivité du rendu.

- http://xml.apache.org/fop/

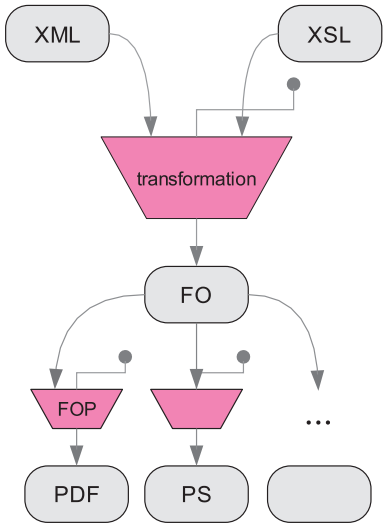


Figure 8–2
Transformations avec FO

Cette feuille de style est décrite dans un langage XML, qui est, là encore, défini dans une recommandation du W3C : XSL ou eXtensible Stylesheet Language.

Le processus de transformation ne limite en aucune façon la nature du document produit. Il peut s'agir de n'importe quel document texte. Une transformation XSL peut donc produire un nouveau document XML, à la manière d'un convertisseur.

Ce type de transformation sera d'un grand recours dans les échanges entre systèmes informatiques, et plus particulièrement pour l'interopérabilité entre services web. Dans PHP Saloon, notre objectif n'est pas de reproduire du XML mais de générer un format de sortie adapté pour le Web : (X)HTML.

L'intérêt du processus est immédiat. Comme avec les CSS, mais avec bien plus de latitude, il nous suffira d'adapter les feuilles de style au type de navigateur détecté pour offrir un niveau de confort optimal. Cette adaptation est réalisable sans même toucher au cœur de notre application.

Nous retrouvons donc tout l'intérêt de l'architecture MVC, avec un découpage précis des différents rôles. Des rôles, en termes de composants informatiques, mais aussi en termes de profils de compétences. Nous avons ainsi séparé des activités qui ne relèvent pas, le plus souvent, des mêmes personnes.

Ce découpage, qui peut paraître artificiel de prime abord, permet à chaque acteur du projet de regagner en liberté d'action, les développeurs pour le développement de la logique métier, les graphistes pour l'élaboration du *Look* & *Feel* de l'application et des feuilles de style.

L'intérêt des transformations XSL réside moins dans leur simplicité propre que dans la méthode implicite, avec à la clé :

- une meilleure interopérabilité ;
- un découpage cohérent du développement.

À RETENIR CSS vs XSLT

Les transformations XSL peuvent sembler complexes, surtout si l'on manie déjà des feuilles de style, de type CSS.
En effet, avec HTML 4 et XHTML 1, le W3C a déjà recentré HTML sur le contenu, laissant une grande partie de la forme aux CSS.
Cette objection est fondée. Mais attention, CSS reste tributaire du langage HTML, il n'est pas question, par exemple, de vouloir se lancer dans le WAP ou même i-mode...

Instructions PHP mises en œuvre

Le principe des transformations XSL est donc d'un maniement aisé. On ne parle que d'une moulinette. L'interface désormais disponible dans PHP 5 est elle-même d'une extrême simplicité :

- une classe de référence pour le processeur XSLT (le moteur qui réalise les transformations) ;
- deux ou trois méthodes pour lire les feuilles de style et exécuter une transformation.

L'ensemble est intimement lié à DOM, ce qui est logique dans la mesure où les feuilles de style elles-mêmes sont des documents XML.

PHP 4 Des possibilités, mais beaucoup de tracas

Avec PHP 4, vous pouvez également transformer des documents avec XSL. Hélas, l'API a été victime de sa jeunesse et les utilisateurs de PHP ont dû subir les contrecoups de chaque évolution.
En fonction de la version de PHP 4 (pré ou post 4.1), différentes options sont possibles, toutes fondées sur Sablotron, un processeur XSLT développé par la société GingerAll.
L'interface reste relativement accessible, même s'il n'est plus question d'objet. Par ailleurs, aucune interopérabilité avec DOM n'est disponible.

▸ http://www.gingerall.com/

Le code PHP type que nous utiliserons dans PHP Saloon est le suivant :

```
<?
$processeur = new XSLTprocessor();
$style = new domDocument();
$style->load('style.xsl');
$processeur->importStyleSheet($style);
print $processeur->transformToXML($document_phpsaloon);
?>
```

Création du processeur.

Analyse de la feuille de style avec DOM (le résultat est un document DOM).

Association du style au processeur.

Exécution de la transformation sur un document DOM (par exemple, un document généré lors du chapitre précédent).

Nous avons intérêt à factoriser ce code en créant une classe `vue`, réutilisable en l'état pour toutes les pages de PHP Saloon. Nous tirerons profit, une fois encore, du modèle objet en étendant la classe `XSLTprocessor` :

```
<?
require_once'i/ivue.php';
class vue extends XSLTprocessor implements iVue {
     function transform($xml, $xslfile) {
          $xsl = new domDocument();
          $xsl->load($xslfile);
          $this->importStylesheet($xsl);
          return $this->transformToXML($xml);
     }
}
?>
```

Il suffira alors d'instancier dans chaque page un objet de classe `vue` pour réaliser le rendu final. Si l'on reprend le cas de la page d'identification, on obtient un code limpide :

```
<?
require_once  'inc/identification.php';
require_once  'inc/vue.php';
$controleur = new identification();
$vue = new vue();
print $vue->transform($controleur, 'style/identification.xsl);
?>
```

Définition du contrôleur et de la vue pour cette page.

Construction du document DOM représentant la page.

Création du processeur XSLT.

Réalisation de la transformation et affichage.

Un code PHP quasi identique sera utilisé pour toutes nos pages en adaptant, à chaque fois, le type de contrôleur instancié et la feuille de style requise.

ALTERNATIVE **Laisser le navigateur réaliser les transformations XSL**

Dans PHP Saloon, nous avons pris le parti de réaliser nous-mêmes les transformations XSL. Ceci est d'autant plus aisé que PHP 5 a pérennisé une interface efficace et simple.

Cependant, alors que la plupart des navigateurs récents sont en mesure de réaliser eux-mêmes les transformations XSL, nous aurions tout à fait pu nous contenter de laisser cette charge au navigateur et donc, au poste client.

C'est d'ailleurs l'option qui avait été choisie dans l'étude de cas du cahier du programmeur PostgreSQL, là où PHP 4 offrait encore un support brouillon des transformations.

C'est une méthode très simple, puisqu'il suffit d'indiquer dans le document XML la feuille de style à appliquer :

```
<?xml-stylesheet type="text/xsl" href="style/identification.xsl" ?>
```

L'inconvénient de la méthode est d'être totalement ou en partie inadaptée aux navigateurs plus ou moins exotiques disponibles sur certaines plates-formes, en particulier les équipements mobiles.

Dans ce cas, le document est tout simplement inaccessible, ce qui constitue un comble pour une application web conçue autour d'XML et qui devrait donc garantir un bon niveau d'interopérabilité et d'accessibilité.

On peut naturellement considérer que les publics exclus sont très minoritaires. C'est évidemment peu éthique, mais surtout, un des aveugles les plus connus du monde est alors exclu : Google. En effet, les moteurs de recherche ne réalisent en général aucune action particulière sur les pages. Au final, le risque de ne pas être référencé est grand.

Constructions des templates de PHP Saloon

Une fois le code PHP mis au point, nous pouvons entrer dans le vif du sujet : le contenu des feuilles de style, celui-là même qui sera transformé en code HTML. Nous n'allons pas explorer de manière exhaustive XSL, qui mérite à lui seul un ouvrage. Nous nous contenterons d'en saisir l'essentiel : la structure, les instructions les plus courantes, autant de clés indispensables pour approfondir ensuite le sujet.

📖 Philippe Drix, *XSLT fondamental*, Eyrolles, 2002.

Structure d'une feuille de style XSL

Comme évoqué précédemment, une feuille de style est un document XML. Et comme dans tout document XML, il existe donc un nœud racine. Pour les transformations XSL, il s'agit de `stylesheet` (dans le code XML de PHP Saloon, il s'agissait de `phpsaloon`). En outre, on utilise la plupart du temps l'espace de nom `xsl` pour distinguer clairement les instructions XSL des balises.

NORMES XSLT, XPath, FO

Pour des raisons pratiques, nous mettons principalement en avant le langage XSLT, explicitement conçu pour transformations. L'environnement XSL complet défini par le W3C est en réalité une famille de trois recommandations :

- XSLT, qui vient d'être évoqué ;
- XPath utilisé pour désigner des morceaux de documents XML ;
- FO (Formatting Objects), un langage XML de mise en page adapté à la production de tout document nécessitant une mise en page précise, par exemple pour l'impression.

▸ http://www.w3.org/Style/XSL/

Une feuille de style XSL se présente donc toujours comme suit :

```
<?xml version="" encoding="" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <?-- instructions de transformation ... -->
</xsl:stylesheet>
```

La syntaxe définissant l'espace de nom `xsl` est quelque peu exotique, mais il s'agit juste de désigner celui-ci de manière unique avec un identifiant sous forme d'URL. Pour HTML, on utiliserait :

```
xmlns:html="http://www.w3.org/TR/REC-html40"
```

On pourrait ainsi mélanger des instructions et des balises résultant de plusieurs langages différents. Nous resterons classiques, pour PHP Saloon.

Des règles, des arbres et des chemins

Une fois définie cette coquille, qu'en est-il des instructions XSLT en elles-mêmes ? Première consigne : oublier les langages procéduraux classiques. En effet, avec XSLT, on ne définit pas un programme et des instructions qui s'enchaînent, mais on écrit des règles qui vont décrire les transformations à appliquer sur des parties du document XML d'origine. Pour les plus curieux, voici un exemple de feuille de style XSLT :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="formulaire//*[@type = 'texte']">
     <div>
          <xsl:value-of select="@question" />
     </div>
     <xsl:element name="textarea">
          <xsl:attribute name="name">
               <xsl:value-of select="@name"/>
          </xsl:attribute>
          <xsl:attribute name="class">
               <xsl:choose>
                    <xsl:when test="@class">
                         <xsl:value-of select="@class"/>
                    </xsl:when>
                    <xsl:otherwise>champ</xsl:otherwise>
               </xsl:choose>
          </xsl:attribute>
        <xsl:value-of select="."/>
     </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Pendant la transformation, le processeur XSLT (c'est-à-dire le programme chargé d'effectuer la transformation) va parcourir l'arbre du document XML à transformer et, pour chaque nœud, déterminer une règle applicable.

D'emblée, on peut retenir que l'ordre d'écriture des règles n'a pas d'importance. En effet, pour un nœud donné, le processeur XSLT choisira la règle adaptée et pas simplement la première règle venue ou la suivante dans la liste.

Comment le processeur décide-t-il de la règle à appliquer ? En fonction d'une expression XPath, ou de celles que nous avons déjà rencontrées. Chaque règle est identifiée par une expression XPath donnée. Si le nœud courant valide l'expression, alors la règle est applicable. Il est donc très facile de définir des règles pour transformer des ensembles entiers du document XML d'origine.

La syntaxe adoptée pour une règle (on parle de template) est la suivante :

```
<xsl:template match="formulaire/connecte">
  <?-- balises et contenu à ajouter dans l'arbre de sortie -->
</xsl:template>
```

L'attribut `match` désigne l'expression XPath qui devra être validée pour que notre règle soit valide. Le contenu de l'élément `template` contient des portions du document final (portions adaptées à la zone que désigne l'expression XPath).

Avec ce système de règles, le processeur XSLT va construire progressivement, tout en parcourant le document d'origine, un document de sortie.

COMPRENDRE Ordre des règles

Dire que l'ordre n'a pas d'importance serait approximatif. Ceci est vrai en général, mais si plusieurs règles peuvent s'appliquer sur un nœud donné, sauf priorité particulière, le processeur choisit la première venue dans le fichier...
On pourrait donc dire que l'ordre ne compte pas, mais définit néanmoins un niveau de priorité en cas de concurrence.

ALTERNATIVES Systèmes de templates vs XML/XSLT

La méthode mise en œuvre avec les transformations XSL est élégante, mais parfois perturbante. Il faut bien en identifier la mécanique, au risque de perdre son temps à reproduire des schémas inadaptés à ce type de langage (on parle de langage fonctionnel, comme l'est LISP, par exemple).

D'autres systèmes de templates ont donc été définis pour PHP, aucun, cependant, ne dispose de la même souplesse que le couple XML/XSLT. En effet, il s'agit dans la quasi-totalité des cas de définir des modèles de pages en leur intégrant des instructions (qui dépendent de l'outil utilisé) qui automatisent la production d'une partie du code de la page finale.

Ces modèles sont lus par PHP, les instructions analysées, et l'interpréteur PHP lui-même produit alors la page finale. Le tout est le plus souvent couplé à un système de cache pour de meilleures performances.

Dans cette catégorie, il suffit de citer Smarty, promu par l'Apache Software Foundation (dont PHP est un projet) ou encore Templeet.

Mais que penser de ces outils ? Certains vous diront que le couple XML/XSLT est un cauchemar, trop complexe, qu'XSL est le pire langage jamais inventé. Certes, le W3C le reconnaît bien volontiers, XSL pourrait (et va) subir un léger lifting pour être plus abordable. Cependant, et il s'agit ici d'une opinion toute personnelle, les systèmes de templates se rapprochent plus de la macro C que d'un réel environnement de conception pour les applications web.

D'une part, les syntaxes adoptées par ces outils sont le plus souvent tout aussi exécrables que celle d'XSL (il s'agit en effet de simplifier l'analyse par PHP). D'autre part, là où avec XML nous avons pu totalement découpler logique métier et apparence, les systèmes de templates se contentent de déplacer le problème. La logique métier n'est plus mélangée au langage HTML, mais aux instructions de l'outil de template, quand ce n'est pas à un soupçon de PHP complémentaire...

Certes, XML et XSLT réclament un effort, mais cet effort peut être salvateur alors que l'avenir est à l'interopérabilité, aux services web et, en ce sens, la qualité des développements n'est plus négociable.

- http://smarty.php.net
- http://www.templeet.org

Transformation de la page d'identification

Le mécanisme que nous venons de décrire s'applique à toutes les pages de PHP Saloon. Considérons le cas de la page d'identification. Le code XML produit a déjà été défini :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<phpsaloon>
  <formulaire>
    <connecte>
      <pseudo name="form[pseudo]" question="Nom de code"
              type="string">
        <![CDATA[foo]]>
      </pseudo>
      <motdepasse name="form[motdepasse]" question="Mot de passe"
                  type="secretstring">
        <![CDATA[bar]]>
      </motdepasse>
    </connecte>
  </formulaire>
  <info type="message">
    Pas inscrit(e)? <a href="inscription.php">Cliquez-ici</a>!
  </info>
</phpsaloon>
```

Il est possible d'identifier quatre opérations distinctes si l'on veut produire une page HTML :

1. générer le squelette d'une page HTML classique ;
2. générer le squelette d'un formulaire ;
3. pour chaque champ, générer le code équivalent en HTML ;
4. ajouter un message.

Toutes ces opérations vont être réalisées avec des règles XSL. Nous allons les définir les unes après les autres, avant de détailler le fonctionnement global de l'ensemble.

Le squelette de la page

Pour cette règle, nous devons identifier deux éléments : l'expression XPath et le contenu. L'expression XPath est on ne peut plus simple, elle désigne simplement la racine du document DOM d'origine, car notre squelette HTML englobe tout le reste.

Le prototype de notre règle sera donc :

```
<xsl:template match="/">
     <?-- contenu à définir -->
</xsl:template>
```

Le contenu ne présente rien d'original, il s'agit de reprendre les balises `html`, `head`, `body` et `title` bien connues, d'où le contenu suivant :

```
<xsl:template match="/">
   <html>
   <title>Bienvenue dans PHP Saloon</title>
   <body>
   </body>
   </html>
</xsl:template>
```

Naturellement, si nous en restons là, la seule chose qui sera produite risque d'être une page vide ! Pour indiquer au processeur XSLT de continuer son analyse de l'arbre DOM original, nous allons nous servir de l'instruction `apply-templates` et en profiter pour ajouter quelques éléments de présentation entre les deux balises `body` :

```
<xsl:template match="/">
   <html>
   <head>
     <title>Bienvenue dans PHP Saloon!</title>
     <link rel="stylesheet" href="style/style.css" type="text/css" />
   </head>
   <body>
      <fieldset>
          <legend>
              <img src="img/phpsaloon.gif" alt="Logo PHP Saloon"/>
          </legend>
          <xsl:apply-templates />
      </fieldset>
   </body>
   </html>
</xsl:template>
```

Cette instruction va relancer le parcours de l'arbre, en partant des enfants de la racine qui vient d'être traitée. Il est possible de restreindre l'ensemble des nœuds qui seront parcourus :

```
<xsl:apply-templates select="formulaire" />
```

Avec cette précision, le processeur aurait certes repris son parcours, mais en ignorant le message d'information.

Le message d'information

Nous allons prendre en compte ce message avec une règle dédiée. Il nous faut créer un élément HTML qui mettra en valeur le texte du message dans le document final. Par ailleurs, si l'on se souvient de notre DTD, nous ne

devons pas omettre deux types de messages : standard, à caractère informatif, et d'erreur, dont la mise en valeur doit être plus accentuée.

Voici une possibilité de règle XSL :

```
<xsl:template match="info">
    <xsl:element name="div">
       <xsl:attribute name="class"><xsl:value-of select="@type"/>
       </xsl:attribute>
       <xsl:copy-of select="." />
    </xsl:element>
</xsl:template>
```

L'idée retenue consiste à placer le message dans un élément `div`, en lui appliquant un style qui dépendra du type. Pour cela, nous utilisons deux nouvelles instructions XSL :

- `xsl:element` pour créer un nouvel élément dans le document de sortie ;
- `xsl:attribute` pour lui associer des attributs.

On notera par ailleurs que la rigueur du processus nous assure la conformité des documents générés aux standards du W3C.

Enfin nous utilisons également une instruction importante : `xsl:value-of` dont l'objectif est de recopier des éléments du document d'origine vers le document de sortie. Le fonctionnement de cette instruction est simple, une donnée est désignée avec l'aide d'une expression XPath puis recopiée en lieu et place de l'instruction elle-même.

Ces deux premières règles vont nous permettre de suivre le déroulement d'une transformation XSL de A à Z. Voyons tout d'abord le résultat avec Cooktop :

Figure 8–3
Première application d'une feuille de style XSL

SYNTAXE XPath

Nous découvrons ici deux nouvelles syntaxes des expressions XPath, « . » qui désigne assez naturellement le nœud courant, et « @ » qui va permettre de spécifier un attribut par son nom.
Dans une expression XPath, on pourra ainsi réaliser des tests sur la valeur des attributs, par exemple :

```
info[@type='erreur']
```

Ce test recouvre les éléments `info` dont l'attribut type est `erreur` et constitue un moyen d'isoler les messages d'erreur des messages d'information simples.

Ce résultat est satisfaisant à première vue. Le squelette de la page est bien restitué, y compris la prise en compte de la feuille de style CSS. Le message d'information est lui aussi présent.

Cependant, la transformation semble avoir malgré tout opéré sur les éléments du nœud formulaire. Or, nous n'avons défini que deux règles, l'une supposée s'appliquer sur la racine, l'autre, sur les éléments de type `info`.

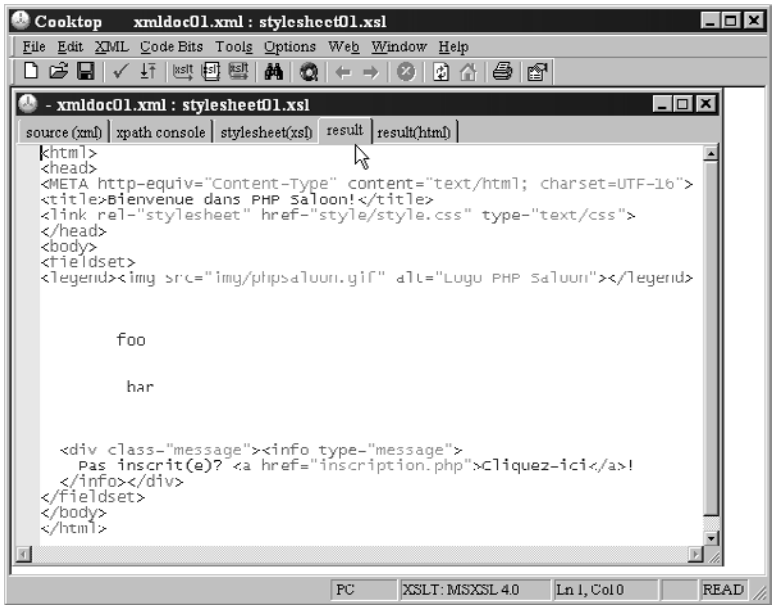


Figure 8–4
Code HTML produit par la transformation XSL

Si l'on regarde le code HTML produit, on constate que l'ensemble des champs texte du formulaire ont été recopiés. Ce résultat surprenant traduirait l'existence de règles cachées.

COMPRENDRE **Les règles de transformation par défaut**

Pour comprendre plus en détail les règles par défaut, le plus simple est encore d'en étudier le code. Très facile, il se contente d'utiliser les instructions de base que nous-mêmes avons déjà expérimentées. Tout au plus, vous découvrirez une palette de nouvelles extensions XPath qui vous sembleront limpides.

```
<xsl:template match="/|*">
     <xsl:apply-templates/>
</xsl:template>
<xsl:template match="processing-instruction()|comment()"/>
<xsl:template match="text()|attribute::*">
     <xsl:value-of select="."/>
</xsl:template>
```

Les lecteurs perspicaces qui resteraient dubitatifs quant à l'impact de ces règles, doivent noter qu'elles ne sont jamais prioritaires. Ce qui veut dire qu'une règle par défaut ne sera jamais préférée à une règle définie explicitement dans une feuille de style.

Ces règles sont de trois ordres et leur objectif est de simplifier les choses en évitant l'écriture de celles « qui vont de soi » :

- pour permettre l'exploration complète de l'arbre DOM par défaut ;
- pour recopier les éléments texte du document d'origine ;
- pour éliminer les éventuelles *Processing Instructions*.

Dans notre cas, que s'est-il passé ? Les règles par défaut ont provoqué le parcours complet de l'arbre DOM du document XML d'origine, y compris les éléments de la branche formulaire. Lors de ce parcours, ces mêmes règles ont recopié des textes contenus dans cette branche, et dans le cadre de notre fichier XML, le pseudo et le mot de passe. Ceux-ci se sont donc retrouvés au milieu de notre mise en page.

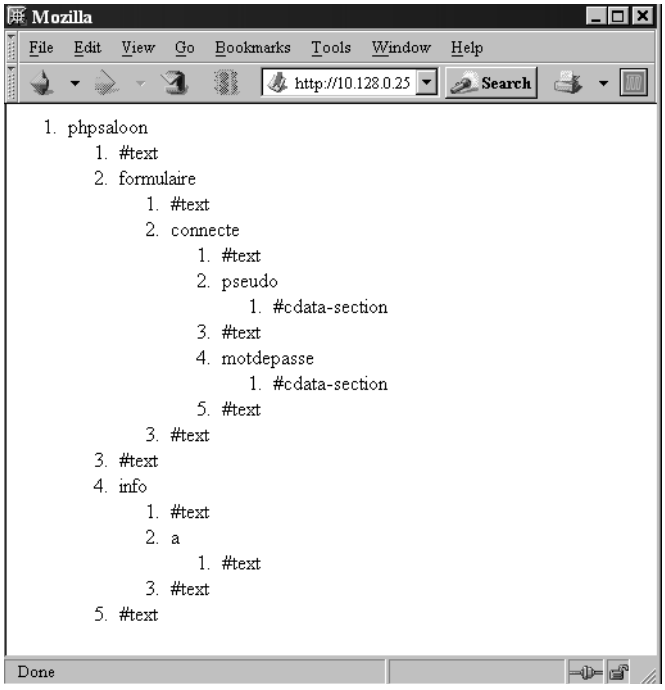


Figure 8–5
Arbre DOM complet du document « identification.xml »

Un dernier point reste à éclaircir. Pourquoi tant d'espaces et de retours à la ligne ? La réponse tient à l'analyse du document XML, ainsi que le rappelle la figure 8-5. Nous avons vu au chapitre précédent que l'analyseur XML intègre dans l'arbre DOM tout ce qu'il rencontre dans le document XML, y compris les espaces et les sauts de ligne entre éléments : autant de nœuds texte que nos règles par défaut vont recopier lorsqu'elles entrent en action.

Nous pouvons donc considérer que l'objectif est atteint, les artéfacts liés aux règles par défaut seront éliminés lorsque les règles prévues pour la gestion de l'élément formulaire seront intégrées à l'ensemble.

La feuille de style complète et son interprétation

Ces règles manquantes sont aux nombre de trois :

- une pour le formulaire en lui-même ;
- une pour le champ texte associé au pseudo ;
- une pour le champ mot de passe.

Considérons le cas du formulaire. Notre règle va devoir :

1. créer le squelette du formulaire avec l'élément `form` et un bouton de validation ;
2. relancer le parcours de l'arbre DOM au milieu de ce squelette pour pouvoir traiter les champs `pseudo` et `mot de passe`.

Une règle possible pour aboutir au résultat est la suivante :

```
<xsl:template match="formulaire">
   <form enctype="multipart/form-data" method="post"
         class="phpsaloon">
      <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
         <xsl:apply-templates />
      <div align="center">
         <input class="phpsaloon" title="Valider le formulaire"
           alt="Valider le formulaire" type="image" src="img/go.png" />
      </div>
   </form>
</xsl:template>
```

Seule précaution dans cette règle élémentaire, applicable à tous les éléments `formulaire` : un champ caché indispensable si le formulaire doit télécharger le contenu d'un fichier (c'est par exemple le cas pour la photo du connecté).

Concernant le champ `texte`, voici une règle applicable, dont nous allons détailler les éléments nouveaux :

```
<xsl:template match="formulaire//*[@type = 'texte']"> (1)
   <div> (2)
      <xsl:value-of select="@question" />
   </div>
   <xsl:element name="input"> (3)
         <xsl:attribute name="name">
              <xsl:value-of select="@name"/>
         </xsl:attribute>
         <xsl:attribute name="class">
            <xsl:choose> (4)
               <xsl:when test="@class">
                  <xsl:value-of select="@class"/>
               </xsl:when>
               <xsl:otherwise>champ</xsl:otherwise>
            </xsl:choose>
         </xsl:attribute>
         <xsl:attribute name="type">texte</xsl:attribute>
```

SYNTAXE XPath

Le symbole « / » permettait de matérialiser une relation de filiation, par exemple le `formulaire/pseudo` qui réduit le périmètre de recherche aux seuls éléments `pseudo` fils d'un élément `formulaire`.
« // » matérialise une relation plus large de descendance. Ainsi, l'expression `formulaire//pseudo` désigne l'ensemble des éléments `pseudo` situés au sein d'un élément `formulaire`, et ce, peu importe le niveau d'imbrication. L'expression `formulaire/a/b/c/pseudo` est donc compatible avec notre première expression.

```
        <xsl:attribute name="value">
          <xsl:value-of select="."/>
        </xsl:attribute>
    </xsl:element>
</xsl:template>
```

Premier point, l'expression XPath (1), qui est de loin la plus complexe rencontrée jusqu'à présent. Pour l'interpréter, le moyen le plus simple est de la lire de droite à gauche. Cette expression désigne l'ensemble des éléments (une expression XPath désigne toujours un ensemble de nœuds) disposant d'un attribut type égal à `'texte'`, dont le nom nous importe peu (d'où le symbole « * »), et descendant d'un formulaire.

L'intérêt de cette formule est de nous laisser une totale liberté en matière d'organisation du document XML. Peu importe la structuration et le nombre d'éléments imbriqués, nous nous intéressons uniquement aux champs textes.

Deuxième point, le contenu de notre règle. Nous organisons le document HTML final en deux zones :

1. le texte de la question (placé dans un élément `div`) (2) ;
2. le champ texte lui-même (3).

Troisième et dernier point, pour le champ texte, que nous créons de toutes pièces, l'application d'un style CSS par défaut est prévue, dans l'hypothèse où aucune classe CSS ne serait spécifiée. Cette petite amélioration permettra de laisser la place au rendu spécifique de certains champs tout en bénéficiant pour les autres, d'un style par défaut.

Nous utilisons dans cette optique une nouvelle instruction XSL : `xsl:choose` (4) qui correspond plus ou moins au `switch` bien connu de PHP, C ou Java. Comme souvent, avec XSL, la syntaxe n'est pas des plus concises, et il est possible de spécifier autant d'alternatives souhaitées avec `xsl:when`. L'option sélectionnée est la première qui valide une des expressions XPath précisée dans l'attribut `test`. Dans notre cas, nous entendons vérifier la présence d'un attribut `class`.

Enfin, comme on peut s'y attendre, `xsl:otherwise` permet de préciser une action par défaut.

À ce stade, nous pouvons tout à fait adapter notre règle et l'utiliser indifféremment pour le mot de passe et le pseudo. Dans ce cas, il suffirait de modifier l'expression XPath, et notamment la condition portant sur l'attribut `type` (on parle de prédicat) comme suit :

```
formulaire//*[@type = 'texte' or @type = 'secretstring']
```

L'inconvénient de cette solution économique est de laisser en clair le mot de passe lors de la frappe. Celui-ci est traditionnellement masqué, les caractères sont alors remplacés par des « * ».

Nous allons donc nous contenter d'adapter notre règle de manière très simple pour en obtenir une deuxième, dédiée à la transformation des mots de passe :

```
<xsl:template match="formulaire//*[@type = 'secretstring']">
   <div>
      <xsl:value-of select="@question" />
   </div>
   <xsl:element name="input">
      <xsl:attribute name="name">
         <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:attribute name="class">
         <xsl:choose>
            <xsl:when test="@class">
               <xsl:value-of select="@class"/>
            </xsl:when>
            <xsl:otherwise>champ</xsl:otherwise>
         </xsl:choose>
      </xsl:attribute>
      <xsl:attribute name="type">password</xsl:attribute>
      <xsl:attribute name="value">
         <xsl:value-of select="."/>
      </xsl:attribute>
   </xsl:element>
</xsl:template>
```

Désormais, toutes les règles indispensables sont définies. Il est donc possible de les regrouper sous forme d'un seul fichier ou de manière plus judicieuse, afin de conserver une approche modulaire en ayant recours, pour notre feuille de style globale, à des inclusions :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="iso-8859-1" />
   <xsl:include href="page/standard.xsl"/>
   <xsl:include href="form/standard.xsl"/>
   <xsl:include href="form/string.xsl"/>
   <xsl:include href="form/secretstring.xsl"/>

   <xsl:include href="form/info.xsl"/>
</xsl:stylesheet>
```

Squelette de la page.

Squelette du formulaire.

Règle de transformation du champ texte.

Règle de transformation du champ mot de passe.

Règle de transformation du message d'information.

À RETENIR Les URL utilisées dans les includes sont compatibles avec PHP

L'instruction `xsl:include` requiert une URL : on peut naturellement spécifier des chemins locaux, ou des URL traditionnelles utilisant le protocole HTTP. Cependant, l'intégration parfaite entre libXML (qui est le fondement de l'extension XML DOM de PHP) et PHP met également à disposition tous les protocoles et filtres disponibles dans PHP, comme l'accès aux fichiers compressés.

Le résultat obtenu (avec les directives CSS ad hoc) est celui escompté : le formulaire est correctement présenté (figure 8-6) et notre contrôleur va traiter les informations qui en résultent, en affichant au besoin un message d'erreur (figure 8-7).

Figure 8–6
Transformation complète de la page d'indentification

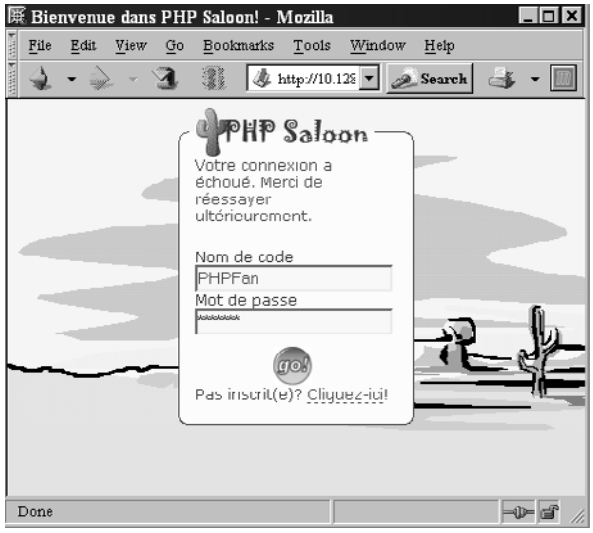


Figure 8–7
Transformation de la page d'identification après erreur

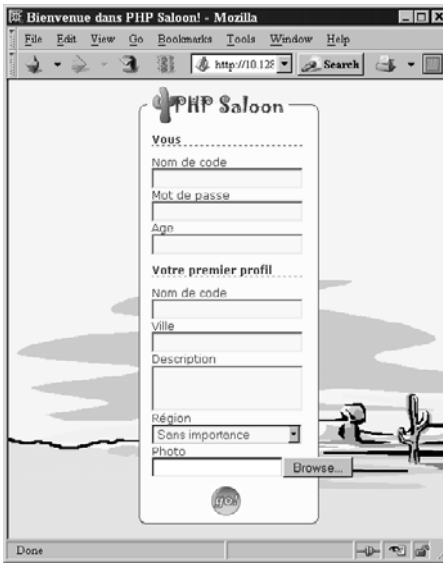


Figure 8–8
La page d'inscription transformée

PHP Saloon, vue d'ensemble de la version HTML

Avec la page d'identification et le formulaire d'inscription qui relève de la même logique, au point de reposer sur les mêmes règles XSLT (figure 8-8), l'ensemble des pages d'accès à PHP Saloon ont été prises en compte.

La page intérieure (ou principale) sera identique sur le principe. Mais pour simplifier les choses et ne pas avoir à produire une page monolithique, nous allons définir un squelette en HTML statique comportant des frames. Chacune sera affectée à un rôle dédié et l'application conservera une approche très modulaire (figure 8-9).

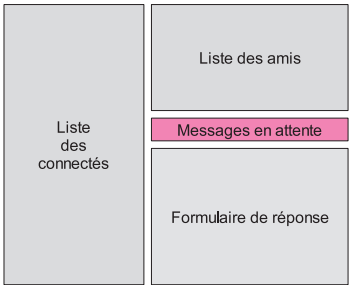


Figure 8–9
Organisation de la page principale de PHP Saloon

Le code HTML de la page reprend très précisément ce découpage fonctionnel, tout en modulant quelques éléments de décoration :

```
<?
   require_once  'inc/sessionvalide.php';
   $session = new sessionValide('identification.php');
?>
<html>
<head>
  <title>PHP Saloon, le chat où l'on cause</title>
  <link rel="stylesheet" href="style/style.css" type="text/css" />
</head>
<body>
<center>
<table cellspacing=0 cellpadding=4 border=0>
  <tr>
    <td>
      <img src="img/phpsaloon.png" alt="logo php saloon">
      <fieldset>
        <legend>Connectés</legend>
        <center>
          <iframe name"connectes" src="connectes.php" frameborder="0">
          </iframe>
        </center>
      </fieldset>
    </td>
    <td>
      <fieldset>
        <legend>Amis</legend>
        <iframe name="amis" src="amis.php" frameborder="0">
        </iframe>
      </fieldset>
      <fieldset>
        <legend>Messages</legend>
         <iframe name=messages src="messages.php" scrolling="no"
                frameborder="0">
         </iframe>
      </fieldset>
      <fieldset>
        <legend>Echanges</legend>
          <iframe name="echanges" src="echanges.php" frameborder="0">
          </iframe>
      </fieldset>
    </td>
   </tr>
  </table>
</center>
/body>
</html>
```

L'accès au site suppose qu'une session valide est en cours. Sinon, il faut renvoyer l'utilisateur vers la page d'identification.

Premier module : la liste des connectés (colonne de gauche).

Deuxième module : la liste des amis, en haut à droite.

Troisième module : les messages en attente, à droite au milieu.

Quatrième et dernier module : la gestion des échanges en bas à droite.

Cette organisation modulaire peut, en outre, factoriser un certain nombre de règles XSLT utilisées. C'est le cas pour les deux listes affichées : connectés et

amis. Dans ces deux cas de figure, ce sont des connectés qui sont affichés, amis ou pas. Du point de vue XML ou de l'affichage, il n'existe aucune différence.

```
<?xml version="1.0"?>
<phpsaloon>
   <connectes>
      <connecte uid="1">
         <pseudo>foo</pseudo>
         <age>32</age>
         <ville>Paris</ville>
         <cv>fan de php5</cv>
      </connecte>
      <connecte uid="45">
         <pseudo>bar</pseudo>
         <age>25</age>
         <ville>Rennes</ville>
         <cv>fan de Perl</cv>
         <photo>GO400GK40.png</photo>
      </connecte>
   </connectes>
</phpsaloon>
```

Le code XSL mis en œuvre sera identique, organisé en deux règles :

- une pour construire le squelette de la page ;
- une pour produire le code HTML associé à chaque connecté.

La première règle est élémentaire et se rapproche naturellement de celle définie pour la racine de la page d'identification :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
   <xsl:output method="html" encoding="iso-8859-1" />
   <xsl:template match="/">
      <html>
      <head>
        <title>Liste des connectés</title>
        <link rel="stylesheet" href="style2.css" type="text/css" />
      </head>
      <body>
         <xsl:apply-templates />
      </body>
      </html>
   </xsl:template>
   <xsl:include href="connecte.xsl" />
</xsl:stylesheet>
```

À RETENIR xsl:output

L'instruction `xsl-output` a pour objectif de préciser au processeur XSL les préférences relatives au document qui va être produit de manière à l'adapter le plus possible.
`Xsl-output` dispose de plusieurs attributs qui vont fixer le type de document produit (html, xml, texte), les besoins en indentation ou encore, l'encodage des caractères.

Il est important de constater que si les règles par défaut déjà évoquées précédemment peuvent parfois paraître bien encombrantes, elles nous assurent ici du parcours totalement automatique de la liste des connectés. C'est la raison pour laquelle il suffit d'intégrer (avec `xsl:include`) la règle de transformation type pour un élément `connecte`.

Cette règle reste purement conventionnelle, en dépit de son apparence plus ou moins volumineuse :

```
<xsl:template match="connecte">
<xsl:element name="table">
     <xsl:attribute name="onclick">
       window.parent.echanges.location='echanges.php?uid=
              ➥<xsl:value-of select="@uid"/>'
     </xsl:attribute>
     <tr>
       <td>
         <xsl:element name="img">
           <xsl:attribute name="src">
             <xsl:choose>
           <xsl:when test="normalize-space(./photo) = ''">
                  img/none.png
                </xsl:when>
                <xsl:otherwise>
                  <xsl:value-of select="./photo" />
                </xsl:otherwise>
             </xsl:choose>
           </xsl:attribute>
         </xsl:element>
       </td>
       <td>
         <xsl:value-of select="./pseudo" />
         <br />
         <xsl:value-of select="./age" />
         <br />
         <xsl:value-of select="./ville" />
         <br />
         <xsl:value-of select="./region" />
       </td>
     </tr>
     <tr>
       <td colspan="2">
         <xsl:value-of select="./cv" />
       </td>
     </tr>
   </xsl:element>
</xsl:template>
```

ASTUCE **Les espaces dans les transformations XSL et les documents XML**

Nous l'avons expérimenté à plusieurs reprises, l'analyse d'un document XML conserve les espaces blancs et les retours à la ligne. Cela vaut naturellement pour le document XML créé dynamiquement en PHP, mais aussi pour la feuille de style XSLT.

Les navigateurs étant parfois très susceptibles, il peut être nécessaire de faire disparaître ces espaces. Une solution, brutale, est de les éliminer physiquement des fichiers, mais dans ce cas, c'est la lisibilité qui devient problématique.

Pour résoudre cet inconvénient, XSLT propose plusieurs solutions. Tout d'abord, la fonction `normalize-space()` dont le rôle consiste à supprimer les espaces au début et en fin de chaîne (dans d'autres langages, cette fonction s'appelle `trim()`) .

De son côté, l'instruction `xsl:strip-space` permet de demander à ce que les textes parasitent certains éléments. Pour la liste des connectés, nous pourrions écrire :

```
<xsl:strip-space elements="connectes" />
```

Il faut noter qu'à l'inverse, il existe `xsl:preserve-space` (ne serait-ce que pour contrecarrer ponctuellement `xsl:strip-space`).

La figure 8-10 représente le résultat final, après transformation, de nos deux listes inclues.

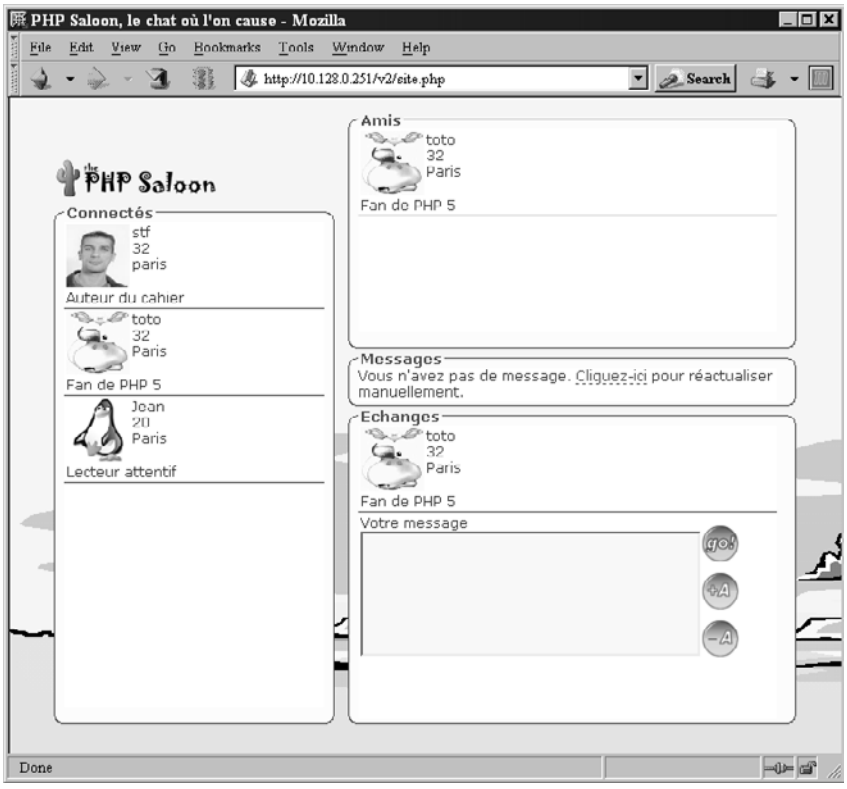


Figure 8–10
PHP Saloon complètement opérationnel

Dépasser les limites d'XSLT avec libXSL

Dans PHP Saloon, nous nous contentons des possibilités offertes par XSLT. Cependant, le nombre de fonctions disponibles (comme `normalize-space()`) est assez limité et peut manquer d'ampleur pour l'utilisateur habitué à PHP. Par chance, la bibliothèque libXSLT, usitée dans PHP 5 pour réaliser les transformations, prend en charge une initiative de la communauté : EXSLT. Ce projet définit un ensemble de fonctions, organisées par thématique, qui vont étendre le jeu de base disponible dans XSLT. Chaque processeur est libre d'implanter ou non ces fonctions qui ne font pas partie intégrante de la recommandation officielle. Dans PHP, la classe `xsltprocesseur` propose la méthode `hasEXSLTSupport()` pour déterminer si EXSLT est activé et disponible. Une autre possibilité offerte cette fois-ci par l'extension PHP elle-même est d'appeler des fonctions PHP directement depuis les feuilles de style XSLT, de la même manière que pour SQLite. Pour ce faire, la classe `xsltprocesseur` dispose d'une méthode `registerPHPfunctions()` qui permet d'activer la prise en charge au sein des feuilles de style XSL.

Le site officiel du projet EXSLT :
- http://www.exslt.org

Cette activation effectuée, il est possible de créer des templates de ce type :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:php="http://www.php.net/xsl">
<!-- ... -->
<xsl:template match="horodatage">
     <xsl:value-of select="php:function('date', '%D-%M-%Y')" />
</xsl:template>
<!-- ... -->
</xsl:stylesheet>
```

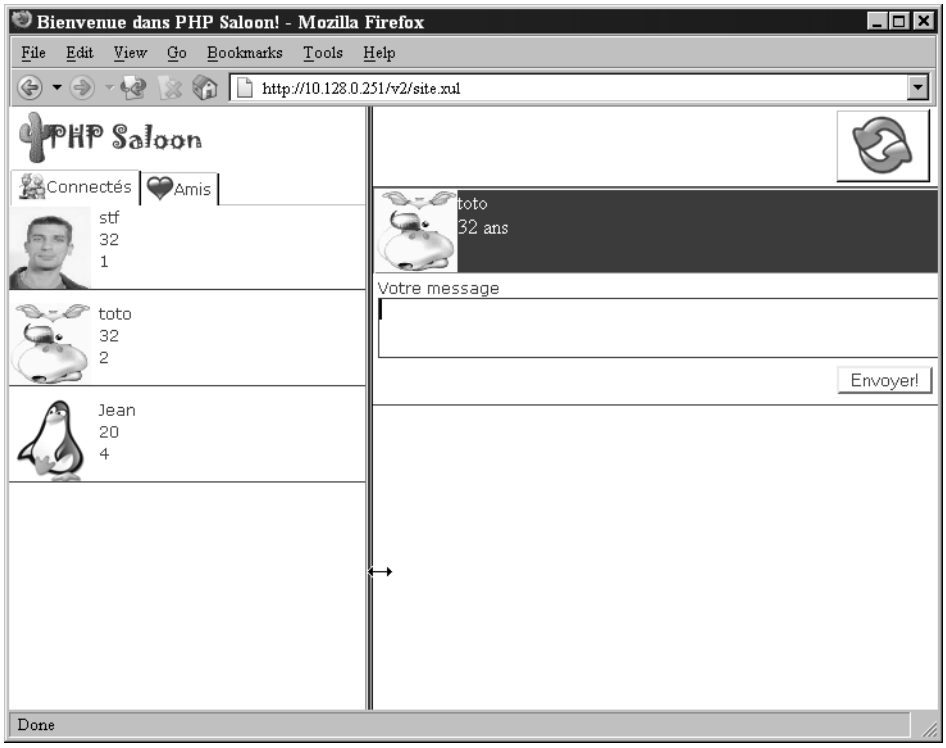
Attention de ne pas oublier le namespace associé à PHP pour cette extension.

Pratique, ce type de raccourcis vous vaudra, sans l'ombre d'un doute, la foudre des puristes XSL. Dans PHP Saloon, il serait possible d'en faire usage pour proposer une mise en page agréable et un horodatage plus avenant de l'historique des conversations.

En résumé...

Dans le chapitre précédent, XML a pu paraître superflu d'autant que DOM semble, à bien des égards, infranchissable. Avec les transformations XSL, on s'aperçoit rapidement du potentiel apporté par ce choix. Nos feuilles de style proposent une première apparence, mais demain, ces mêmes transformations pourraient aussi permettre de partager de l'information avec d'autres applications pour former une plate-forme de service plus large, ou encore offrir au visiteur une accessibilité améliorée grâce à plusieurs profils d'interfaces, dont des adaptations pour les personnes souffrant d'un handicap.

chapitre 9


Bienvenue dans PHP Saloon! - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://10.128.0.251/v2/site.xul
PHP Saloon
Connectés
Amis
stf
32
1
toto
32
2
Jean
20
4
toto
32 ans
Votre message
Envoyer!
Done

Une version Mozilla/XUL facile avec XSL

Notre version HTML de PHP Saloon est tout à fait satisfaisante et nous tirons déjà profit de l'architecture MVC en termes d'organisation pour le code source. Cependant, avec XML et XSLT, nous irons bien au-delà de cette première étape en créant une version plus riche et mieux adaptée aux utilisateurs de la suite Mozilla ou de Mozilla FireFox.
Cette réalisation sera d'autant plus aisée qu'XSLT permet de ne rien toucher au cœur de PHP Saloon.

SOMMAIRE

MOTS-CLÉS

- RDF
- XUL
- XPCOM
- JavaScript

Mozilla : une plate-forme technologique étonnante

Vous avez certainement déjà eu l'occasion de tester Mozilla. Fondé à ses débuts sur le code de Netscape, ce projet a, depuis, pris son envol en améliorant la réactivité du logiciel et en posant les fondements d'une véritable plate-forme de développement autour des standards web.

Aujourd'hui, plus qu'un navigateur, Mozilla est un outil complet qui permet de définir rapidement des applications communicantes d'une très grande richesse, qu'elles soient locales ou distantes. Ainsi, c'est sur la base de la plate-forme Mozilla qu'a été réalisé MAB (Mozilla Amazon Browser), un outil qui permet de consulter le catalogue d'Amazon de manière très pratique.

▸ http://mab.mozdev.org/

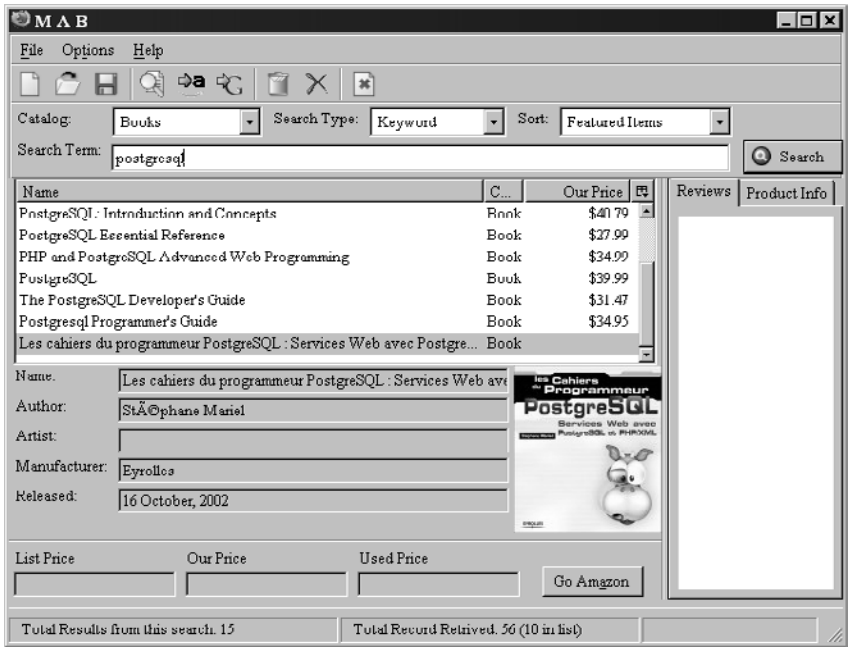


Figure 9–1 Mozilla Amazon Browser en action

DANS LA VRAIE VIE **Retour d'expérience**

Pour en savoir plus, découvrez l'expérience d'une entreprise qui construit ses prestations avec Mozilla, au travers dune interview exclusive.

▸ htttp://www.stephanemariel.com/article.php?id_article=42

Ce type d'outil est l'exemple même des possibilités offertes par Mozilla et les applications sont nombreuses ; certaines sociétés l'ont bien compris et développent aujourd'hui leurs applications plus rapidement grâce à l'aide de cet outil. Dans PHP Saloon, nous allons effleurer le sujet en proposant un échantillon des possibilités offertes.

Pour évaluer le potentiel disponible, il est important de comprendre ce que regroupe Mozilla comme briques essentielles. Car, au-delà du moteur de

rendu (le fameux Gecko) qui fédère un certain nombre d'éléments, c'est un véritable framework qui est défini par Mozilla, avec :

- une couche d'abstraction (NSPR, Netscape Portable Runtime) pour la création d'applications multi-plates-formes (typiquement Windows, Mac, Linux/Unix) ;
- un modèle de composant (XPCOM) pour servir, à la manière de COM pour les environnements Microsoft, de socle à l'ensemble des éléments de plus haut niveau. XPCOM dispose d'une API pour C/C++ mais aussi pour Python, et via XPCONNECT, s'interface avec JavaScript ;
- Necko, un composant qui implémente l'intégralité des protocoles réseau classiques, ceux typiquement utilisés dans les URL ;
- Gecko, le moteur de rendu qui supporte l'ensemble des standards du Web, XML, HTML, CSS, et qui propose une API conforme à DOM, accessible depuis JavaScript.
- un toolkit pour construire des interfaces complexes avec notamment XUL et XBL.

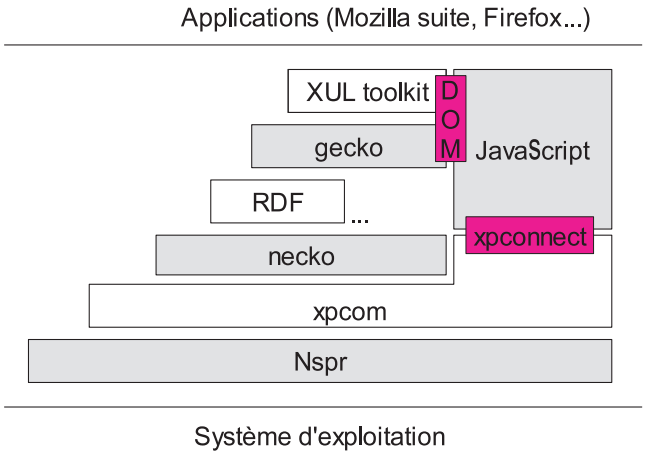


Figure 9–2 La plate-forme Mozilla

Il faut donc réellement cesser de voir en Mozilla un navigateur pataud. En premier lieu, le navigateur lui-même a beaucoup évolué et constitue aujourd'hui un concurrent redoutable pour Internet Explorer ; en second lieu, cette image d'Épinal ignore la réalité d'une plate-forme qui met en œuvre aujourd'hui (avec Xul notamment) des technologies que Microsoft lui-même n'intégrera que dans les prochaines versions de son système avec XAML et Avalon.

CULTURE Dites Zool

Comme nombre d'acronymes commençant par X, XUL se prononce « zool ».

RDF et les premiers tests de PHP Saloon avec XUL

Comme on le devine à l'énoncé des protocoles, Mozilla a pris le parti des standards de l'Internet et il n'est donc pas surprenant qu'XML soit au cœur de l'interface elle-même. Avec XUL (XML User Interface Language), nous allons donc décrire en XML l'interface de PHP Saloon, interface bien plus riche et conviviale que celle autorisée par HTML.

RÉFÉRENCE **XulPlanet pour tous renseignements sur XUL**

Dans cette version XUL de PHP Saloon, nous n'allons adapter que la page principale et n'explorer qu'une partie des éléments disponibles en XUL. Pour en savoir plus, le site Xul Planet dresse la liste exhaustive des éléments du langage et fournit par ailleurs un tutoriel très complet.
Enfin, il existe depuis peu un Wiki en français sur le sujet : XulFr.org

- http://www.xulplanet.com
- http://www.xulfr.org

REGARD DU DÉVELOPPEUR

La description de l'interface utilisateur avec XML : une méthode d'avenir

Mozilla est un précuseur en la matière, avec une solution fonctionnelle depuis quelques années, mais les poursuivants sont là. Ainsi, Macromedia (auteur de Flash) propose le MXML et sa technologie Flex.

De son côté, Microsoft entend faire de XAML (prononcez « zamel » à la manière de XUL) un moyen privilégié pour décrire les interfaces graphiques des applications. Là où Mozilla propose sa plate-forme et les standards web avec JavaScript et XUL, Microsoft répond avec son propre framework .Net et le couple XAML/C#. À l'avantage de ce dernier, la grande richesse des classes graphiques des environnements Windows, naturellement disponibles en XAML. À l'avantage de Mozilla, un environnement déjà en production, une approche plus classique pour les développeurs web (JavaScript et les CSS) qui ne devrait donc traumatiser personne et permettre de capitaliser l'expérience acquise.

Composants graphiques

Pour avoir une image plus précise de XUL, nous allons construire progressivement une première version statique de l'interface souhaitée pour la page principale de PHP Saloon ; mais avant cela, l'incontournable « Hello World ! ».

CONFIGURATION **XUL et votre serveur web**

XUL est un document XML particulier. À ce titre, votre serveur web doit le délivrer en affichant clairement dans les en-têtes de sa réponse le type de contenu `application/vnd.mozilla.xul+xml`. À défaut, Mozilla traitera le document selon le type de contenu prétendu (en général « text/html ») et rien ne sera affiché comme prévu.
Pour Apache, ce paramétrage est contrôlé via le fichier `mime.types` (généralement situé dans le répertoire de configuration à côté du fichier `httpd.conf`). Vérifiez la présence de la ligne :

```
application/vnd.mozilla.xul+xml  xul
```

qui associe l'extension `.xul` à ce type de contenu.

```
<?xml version="1.0" encoding="iso-8859-15" ?>
<window
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/
         there.is.only.xul"
  title="PHP Saloon, le salon où l'on cause"
  orient="horizontal">

  <button label="Hello World !"
          oncommand="alert('Eh ! Hello you guys !')"/>
</window>
```

Cet exemple élémentaire permet d'ores et déjà d'avoir une idée de la structure d'un document XUL. Pour Mozilla, une page ou une fenêtre applicative relèvent de la même logique. Dans PHP Saloon, nous n'allons évidemment pas nous limiter à construire des boutons. L'idée retenue repose sur une division en deux parties de la fenêtre. À droite, les échanges et l'envoi de messages, à gauche, les listes des connectés et des amis accessibles au moyen d'onglets.

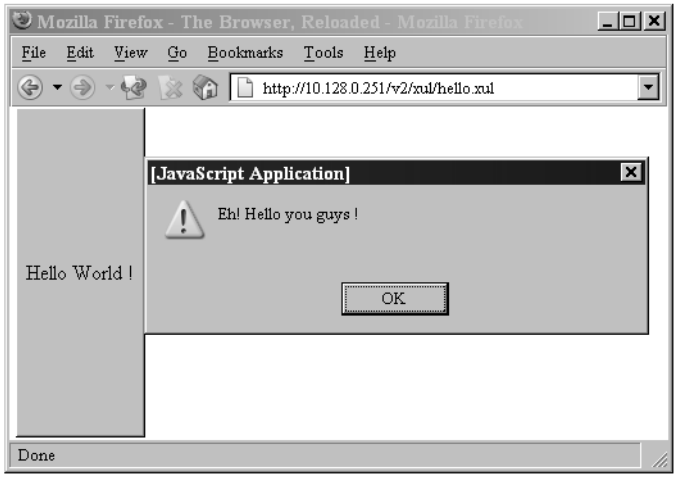


Figure 9–3 « Hello World ! » XUL en action

Pour cela, XUL met à notre disposition un certain nombre d'éléments :

- un `splitter`, qui comme son nom l'indique, va permettre de séparer l'objet placé à gauche et celui placé à droite ;
- une boîte à onglets : `tabbox` pour nos listes de connectés.

L'approche XML/XUL va tester immédiatement notre vision des choses. On peut, par exemple, tester le code suivant :

```
<?xml version="1.0" encoding="iso-8859-15" ?>
<window
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/
  ➥    there.is.only.xul"
  title="PHP Saloon, le salon où l'on cause"
  orient="horizontal">
  <tabbox flex="1"> 2
    <tabs> 4
      <tab label="Connectés" />
      <tab label="Amis" />
    </tabs>
    <tabpanels flex="1">3 5
      <tabpanel/>
      <tabpanel/>
    </tabpanels>
  </tabbox>
  <splitter collapse="before" resizebefore="closest"
resizeafter="closest" />
  <box flex="3" />1 2
</window>
```

Nous avons placé temporairement une boîte 1 à droite pour matérialiser l'espace en question. L'attribut `flex` 2 permet de préciser la proportion occupée par chaque élément. Ainsi, dans cet exemple, 1/3 sera réservé à la

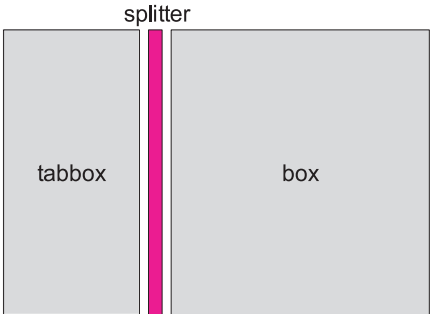


Figure 9–4 Organisation de la page principale de PHP Saloon XUL

partie gauche, 2/3 à la partie droite (pour le moment notre boîte vide). Le même attribut (3) nous assure que les pages associées aux onglets occuperont l'intégralité de la hauteur disponible (on aurait pu indiquer 100 % en lieu et place de 1).

L'élément `tabbox`, qui peut paraître complexe de prime abord, se décompose en deux parties :

1. Des éléments `tab` pour décrire les onglets par eux-mêmes et notamment leur texte, tous ces éléments étant regroupés au sein d'un élément `tabs` (remarquez le « s ») (4) ;
2. Des éléments `tabpanel` correspondant à la page associée à chaque onglet défini précédemment (dans notre cas, ces pages sont vides). Ces éléments sont regroupés au sein d'un élément `tabpanels` (remarquez le « s »). (5) Bien entendu, le nombre de `tab` doit correspondre au nombre de `tabpanel`...

La figure 9-5 présente ce premier résultat. Pour plus de lisibilité, il est possible d'utiliser directement les CSS (dans la pratique, il faudra réaliser un fichier externe pour tout regrouper). Sur la figure 9-6, chaque élément a été coloré en ajoutant un style dédié :

```
style="background: red;"
```

Figure 9–5 Premier test pour la page principale de PHP Saloon

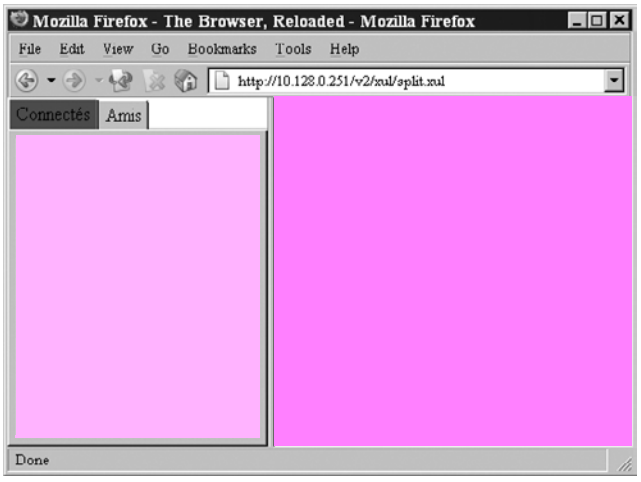


Figure 9–6 Premier test pour la page principale de PHP Saloon (version colorée)

Pour représenter les connectés dans chaque onglet, XUL dispose d'éléments pour placer l'information en grille, dans des boîtes, et pour afficher des images et des textes. Pour un connecté donné, il est possible d'appliquer une mise en page identique à celle schématisée en figure 9-7.

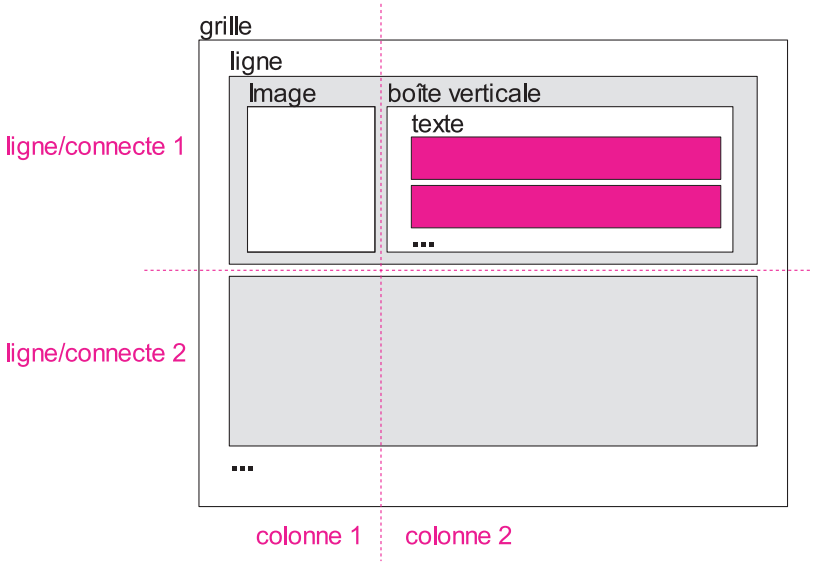


Figure 9–7 Représentation d'un connecté avec XUL

La traduction en XUL est élémentaire ; pour l'élément `grid`, il est possible de décrire le contenu soit en ligne, soit en colonne. Dans notre cas, nous procéderons en ligne (une ligne par connecté), l'élément `columns` n'est donc utilisé que pour préciser le nombre de colonnes effectivement requises.

```
<?xml version="1.0" encoding="iso-8859-15" ?>
<window
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/
there.is.only.xul"
  title="PHP Saloon, le salon où l'on cause"
  orient="horizontal">
  <tabbox flex="1">
    <tabs>
      <tab label="Connectés" style="-moz-appearance: none;
                                    background: green;"/>
      <tab label="Amis" />
    </tabs>
    <tabpanels flex="1">
      <tabpanel style="background: orange;">
   <grid>
     <columns>
       <column/>
       <column/>
     </columns>
     <rows>
       <!-- premier connecté -->
       <row>
         <image width="64" height="64" src="../jean.png" />
         <vbox>
           <label value="Jean" />
```

```
            <label value="Paris" />
          </vbox>
        </row>
        <!-- deuxième connecté -->
        <row>
          <image width="64" height="64" src="../vache_petite.gif" />
          <vbox>
            <label value="Stéphane" />
            <label value="Nice" />
          </vbox>
        </row>
      </rows>
    </grid>
        </tabpanel>
        <tabpanel/>
      </tabpanels>
    </tabbox>
    <splitter collapse="before" resizebefore="closest"
              resizeafter="closest"/>
    <box flex="3" style="background: red;"/>
</window>
```

Le résultat dans le navigateur est indiqué en figure 9-8.

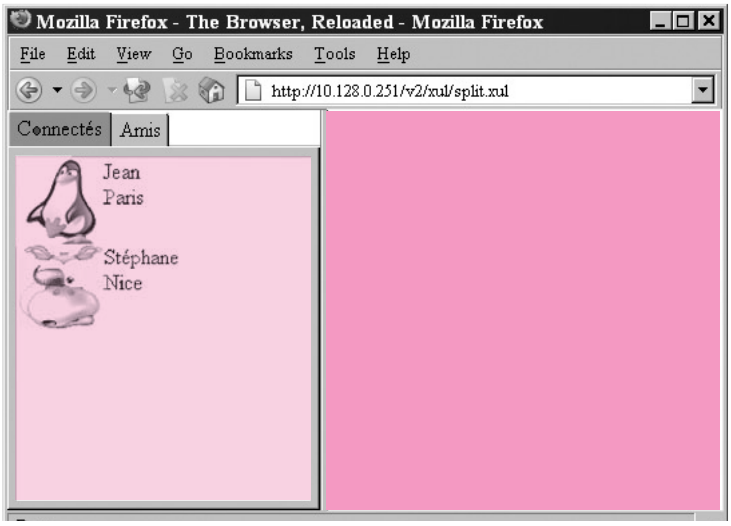


Figure 9–8 Représentation des listes de connectés (version statique)

Sources RDF

Sur la base de ce document totalement statique, nous pourrions naturellement utiliser PHP pour obtenir une génération complète conforme. Cependant, cette méthode brutale provoquerait la génération systématique de l'ensemble des informations dans un document volumineux.

Mozilla propose une mécanique beaucoup plus élégante pour construire un document dynamique à partir de notre squelette, en tirant profit des sources de données RDF (Resource Description Framework).

Comme son nom le laisse entendre, RDF est un langage dont l'objectif est de décrire des ressources (qu'il s'agisse d'objets, de personnes, de concepts) ou plus précisément des ressources et les relations de ces ressources entre elles.

C'est cette capacité d'organisation de l'information dont Mozilla tire profit en utilisant RDF à diverses reprises, que ce soit pour la gestion des signets, des préférences ou l'historique des sites visités.

Mieux, Mozilla aura recours aux ressources RDF pour construire dynamiquement certains éléments XUL. Pour PHP Saloon, nous n'allons donc pas regénérer l'intégralité de l'interface, mais plus simplement nous servir des transformations XSL pour exprimer les listes de connectés en sources d'information RDF. À charge ensuite à Mozilla de construire la version XUL des listes à partir de ces sources.

Avant d'examiner plus en détail ce processus, il nous faut cependant regarder d'un peu plus près ce fameux RDF. Comme on pouvait s'y attendre, il s'agit d'un langage qui sera représenté en XML (notons au passage que l'adaptation des transformations XSL devrait donc être relativement simple).

Cependant, pour bien comprendre la nature de RDF, il est préférable de ne pas commencer par la syntaxe (en XML) en se concentrant sur les concepts mis en œuvre. Avec RDF, nous allons représenter sous forme d'arbre les relations entre les ressources.

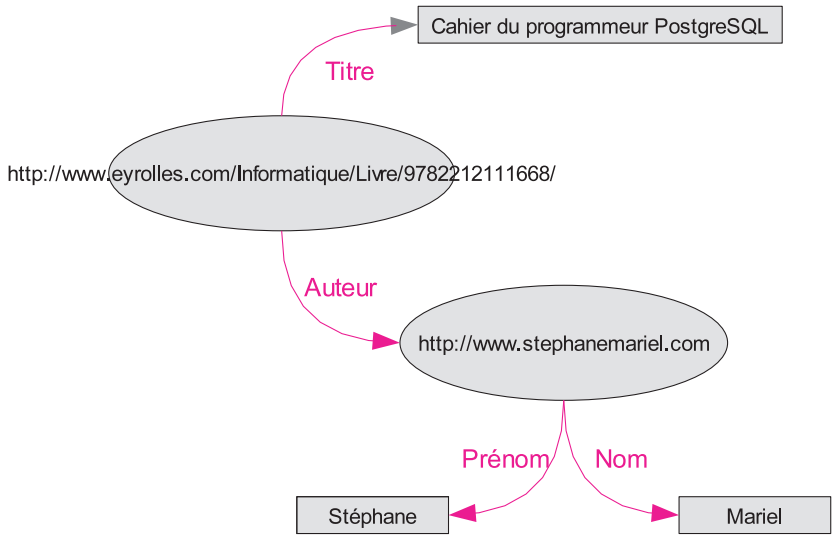


Figure 9–9 Relations entre ressources dans RDF

REGARD DU DÉVELOPPEUR **De l'intérêt de RDF**

RDF se positionne au sein de l'initiative « Web sémantique » du W3C. L'objectif est de répondre aux limites du fonctionnement actuel du Web et de l'Internet. En effet, nous produisons tous des documents, ces documents sont rendus disponibles via le Web, via des partages de fichiers, mais ils sont en réalité tous publiés en vrac. L'information est généralement disponible de manière non structurée et ce, sans qu'aucune trace du contexte ne soit sauvegardée.

Ainsi, pour rechercher sur le Web, la seule méthode disponible consiste à scanner tous les documents publiés et à retenir un sous-ensemble de mots présents dans ces documents. Il s'agit naturellement de la pire méthode envisageable. Impossible par exemple de demander les pages faisant référence à « Mariel, Stéphane », celui-là même justement qui est l'auteur du « Cahier du Programmeur PostgreSQL ». Il va falloir chercher, parmi les milliers de réponses obtenues, celles qui correspondent à la recherche.

Avec RDF, nous aurions pu référencer différentes ressources (par exemple, l'individu « Mariel, Stéphane » ou encore l'ouvrage « Cahier du Programmeur PostgreSQL ») et leurs éventuelles relations (« est l'auteur de »).

À RETENIR RDF & RSS

Mozilla n'est pas seul à exploiter RDF. Avec RSS (RDF Site Summary), de nombreux sites se servent de RDF pour diffuser leur contenu sur Internet. Il existe d'ailleurs des répertoires de flux RSS.
Les mécanismes décrits dans ce chapitre peuvent très largement être ré-utilisés au sein d'une application PHP pour présenter ces flux, les fusionner (on parle de syndication) et les rendre plus largement accessibles.

- http://web.resource.org/rss/1.0/

Pour désigner les arcs et les ressources, RDF impose le plus souvent de choisir un identifiant unique (dans le vocabulaire W3C, on parle d'URI, Uniform Resource Identifier, dont l'URL est une version connue). RDF se moque de la nature de l'URI, il nous demande simplement de nous assurer que chaque ressource en dispose de manière unique. Dans PHP Saloon, nous adopterons un format d'URI très classique. Ainsi, pour désigner un connecté dont le pseudo est « toto », nous utiliserons l'URL : http://www.phpsaloon.com/connectes/toto.

Techniquement, l'ensemble de ce graphe peut être modélisé avec des triplets de la forme :

ressource → relation → ressource

que ces ressources soient des objets composites, ou simplement des littéraux (chaîne de caractères, entier...).

La spécification RDF propose une syntaxe XML pour représenter ces triplets en les regroupant sous forme d'éléments `description`. Le réseau de la figure 9-9 peut alors s'écrire :

```
<rdf:rdf xmlns:rdf="http://www.w3.org/1999/02/22/rdf-syntax-ns"
     xmlns:eyrolles="http://www.eyrolles.com/rdf">
  <rdf:Description about="http://www.eyrolles.com/Informatique/Livre/
                        ➥ 9782212111668/">
     <eyrolles:titre>
        Cahier du Programmeur PostgreSQL
     </eyrolles:titre>
     <eyrolles:auteur>
        <rdf:Description about="http://www.stephanemariel.com">
           <eyrolles:nom>
              Mariel
           </eyrolles:nom>
           <eyrolles:prenom>
              Stéphane
           </eyrolles:prenom>
        </rdf:Description>
     </eyrolles:auteur>
  </rdf:Description>
</rdf:rdf>
```

Dans PHP Saloon, nous allons donc réécrire les transformations XSL utilisées pour les listes de connectés, de façon à obtenir une description RDF de ces informations. Dans notre cas, il va s'agir de produire une séquence de triplets chacun désignant des connectés.

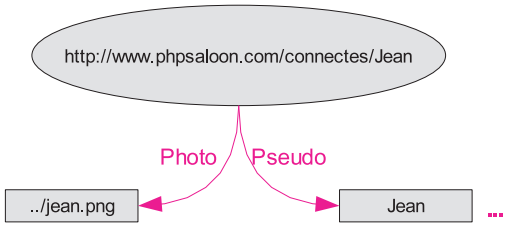


Figure 9–10
Représentation RDF d'un connecté

En XML, nous obtenons pour un connecté :

```
<rdf:Description about="http://www.phpsaloon.com/connectes/toto">
   <connectes:photo>../img/none.png</connectes:photo>
   <connectes:pseudo>toto</connectes:pseudo>
</rdf:Description>
```

À partir de là, nous pourrions définir une ressource « connectés » (au pluriel) en relation avec chaque connecté. Une telle méthode manquerait cruellement de lisibilité. RDF propose pour améliorer les choses l'utilisation de conteneurs pour regrouper différentes descriptions (et donc différents graphes) sous forme d'ensembles ou de listes.

Pour PHP Saloon, la liste des connectés pourra donc être représentée comme suit :

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:connectes="http://www.phpsaloon.com/#">
  <RDF:Seq RDF:about="http://www.phpsaloon.com/connectes">
    <RDF:li>
      <RDF:Description RDF:about="http://www.phpsaloon.com/connectes/
                                  ➥ stf">
          <connectes:photo>../img/none.png</connectes:photo>
          <connectes:pseudo>Stf</connectes:pseudo>
      </RDF:Description>
    </RDF:li>
    <RDF:li>
      <RDF:Description RDF:about="http://www.phpsaloon.com/connectes/
                                  ➥ toto">
          <connectes:photo>../img/none.png</connectes:photo>
          <connectes:pseudo>toto</connectes:pseudo>
      </RDF:Description>
    </RDF:li>
  </RDF:Seq>
</RDF:RDF>
```

C'est ce type de document que Mozilla va interpréter pour nous et utiliser pour peupler chacune des lignes de notre grille.

Template et génération dynamique de l'interface utilisateur

Ce mécanisme est désigné sous le terme de template, qui, comme son nom l'indique, repose sur l'utilisation de patrons reproduits pour chaque élément des séquences RDF.

Reprenons de manière expurgée la structure de notre document XML :

```
<rows>
   <!-- premier connecté -->
      <row>
         <image width="64" height="64" src="../jean.png" />
         <vbox>
            <label value="Jean" />
            <label value="Paris" />
         </vbox>
      </row>
      <!-- connectés suivants -->
</rows>
```

Nous devons dans un premier temps indiquer :

- la source des données RDF qui sera utilisée ;
- la ressource qui décrit les éléments à parcourir, dans notre cas, la séquence des connectés.

Ceci s'effectue au niveau de l'élément `rows`.

```
<rowsdatasources="connectes.rdf" ref="http://www.phpsaloon.com/
                ➥ connectes" id="connectes">
   <!-- ... -->
</rows>
```

Désormais, Mozilla sait qu'il va devoir construire le contenu de l'élément `rows` à partir de notre fichier RDF. Il reste à décrire le patron qui sera reproduit. Le code suivant illustre la syntaxe retenue par Mozilla :

```
<template> 1
   <rule> 2
      <rowuri="rdf:*"> 3
         <image width="64" height="64"
                src="rdf:http://www.phpsaloon.com/#photo"/>
         <vbox>
            <label value="rdf:http://www.phpsaloon.com/#pseudo" /> 4
            <label value="rdf:http://www.phpsaloon.com/#age" />
         </vbox>
      </row>
   </rule>
</template>
```

L'élément `template` 1 désigne le patron. Celui-ci peut être composé de plusieurs règles 2 conditionnées ou non. Dans notre cas, le modèle est très simple, une seule règle de reproduction (inconditionnelle) est mise en œuvre.

Le contenu d'une règle est en tout point identique à du code XUL classique. La différence est que ce code va être reproduit autant de fois que requis et certains paramètres seront remplacés.

L'attribut `uri` ❸ de l'élément `row` peut paraître quelque peu barbare. En réalité, l'étoile indique, comme pour les expressions régulières, que tous les éléments de la séquence sont acceptables (et vont donc donner lieu à la création d'un élément `row`).

La suite est évidente, les URI de la forme `rdf:XXX` ❹ vont être remplacés par l'élément correspondant de la description RDF. Là encore, la syntaxe peut sembler étrange mais, en réalité, il s'agit de la concaténation de l'espace de nommage `connectes`, utilisé dans notre fichier RDF (`http://www.phpsaloon.com/#`) et du nom de l'élément dans la description (`pseudo`, `âge`...).

Après avoir ajouté ce template au document XUL construit précédemment, nous obtenons le résultat présenté en figure 9-11. Comme attendu, les deux connectés de la source RDF ont été reproduits. Le rendu est identique à la version statique. Le moment venu, il suffira donc de remplacer le fichier `connectes.rdf` par `connectés.php` et le tour sera joué.

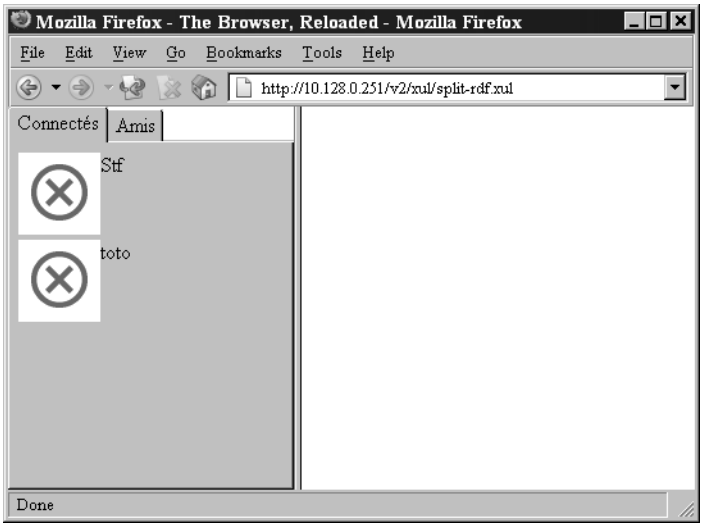


Figure 9–11 Création de l'interface par Mozilla à partir d'une source RDF

Rafraîchissement et sécurité

Avant d'aborder ce dernier point (qui consistera à adapter nos feuilles de style XSL), il reste un problème à prendre en compte : celui de la réactualisation de la source de données. Ce point n'a rien d'épineux, Mozilla disposant des interfaces requises, mais il faut au préalable tenir compte des contraintes de sécurité.

ASTUCE Modifier le niveau de sécurité de Mozilla (ou de Firefox)

Mozilla propose une fonctionnalité, certes merveilleuse, mais évidemment pleine de risques. L'URL `about:config` donne en effet accès à la totalité des paramètres de configuration. L'interface utilisateur permet de rechercher dans les préférences, par mots-clés, et les éléments personnalisés sont mis en gras pour davantage de lisibilité. L'élément qui nous intéresse est :

`signed.applets.codebase_principal_support`

Pour accéder aux privilèges nous concernant, il suffit de faire passer la valeur de false à true (cliquez à cet effet sur la ligne en question, avec le bouton droit de la souris).

En effet, depuis le début de ce chapitre, nous avons pu constater l'étendue des possibilités offertes par le framework Mozilla, RDF ajoutant encore à la richesse de celui-ci. On ne peut accéder à un tel niveau fonctionnel sans un minimum de précautions. Parmi celles-ci, l'interdiction faite aux documents XUL distants de manipuler directement les sources RDF, ce qui nous est pourtant nécessaire pour la réactualisation.

Là encore, Mozilla propose plusieurs réponses :

- la signature des applications distantes (dans notre cas, il s'agit de créer une archive de l'ensemble des documents et de la signer) ;
- l'installation de cette archive en tant qu'application locale (Mozilla propose un outil pour télécharger directement de telles extensions) ;
- l'abaissement du niveau de sécurité de Mozilla.

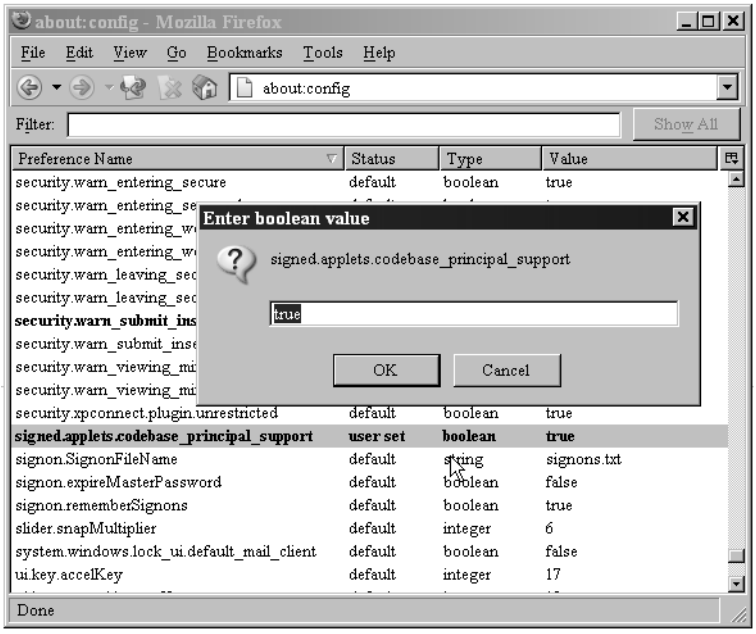


Figure 9–12 Modification de la configuration de Mozilla

Les deux premières solutions sont naturellement préférables ; cependant, pour notre exemple, nous allons adopter la troisième solution, facile pour quelques tests, et réversible.

Une fois la sécurité adaptée, il devient possible de définir avec JavaScript une fonction `refresh()` laquelle, déclenchée périodiquement, provoquera le rafraîchissement de l'interface et des sources de données.

En voici le code commenté :

```
function refresh() {
  try {
  netscape.security.PrivilegeManager.enablePrivilege(
                                          'UniversalXPConnect');
  } catch(e) {
     alert("Le niveau de sécurité en vigueur ne permet pas à\n"+
          "PHP Saloon de fonctionner.\n"+
          "Agissez ou vous serez déconnectés automatiquement !");
     return;
  }
    connectes = document.getElementById('connectes');

  sources   = connectes.database.GetDataSources();
  if (sources.hasMoreElements) {
     source = sources.getNext();

  }
  source = source.QueryInterface(Components.interfaces.
           ➥ nsIRDFRemoteDataSource)


  source.Refresh(false);

  connectes.builder.rebuild();

    setTimeout('refresh()', 15000);
}
```

- Demande le privilège d'accéder à la totalité des possibilités offertes par la plate-forme Mozilla.
- En cas d'échec, informe l'utilisateur, les exceptions JavaScript fonctionnant de manière analogue à celles disponibles en PHP.
- Et nous en restons là.
- Nous récupérons l'élément rows désigné par l'id `connectes`.
- Puis la liste des sources RDF...
- Sous la forme d'une sorte d'itérateur.
- Nous utilisons la première source uniquement (normalement la seule).
- À partir de l'objet générique, nous demandons l'interface spécifique aux sources RDF distantes (ce qui nous serait refusé si la source n'était pas une source RDF).
- Cette interface nous permet d'appeler la méthode `refresh()` de la source RDF.
- Cela fait, il suffit de demander à l'élément rows de se reconstruire.
- La fonction sera ré-appelée dans 15 secondes.

Adaptation des transformations XSL

Tout est désormais prêt pour la connexion à la partie Contrôleur de PHP Saloon. Pour ce faire, nous allons :

1. définir des feuilles de style adaptées (qui produiront le code RDF) ;
2. modifier la vue pour que les navigateurs soient détectés et les feuilles de style correspondantes, utilisées.

Nouvelles transformations

Ainsi, nous devons adapter la feuille de style `connectes.xsl` associée à la liste des connectés. Dans la version HTML, elle permettait de produire le squelette de la page HTML. Dans la mouture RDF, le principe reste identique et cette nouvelle version se contente de produire l'élément racine du document RDF et la base de la séquence.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" 1
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:connectes="http://www.phpsaloon.com/#">
  <xsl:output method="xml" indent="yes" encoding="iso-8859-1" /> 2
  <xsl:template match="/">
     <rdf:rdf xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:connectes="http://www.phpsaloon.com/#">
       <rdf:Seq rdf:about="http://www.phpsaloon.com/connectes">
         <xsl:apply-templates />
       </rdf:Seq>
     </rdf:rdf>
  </xsl:template>
  <xsl:include href="connecte.xsl" />
</xsl:stylesheet>
```

On notera la présence de l'intégralité des namespaces manipulés 1 ainsi que la modification de la nature du document de sortie qui passe d'HTML à XML 2.

L'adaptation de la règle de transformation prévue pour chaque connecté n'est guère plus complexe ; RDF ne se préoccupe que des données, ce qui simplifie les transformations.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:connectes="http://www.phpsaloon.com/#">
  <xsl:output method="xml" encoding="iso-8859-1" />
  <xsl:template match="connecte">
     <RDF:li>
       <RDF:Description RDF:about="http://www.phpsaloon.com/
                                   ➥connectes/{./pseudo}">
         <connectes:photo>
           <xsl:choose>
             <xsl:when test="normalize-space(./photo) = ''">
               img/none.png
             </xsl:when>
```

```
                <xsl:otherwise>
                  <xsl:value-of select="./photo" />
                </xsl:otherwise>
              </xsl:choose>
            </connectes:photo>
            <connectes:uid>
              <xsl:value-of select="@uid" />
            </connectes:uid>
            <connectes:pseudo>
              <xsl:value-of select="./pseudo" />
            </connectes:pseudo>
            <connectes:age>
              <xsl:value-of select="./age" />
            </connectes:age>

        </RDF:Description>
      </RDF:li>
    </xsl:template>
</xsl:stylesheet>
```

Amélioration de la vue

Après avoir créé les nouvelles transformations pour Mozilla/XUL, il reste encore une petite adaptation. En effet, pour le moment, les classes `vue` utilisées ne se préoccupent pas de la nature du navigateur dont se sert le visiteur. Dans ce cas, aucune chance que les transformations spécifiques ne soient appelées.

Nous allons donc définir deux nouvelles fonctions :

- `contenu_prefere()`,
- `redirect()`.

La première retourne le type de document préféré par le navigateur (XUL pour Mozilla, cHTML pour les terminaux i-mode, et HTML pour tous les autres). La classe `vue` peut alors être modifiée pour que soit chargée la feuille de style la mieux adaptée :

```
$xsl->load('style/' . contenu_prefere() . '/' . $xslfile);
```

La deuxième redirige vers la page principale du site en fonction du type de contenu (par exemple `site.xul` ou `site.html`). Cette fonction est, entre autres, indispensable après la phase d'identification.

Ces quelques modifications apportées, l'utilisation de PHP Saloon avec Mozilla ou Firefox peut commencer, avec une interface propre.

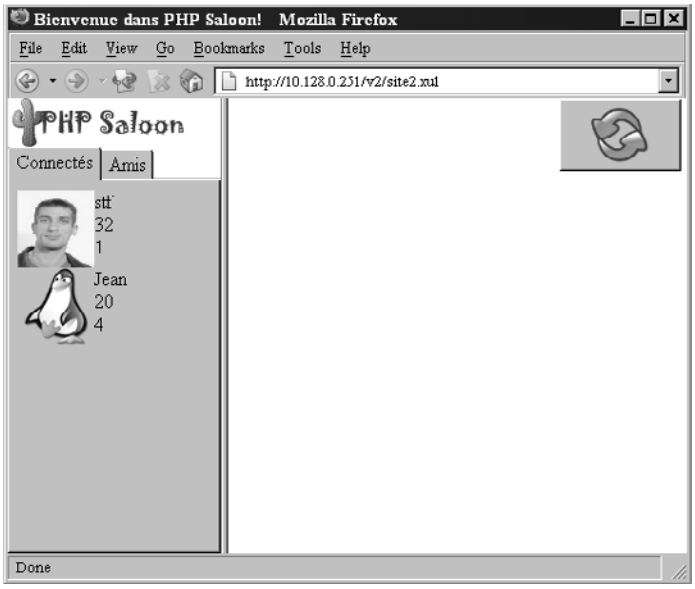


Figure 9–13
Version finale de l'interface XUL pour PHP Saloon

Finalisation de l'interface avec CSS

Évidemment, l'interface obtenue n'est pas particulièrement jolie. Une raison à cela : nous n'avons pour le moment appliqué aucun style. Comme pour tout le reste, Mozilla a adopté un standard bien connu du W3C : les feuilles de style CSS, nous restons donc en territoire connu.

Pour modifier l'apparence de chaque ligne, on pourra définir le style suivant :

```
row {
        border-bottom: 1px solid darkgreen;
        margin-bottom: 10px;
        color: darkgreen;
}
```

Ce procédé est disponible pour l'intégralité des éléments XUL, y compris la fenêtre principale :

```
window {
        background: white;
        font-family: Verdana, Helvetica, Arial, Sans-Serif;
        font-size: 12px;
        color: darkgreen;
}
```

Dans la pratique, on aura intérêt à associer des classes aux éléments XUL de manière à ne pas changer l'apparence globale d'un élément donné (par exemple `button`).

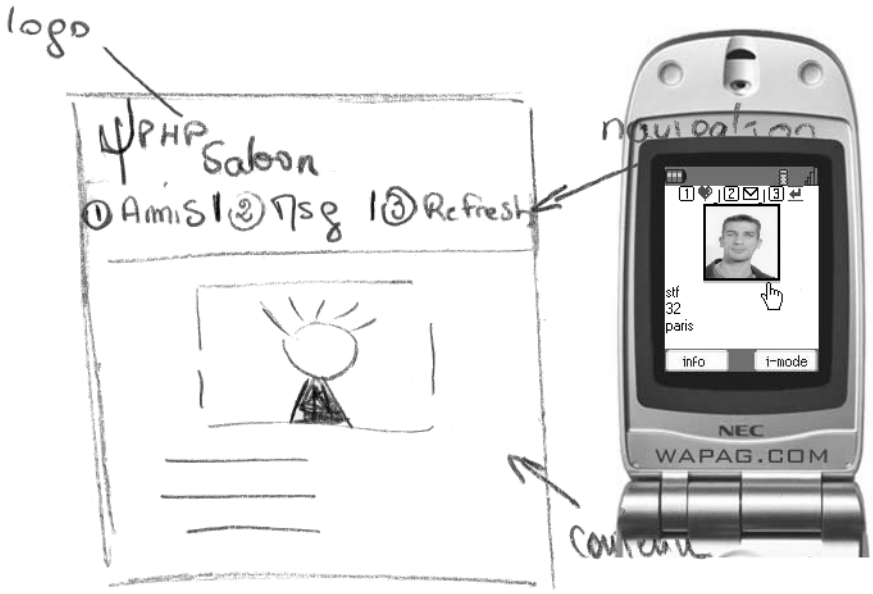
Après ces quelques ajouts, notre PHP Saloon XUL a déjà bien meilleure allure !

Figure 9–14
PHP Saloon XUL totalement opérationnel

En résumé...

Grâce à notre architecture MVC et à l'utilisation d'XML, nous venons de définir une version totalement revue de PHP Saloon. Ce potentiel est clairement le fait de PHP 5 qui stabilise les API XML/XSL et ouvre ainsi à la communauté PHP les portes de l'univers XML. Naturellement, ce qui vient d'être réalisé pour PHP Saloon est transposable à des applications plus complexes. En entreprise, on pourra ainsi assurer l'interopérabilité avec l'ensemble des éléments du système d'information et définir des interfaces diverses, riches avec un client lourd comme Mozilla sur le poste de travail interne, allégées pour les mobiles, ou HTML pour l'extranet. Tout cela avec un minimum d'effort et dans un cadre davantage propice à la qualité et à l'exploitation des développements.

chapitre 10


logo
PHP Saloon
1 Amis 2 Msg 3 Refresh
stf
32
paris
info
i-mode
NEC
WAPAG.COM

Version i-mode allégée

Avec la version XUL de PHP Saloon, nous avons procédé à une refonte en profondeur de l'apparence. Une modification aussi importante n'est pas toujours nécessaire. Ainsi, en proposant une version i-mode, il ne sera plus question de la richesse de l'interface mais de son adéquation à un terminal mobile. Il faudra donc simplifier à l'extrême et tirer profit des astuces ergonomiques de la plate-forme cible.

SOMMAIRE

MOTS-CLÉS

- cHTML
- Cookie
- session_id

Contraintes et propositions pour une utilisation mobile

Avec XUL et Mozilla, nous avons répondu à une exigence : utiliser au mieux le potentiel disponible sur la plate-forme de l'utilisateur de PHP Saloon. Utiliser l'espace, avec les onglets, pour une meilleure présentation des listes et des messages ; organiser l'interface pour intégrer les possibilités habituelles et reconnues des systèmes modernes et qui dépassent très largement celles du Web et du langage HTML.

Notre exigence avec cette version i-mode demeure inchangée. Il s'agit toujours d'occuper au mieux l'espace et de tirer profit de la plate-forme client, mais dans ce cas, les contraintes vont être diamétralement opposées.

Les contraintes physiques

Là où l'écran d'un ordinateur fixe aurait pu paraître trop grand pour PHP Saloon, l'écran d'un téléphone mobile va clairement poser problème. Les dimensions standards pour ce type de terminal font état de 120 pixels de largeur pour 160 pixels de hauteur. Il va sans dire que notre approche initiale à base de frame est incompatible avec ce type d'écrans.

Autre contrainte, celle d'un support restreint du langage HTML et de ses extensions. Il n'est plus question alors d'utiliser CSS ou JavaScript, ce qui entraîne les deux conséquences suivantes :

- Les transformations devront intégrer l'ensemble des options de mise en page (par chance fort limitées).
- Le rafraîchissement de l'information ne pourra être réalisé de manière automatique ; il faudra donc prévoir un moyen pour le provoquer manuellement.

Ce dernier point peut paraître totalement anodin mais il s'avère fort judicieux, compte tenu de la faible bande passante disponible, même si un des corollaires de cette contrainte est de devoir trouver la place pour un bouton supplémentaire.

Le support du langage HTML lui-même est réduit à un sous-ensemble (NTT Docomo, l'inventeur d'i-mode l'a nommé cHTML). Cependant, les balises clés utilisées dans PHP Saloon sont présentes, et compte tenu des faibles possibilités de mises en page, l'ensemble est très suffisant.

Cependant, là encore, il faudra intégrer une caractéristique limitative : l'absence de clavier et de souris. Pour pallier ce dernier point, cHTML définit une méthode permettant d'associer un lien à certaines touches du clavier.

ALTERNATIVES **Wap ou i-mode**

Pour PHP Saloon, nous avons fait le choix d'i-mode, choix guidé par la simplicité et par le dynamisme d'une plate-forme aujourd'hui portée par un ratio d'utilisation supérieur à celui disponible sur les téléphones Wap.
Simplicité avec un cHTML, dans la droite ligne d'HTML, même si la version 2.0 de WML (WAP) corrige les erreurs de jeunesse et simplicité toujours avec un support facilité des terminaux.
Mais ne nous leurrons pas. Que ce soit i-mode ou Wap, notre méthode de travail est adaptable aux deux. Pour créer une version Wap de PHP Saloon, il suffira d'adapter les feuilles de style XSLT comme pour i-mode, ni plus ni moins.

POUR PLUS D'INFORMATIONS

Le guide de développement i-mode

Pour obtenir le détail des éléments HTML pris en charge et les contraintes qui s'appliquent à la plate-forme i-mode, un guide est disponible à l'attention des développeurs.

▸ http://communaute.imode.fr

Dernière contrainte, le navigateur i-mode embarqué dans les terminaux ne gère qu'une sous-partie du protocole de transport HTTP. Il faudra donc nous passer des cookies et gérer les sessions autrement.

Tableau 10–1

Contrainte	Conséquence
Faible taille de l'écran	Frame impossible : prévoir une navigation entre les trois parties de l'application.
Pas de JavaScript	Prévoir un bouton ad hoc pour le rafraîchissement des listes, par exemple.
Pas de clavier ni de souris	Utiliser les raccourcis proposés par cHTML pour activer les liens plus rapidement.
Pas de cookies	Gérer les sessions sur l'URL.

Éléments d'adaptation pour PHP Saloon

Malgré l'importance apparente de ces contraintes, peu de modifications de fond seront nécessaires. En ce qui concerne les pages d'inscription et de connexion, notre mise en page actuelle est acceptable en l'état, pour peu que l'on se contente de la partie centrale (voir figure 10-1). Quant au cas particulier de l'inscription, il faudra toutefois se passer de photographie, impossible à télécharger simplement, depuis un terminal mobile.

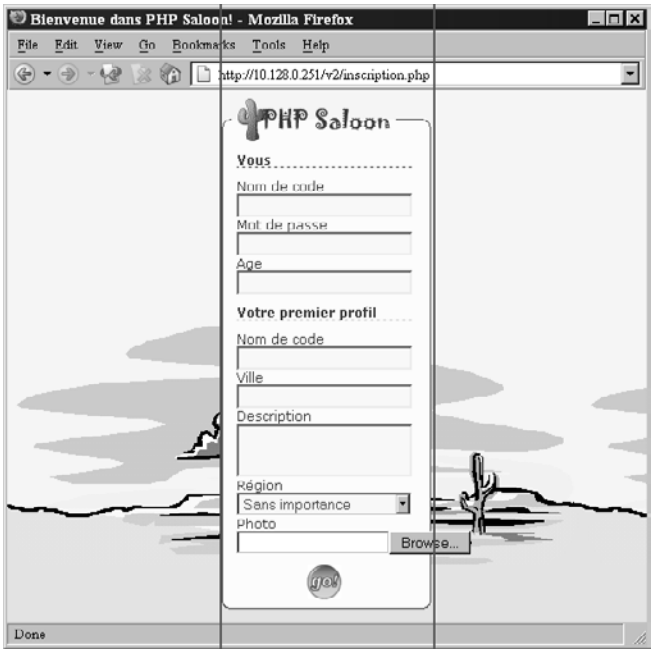


Figure 10–1 Adaptation des pages d'inscription et de connexion

Bien entendu, l'utilisation des frames sur la page principale impose de revoir certaines choses mais, là encore, il n'est pas question de tout bouleverser de fond en comble.

Nous avons trois scripts importants :

- liste des connectés ;
- liste des amis ;
- échange et envoi de message.

Dans la mesure où ces trois éléments ne sont plus affichables simultanément, nous allons ajouter à chaque transformation XSLT la production d'un menu de navigation. Ce menu (voir les figures 10-2 et 10-3) permettra de passer d'une page à l'autre.

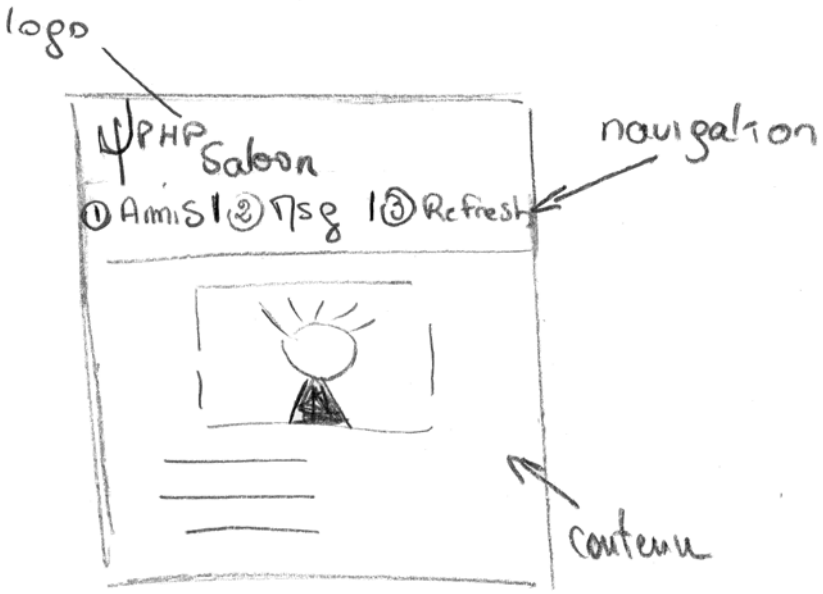


Figure 10–2 Réorganisation de l'interface et des listes de connectés

Ce même menu intégrera le lien de rafraîchissement.

Pour gérer les raccourcis sur ces liens, nous nous servirons de la méthode proposée dans le guide de développement i-mode en associant à chaque lien une `accesskey` (cet attribut est d'ailleurs tout à fait exploitable en HTML 4). Le plus simple est alors d'associer les touches 1, 2 et 3 à chaque lien. Dans cette hypothèse, la touche 1 permet alors de basculer en permanence de la liste des amis à la liste complète des connectés. Afin de clarifier davantage les choses, nous allons suivre les conventions proposées et ajouter dans les liens les Emoji (pictogrammes i-mode) correspondants.

CULTURE Les Emoji

Les pictogrammes i-mode ne sont rien d'autre que de petites images toutes simples. Elles ont cependant un avantage. Disponibles directement dans le terminal mobile, elles ne requièrent ni transfert ni ressources pour être affichées et peuvent constituer un moyen simple d'alléger les pages, comme c'est le cas, dans PHP Saloon, pour la navigation.
La liste de ces pictogrammes est disponible en téléchargement sur le site officiel d'i-mode en France.

▸ http://communaute.imode.fr

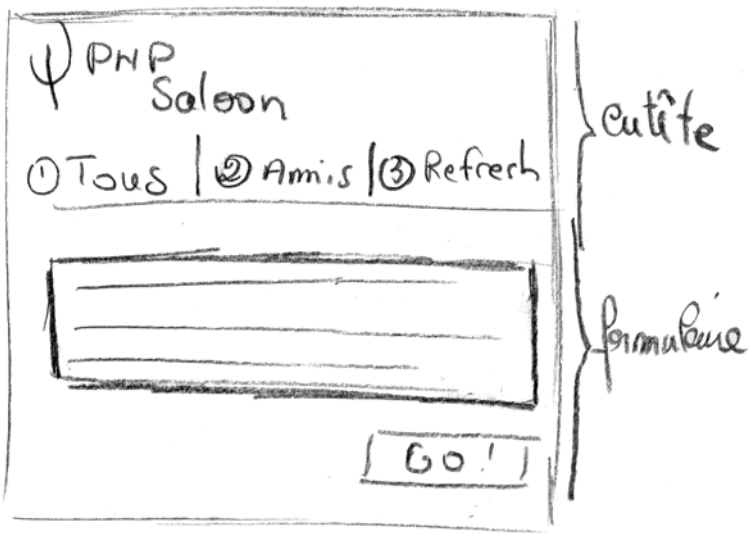


Figure 10–3 Envoi de message dans PHP Saloon i-mode

Pour les sessions, malgré les réserves émises au chapitre 6, nous utiliserons le support intégré de PHP favorisant le transport de l'identifiant de session sur l'URL en lieu et place des cookies. Celui-ci est le plus souvent activé par défaut chez les hébergeurs, il faudra donc veiller à conserver des URL de longueur raisonnable. Pour i-mode, cette longueur maximale est en effet fixée à 200 caractères.

Adaptation des feuilles de style

Notre route est tracée : il faut maintenant mettre les décisions en pratique. À défaut de téléphone i-mode, vous pourrez tester le résultat des modifications en utilisant l'excellent simulateur Wapag.

Nous allons opérer sur deux fronts :

1. une simplification drastique du code HTML produite par les transformations ;
2. l'ajout systématique du menu de navigation aux scripts de la page principale utilisée dans la version classique.

Le premier point se règle de manière brutale, on pourrait presque dire « à la hache ». Ainsi, si l'on prend comme exemple le code de la page d'identification, le résultat est éloquent.

OUTIL **Wapag**

Wapag est un outil de simulation i-mode utilisable directement en ligne pour effectuer quelques tests. Le rendu est la plupart du temps fiable même si celui-ci est réalisé en grande partie avec des feuilles de style CSS.

▸ http://www.wapag.com/en/simulation.html

Règle de transformation standard

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
        <head>
          <title>Bienvenue dans PHP Saloon!</title>
            <link rel="stylesheet"
              href="style/style.css" type="text/css" />
         </head>
         <body>
         <div align="center">
         <div align="left" style="width: 200px;">
         <fieldset>
                  <legend>
                  <img src="img/phpsaloon.gif"
                       alt="Logo PHP Saloon"/>
             </legend>
              <div class="innerdiv">
                     <xsl:apply-templates />
           </div>
               </fieldset>
            </div>
         </div>
          </body>
       </html>
   </xsl:template>
</xsl:stylesheet>
```

Règle simplifiée pour les terminaux i-mode

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
         <head>
            <title>Bienvenue dans PHP Saloon!</title>
         </head>
         <body text="#00b600" link="#b60000"
               vlink="#b60000">
          <img src="img/phpsaloon-petit.gif"/>
          <xsl:apply-templates />
         </body>
      </html>
   </xsl:template>
</xsl:stylesheet>
```

Globalement, il faudra procéder de la même façon pour l'immense majorité des règles de transformation. Le résultat obtenu est visualisable avec Wapag (voir figure 10-4).

Figure 10–4 Simulation de la page de connexion sur un terminal Nec

Pour la liste des connectés et celle des amis, tout comme pour l'envoi d'un message, il faudra procéder de manière identique, en prenant soin d'ajouter le menu évoqué précédemment (point 2).

Pour le réaliser, nous irons au plus simple : trois liens séparés par le traditionnel « | ». Cette méthode, simple et efficace, est souvent utilisée, même pour les pages classiques.

Chaque lien comportera deux Emoji :

1. le numéro de la touche utilisable pour valider le lien ;
2. un pictogramme signifiant le plus précisément possible le rôle du lien.

Une page HTML statique peut être utilisée pour tester le résultat avant d'incorporer le tout dans nos transformations.

```
<center>
&#59106; <a href="#">&#59116;</a> |
&#59107; <a href="#">&#59091;</a> |
&#59108; <a href="#">&#59098;</a>
</center>
```

Une fois validé (voir figure 10-5), ce code doit être incorporé dans les transformations utilisées par les listes de connectés. On obtient alors l'écran présenté en figure 10-6 et un PHP Saloon, version i-mode, désormais fonctionnel sur cette nouvelle plate-forme.

À RETENIR **Identifier la plate-forme**

Un des avantages de la plate-forme i-mode est de fixer un cadre qui peut certes paraître très contraignant, mais qui permet de développer une application web avec la garantie qu'elle fonctionnera sur l'ensemble du parc de terminaux.
Pour PHP Saloon, nous allons donc nous contenter de détecter la bannière i-mode au sein de l'en-tête USER-AGENT sans nous préoccuper de la nature exacte du terminal utilisé.
En France, comme dans de très nombreux pays, le type de navigateur communiqué est `portalmmm`. Cette indication nous suffit pour détecter un terminal i-mode.

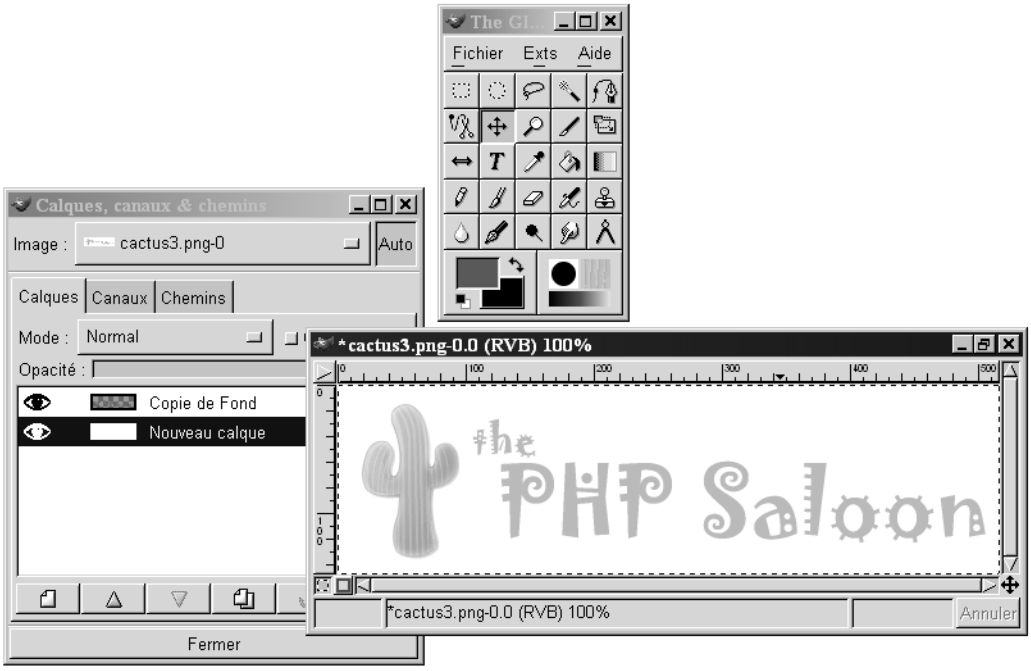
Figure 10–5 Exemple de menu de navigation dans PHP Saloon i-mode

Figure 10–6 Affichage de la liste des connectés (version i-mode)

En résumé...

Le recours aux transformations XSLT nous permet de bénéficier d'une très grande liberté, qu'il s'agisse de proposer des interfaces complexes, pour une utilisation sur un client lourd en entreprise, ou tout au contraire, des interfaces allégées pour l'utilisation mobile, par exemple sur un PDA, comme nous venons de le faire. En outre, et même si ce dernier point n'a pas été expérimenté pour PHP Saloon, ces transformations favorisent également la création de ponts vers d'autres applications, pour mettre en œuvre des outils de « single sign on » (authentification unique) ou d'échanges entre partenaires. On le voit, derrière une apparente complexité, le duo XML/XSL livre un potentiel très élevé, en termes de richesse fonctionnelle comme en amélioration des coûts de maintenance et d'exploitation.

chapitre 11


Fichier
Exts
Aide
Calques, canaux & chemins
Image :
cactus3.png-0
Auto
Calques
Canaux
Chemins
Mode :
Normal
Opacité :
Copie de Fond
Nouveau calque
Fermer
*cactus3.png-0.0 (RVB) 100%
the PHP Saloon
*cactus3.png-0.0 (RVB) 100%
Annuler

Protection des images et opérations graphiques avec GD

Avec les transformations XSL nous n'avons manipulé que des documents texte. Mais PHP dispose aussi d'un large éventail d'extensions pour créer ou modifier des documents binaires tels que les images ou des fichiers PDF.
Grâce à la bibliothèque GD, nous exploiterons ces possibilités pour protéger de tout détournement les photos des utilisateurs de PHP Saloon.

SOMMAIRE

MOTS-CLÉS

- Watermarking
- AlphaBlending
- GD

Jusqu'à présent, l'utilisation de notre couple infernal XML/XSLT a pu laisser penser que seuls les documents texte étaient manipulables avec PHP. Il n'en est rien, naturellement.

Tout d'abord, ainsi qu'évoqué précédemment, les transformations XSL peuvent produire des documents binaires via FO (Formating Objects). Enfin, et c'est tout l'objet de ce chapitre, PHP lui-même dispose d'extensions capables de manipuler directement les documents binaires, qu'il s'agisse de modifier des documents existants ou de les créer de toutes pièces.

Dans ce chapitre, nous allons nous intéresser plus particulièrement à l'extension GD (GD étant une librairie graphique). Cependant, PHP est particulièrement riche en la matière. Il est ainsi possible de créer des documents PDF pour améliorer le rendu imprimé et la transportabilité des informations.

À RETENIR **GD, standard du moment**

GD est disponible au sein de PHP depuis les débuts du langage. De nombreux exemples sont disponibles pour vous aider à en tirer profit.
L'ensemble reste néanmoins perfectible et l'avènement de PHP 5 devrait être l'occasion de mieux intégrer GD à PHP, et pour de nombreux projets d'atteindre une certaine maturité.

Problématique

Pour égayer la liste des connectés et faire de PHP Saloon un lieu d'échange plus convivial, chaque utilisateur peut télécharger la photographie de son choix. Celle-ci lui sera alors associée dans ses échanges avec les autres connectés. Cette fonctionnalité apparaît de prime abord complètement anodine.

Deux écueils peuvent pourtant rapidement donner des sueurs froides au gestionnaire du site :

1. La légalité des photos utilisées n'est en rien assurée par défaut.
2. Le détournement par des tiers, des photos déposées, est toujours possible.

COMPRENDRE **Responsabilité de l'auteur du site**

Avant de faire un site Web, il faut être conscient du rôle que l'on endosse implicitement : celui de directeur de publication. Et donc, pour les aspects les plus fâcheux, le rôle de responsable des contenus.
La loi est précise sur divers points quant à l'étendue de ce rôle, en matière de respect du droit d'auteur, ce qui est probablement le plus simple à gérer, mais aussi en matière de contrôle des contenus émis par les visiteurs du site ou diffusés par l'intermédiaire de celui-ci. Mais ceci est une toute autre gageure.

Le premier point ne connaît (hélas !) aucune solution technique ; il appartient donc à l'administrateur en charge du site de vérifier jour après jour qu'aucune photo manifestement illicite ne se soit glissée parmi les autres. Une possibilité consiste à n'intégrer les photographies des membres qu'après une vérification manuelle préalable.

Le deuxième point, quoique peut-être moins évident ou moins mis en avant, est pourtant tout aussi redoutable. Certaines photos peuvent tout simplement être réutilisées à l'insu de leurs propriétaires sur d'autres sites, voire dans certains documents commerciaux.

Pour résoudre ce problème, et éviter que les membres de PHP Saloon ne se retrouvent sans le savoir, acteurs à succès de publicités douteuses, nous allons nous servir des capacités graphiques de l'extension GD disponible dans PHP. Nous allons mettre en œuvre une technique en réalité très simple et connue sous le terme de « watermarking ». L'idée de base est équivalente à celle du filigrane. Nous allons ajouter systématiquement sur chaque image un élément en filigrane qui indiquera de manière indiscutable la provenance de la photo, la rendant inexploitable pour tout fraudeur.

Découverte de GD

GD est un compagnon historique de PHP ; une version des sources de la librairie est même incluse dans sa distribution, ce qui permet aujourd'hui de disposer de ressources importantes sur le sujet. Pour notre part, nous allons nous contenter d'effleurer ce thème, la documentation officielle de PHP listant de manière exhaustive les possibilités de la librairie. Il ne faut donc pas hésiter à s'y reporter, d'autant que les commentaires des utilisateurs sont très souvent précieux quant aux subtilités de GD.

PHP 4 **GD aussi**

Les utilisateurs habitués à PHP 4 ne doivent pas s'attendre à un bouleversement en matière de traitement d'image. Les successeurs de GD ne sont pas prêts, et la librairie poursuit son intégration dans PHP.

Principes d'utilisation

Nous allons explorer un cas d'école typique d'une utilisation de GD avec PHP. Nous verrons un peu plus loin que, pour PHP Saloon, les choses sont encore plus simples.

Notre exemple de prise en main consiste tout simplement à :

- créer une image de toutes pièces ;
- y ajouter différents éléments graphiques (dans la pratique, il pourrait s'agir de diagrammes, de camemberts...) ;
- délivrer directement l'image au navigateur.

CONFIGURATION **Disponibilité de GD**

Dans la plupart des cas et notamment chez les hébergeurs, GD est directement intégrée dans PHP (un `phpinfo()` peut permettre de s'en assurer).

Il peut être parfois nécessaire de charger l'extension avant de commencer à l'utiliser en s'aidant de l'instruction `dl()`.

```
dl('gd');
```

Plusieurs versions de GD coexistent parfois sur les systèmes sans parler de la version incluse dans la distribution PHP. Au final, il est parfois difficile de connaître précisément l'étendue des fonctionnalités disponibles (on pense notamment au support des images GIF, retiré pour cause de brevet), ou encore au support de certains formats d'image (PNG, JPEG) ou des polices « True Type » qui dépendent chacune de librairies externes.

Dans ce cas, il est possible de s'assurer que les opérations utilisées sont disponibles, en ayant recours à la fonction `get_extension_funcs()` qui permet de connaître les fonctions rendues disponibles par une extension.

```
php -r 'get_extension_funcs("gd");'
```

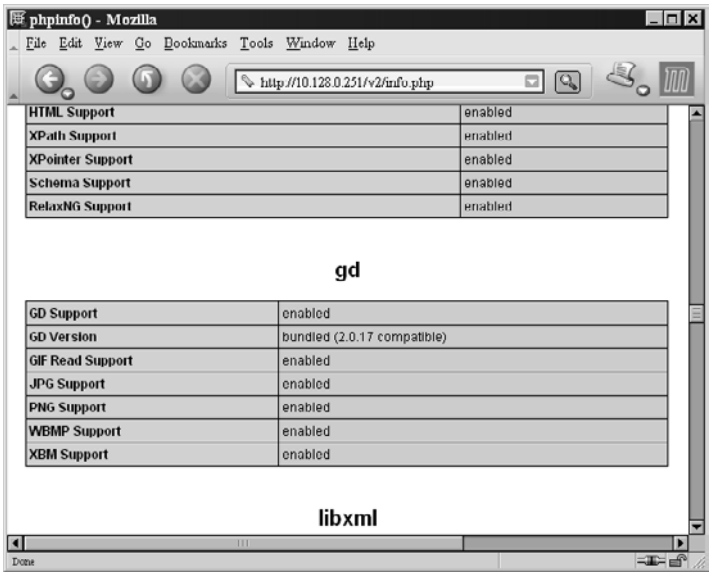


Figure 11–1 Vérification de l'activation de GD dans PHP avec `phpinfo()`

Les fonctions de GD commencent toutes, en général, par `image` et en dépit de leurs noms à rallonge, il ne faut pas y voir de complexité particulière.

Première confrontation à GD

```
<?
$img = imageCreateTrueColor(400,400);

$red   = imageColorAllocate($img, 255, 0, 0);
$white = imageColorAllocate($img, 255, 255, 255);
$gray = imageColorAllocate($img, 200, 200, 200);

imagefill($img, 0,0, $gray);
imagefilledellipse($img, 200,200, 300, 200, $white);
imagestring($img, 5, 150, 195, "PHP Saloon!", $red);

header('Content-type: ' . image_type_to_mime_type(IMAGETYPE_JPEG));

imageJPEG($img);

imageDestroy($img);

?>
```

Création de l'image

Définition des couleurs

Quelques opérations graphiques élémentaires sont mises en œuvre. Le chiffre 5 désigne l'une des cinq polices de caractères intégrées à GD, mais il est cependant possible de télécharger des polices additionnelles (dans ce cas, on utilisera plutôt les fonctions dédiées aux polices « True Type »).

Avant de renvoyer le code de l'image au navigateur du visiteur, il faut fixer le type de contenu. GD comporte une table de correspondance qui dispose du type MIME requis à partir des constantes simples utilisées.

Cette fonction produit tout le code de l'image.

Juste une histoire de ménage

Attention à ne pas laisser de lignes vides après ?>, pas même un simple retour à la ligne, sinon le contenu de l'image serait corrompu !

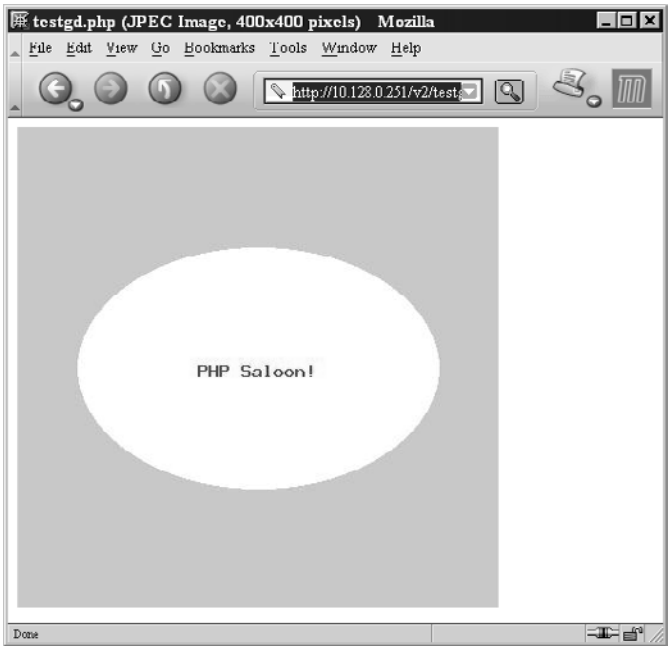


Figure 11–2
Image créée dynamiquement avec GD

Intégration des images dans les pages web traditionnelles

Cette image est loin de constituer une œuvre d'art, c'est certain (voir figure 11-2). Mais comme toutes les images créées à partir de PHP ou statiques, elle peut être intégrée dans des pages web de manière tout à fait classique. Le code ci-dessous permet de s'en assurer (voir figure 11-3).

```
<html>
   <head>
      <title>Test de GD</title>
   </head>
   <body background="red">
      <h1>Voici une image générée</h1>
      <center>
         <img src="testgd.php">
      </center>
   </body>
</html>
```

Figure 11–3 Intégration de l'image GD dans une page Web

MÉTHODE Authentification et génération d'images

Dans cet exemple, nous ne nous sommes pas préoccupés de la gestion des droits. Nous avons généré une image, et c'est tout.
Ce point particulier constitue un oubli classique, et il en résulte une faille de sécurité importante, puisque des informations sont alors accessibles, en dehors de tout contrôle.
Il faut donc, dans tous les cas, prévoir de contrôler le droit d'accès (via le cookie de session par exemple) et, le cas échéant, produire une image d'erreur.

Traitement des photos confiées à PHP Saloon

Le traitement requis pour PHP Saloon est à peine plus complexe que notre image de démonstration, juste un peu plus technique.

Nous allons réaliser une fonction qui va apporter plusieurs modifications à l'image téléchargée par l'utilisateur :

1. normaliser sa taille selon un gabarit donné (64 × 64 pixels) ;
2. ajouter un filigrane en transparence.

Voici le code résultant, où trois paramètres au minimum sont requis :

- l'image d'origine (elle sera téléchargée par le connecté à PHP Saloon) ;
- l'image à produire ;
- le filigrane.

Par défaut, notre fonction redimensionnera l'image originale selon le format du filigrane (pour PHP Saloon, il est prévu 64 × 64 pixels). Si le quatrième paramètre est placé à `false`, alors le filigrane est positionné au milieu de l'image d'origine dont le format a été conservé.

```
function watermark($srcfilename, $newname, $watermark,
                   $resize = true) {
   $photoImage =
            imageCreateFromString(file_get_contents($srcfilename));
   $logoImage = ImageCreateFromPNG($watermark);
   if ($resize) {
      $icone = imagecreatetruecolor(imagesx($logoImage),
                                    imagesy($logoImage));
   imagecopyresized($icone,$photoImage, 0, 0, 0, 0,
        imagesx($logoImage), imagesy($logoImage),
        imagesx($photoImage), imagesy($photoImage));
   } else
      $icone = $photoImage;
   ImageAlphaBlending($icone, true);
   ImageCopy($icone, $logoImage,
           max((imagesx($icone) - imagesx($logoImage)) / 2,0),
           max((imagesy($icone) - imagesy($logoImage)) / 2,0),
            0, 0, imagesx($logoImage), imagesy($logoImage));
   ImageJPEG($icone,$newname);
   ImageDestroy($photoImage);
   ImageDestroy($logoImage);
   if ($resize)
   ImageDestroy($icone);
}
```

Chargement de l'image originale

Puis chargement du filigrane

Recopie de l'image d'origine, redimensionnée si nécessaire

Ajout du canal Alpha

Superposition du filigrane à l'image d'origine (éventuellement au milieu de celle-ci si aucun redimensionnement n'a eu lieu).

Production de notre icône

Nettoyage de tout ce qui ne sert plus à rien.

Nous utilisons astucieusement la fonction `imageFromString()` pour ne pas avoir à nous soucier du type de l'image (sinon il aurait fallu tester le type de l'image nous-même, puis appeler les fonctions GD ad hoc, comme `imageFromJPEG()` ou `imageFromPNG()`).

Enfin, il nous reste un dernier point à examiner, pour ceux qui s'interrogent sur ce mystérieux « alpha blending ». Il s'agit en réalité d'ajouter un canal pour la transparence (comme il en existe pour les couleurs rouge, verte et bleue). Ceci fait que toutes les opérations graphiques vont correctement intégrer les transparences présentes dans les images.

On notera au passage que le résultat final repose sur le fait que l'image utilisée pour le watermarking dispose de zones transparentes (ce qui est permis de manière performante dans les images PNG), faute de quoi la copie serait opaque et notre image de départ, entièrement recouverte par le filigrane. Dans cet exemple, l'image utilisée est créée avec Gimp (voir à la figure 11-4 où un fond blanc a été ajouté, afin de rendre le filigrane visible).

Figure 11–4 Création du filigrane dans Gimp

On s'aperçoit par ailleurs que « filigrane » est un terme abusif, car il s'agit en réalité de superposition, toute l'illusion reposant sur la transparence.

Notre fonction doit être ajoutée à la phase d'inscription. Elle opérera sur l'image transmise via le formulaire de cette page, mais il est évidemment possible d'effectuer quelques tests immédiatement, comme le montrent les figures 11-5 et 11-6.

GD et les images GIF

Nous avons créé notre filigrane avec une image PNG parfaitement adaptée au traitement de la transparence. Une image GIF aurait pu être utilisée aussi, cependant le support des images GIF est rarement présent dans GD en raison d'un problème de brevet, et, l'on s'en doute, de redevance.

Figure 11–5 Image originale

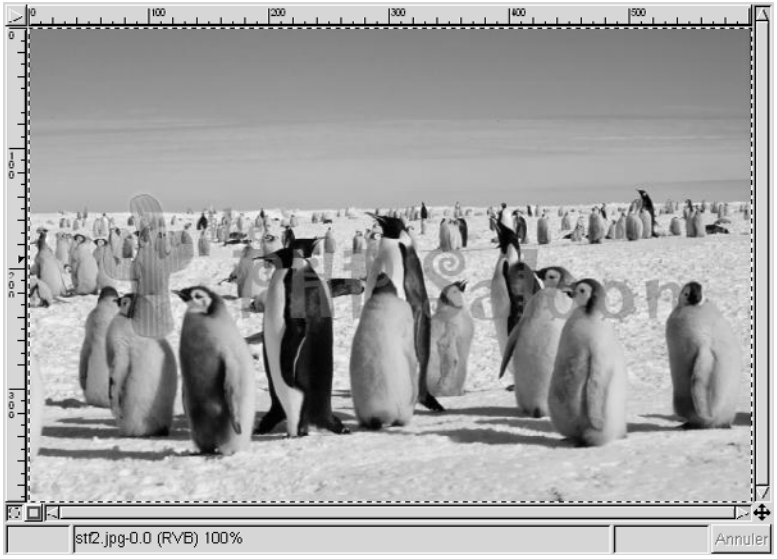


Figure 11–6 Image après watermarking

En résumé...

L'utilisation de GD au sein de PHP est caractéristique d'un large éventail d'extensions disponibles, qui permettent à PHP d'accéder à des ressources externes, qu'il s'agisse de fichiers image comme pour GD, ou de base de données, ce qui est le cas avec SQLite. Ces extensions donnent accès à des composants développés, le plus souvent, en C ou en C++ sans être confrontées à la complexité de l'interface de programmation par défaut : PHP joue ainsi parfaitement son rôle de langage glu.

chapitre 12


欢迎到PHP Saloon! - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://10.128.0.251/v2/site.
PHP Saloon
网上用户
朋友
stf
32 岁
1
Jean
20 岁
4

Internationalisation

L'internationalisation d'une application web apporte souvent la dernière touche professionnelle et peut s'avérer indispensable dans bien des cas pour toucher l'intégralité du public visé.
Nous verrons comment internationaliser PHP Saloon en explorant l'éventail de possibilités offertes par PHP bien sûr, mais aussi par le serveur web Apache grâce aux informations disponibles via le protocole HTTP.

SOMMAIRE

MOTS-CLÉS

- Multiviews
- Locale
- Gettext
- Accept language

COMPRENDRE

Les langues selon l'ISO et le W3C

La plupart du temps les différentes langues ne seront pas désignées par leur nom (d'ailleurs dans quel langue ce nom devrait-il être exprimé ?) mais de manière codée, avec deux lettres. Dans la plupart des cas ces lettres correspondent aux initiales du pays racine de la langue : FR pour le français, EN pour l'anglais. Toutefois l'attribution n'est pas toujours aussi limpide, l'ISO maintient une liste normative de ses correspondances.

En outre, chaque langue est susceptible de disposer de variantes. Pour le français, il peut s'agir des adaptations québecquoises, suisses ou encore belges. Pour l'anglais, on peut citer l'anglais américain. Dans ce cas la notation utilisée consite à compléter le code initial par un deuxième code : FR-CA ou EN-US, par exemple.

Dans certains cas, on pourra également préciser l'ensemble par le type de codage utilisé pour les caractères (ISO-8859-15 pour l'Europe occidentale par exemple).

- http://www.faqs.org/rfcs/rfc1766
- http://www.w3.org/WAI/ER/IG/ert/iso639.htm

Internationaliser PHP Saloon ?

Dans les chapitres précédents nous avons porté une attention toute particulière à l'accessibilité, à l'adaptation de PHP Saloon, aux différents types d'utilisateurs et de terminaux. Toutefois, cet effort ne serait pas complet si l'application ne devait au final proposer que le français au niveau de l'interface utilisateur.

En effet, même pour un outil plutôt classé dans la catégorie loisirs, Internet est ainsi fait que des utilisateurs du bout du monde vont essayer l'application, parce qu'un moteur de recherche leur aura proposé la page, parce que PHP Saloon est un moyen d'entrer en contact avec des utilisateurs francophones.

Cette réalité est transposable très rapidement pour les applications commerciales et les sites d'entreprises, y compris pour les PME et l'on pourrait même dire surtout pour les PME qui disposent en général d'un savoir faire particulier dans leur métier et dont la production peut satisfaire des besoins au-delà des frontières nationales.

Comme pour chaque chose, internationaliser n'est pas gratuit, d'une part nous allons le voir, l'application doit se prêter à cette transformation, et enfin, sauf à disposer d'un interprète, la traduction de l'environnement utilisateur en lui-même requiert des ressources extérieures. Sur ce point il faut admettre que la nature répartie des projets Open Source donne souvent un avantage compétitif non négligeable, l'intervention de la communauté permettant de disposer rapidement de sites et de logiciels dans plusieurs langues. Un argument « marketing » important qui agit tant sur la visibilité que sur l'attrait du logiciel.

Pour PHP Saloon nous allons porter notre effort à deux niveaux. D'une part, en tirant parti de notre plate-forme web (avec Apache), et, d'autre part, en utilisant un grand standard de la galaxie GNU fort heureusement disponible dans PHP sous forme d'extension.

Déterminer les attentes du visiteur et réagir avec HTTP et Apache

Nous allons donc internationaliser, mais encore faut-il avoir une idée des désidérata de l'utilisateur. L'objectif étant naturellement de ne pas obliger celui-ci à sélectionner manuellement sa langue de prédilection après avoir subi l'affichage d'une page par défaut selon toute probabilité incompréhensible pour lui.

Quiconque a déjà eu la malchance d'expérimenter des sites japonais ou russes a sans nul doute été confronté à la pénible et approximative recherche du lien vers la version anglaise pour tenter de retomber sur un discours plus abordable.

Découvrir les préférences des utilisateurs

Par chance le protocole HTTP à tout prévu.

Lors de notre exploration des sessions, nous avons déjà pu observer que les échanges entre un navigateur et le serveur web ne se limitent pas à une séquence de questions/réponses. Tant le navigateur que le serveur web précisent leurs échanges avec des informations additionnelles, les fameux en-têtes, dont le cookie naturellement.

Les préférences de l'utilisateur en matière de langues vont transiter jusqu'au serveur web de façon identique. Le navigateur, en fonction de la version installée, des préférences adoptées par l'utilisateur et stockées dans sa configuration va transmettre une information précise qu'il appartiendra au serveur web d'exploiter... ou non.

De prime abord le format de ces en-têtes est simple. Ainsi pour indiquer que la langue préférée de l'utilisateur est le français votre navigateur envoie très probablement un en-tête voisin de celui-ci :

```
Accept-Language: fr
```

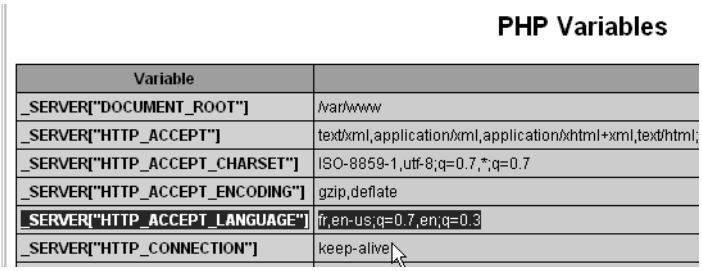
Les choses se compliquent ensuite légèrement, car définir une langue favorite est une chose, mais préciser les autres langues acceptables en est une autre.

Pour cela, l'en-tête `Accept-Language` va se décomposer en plusieurs éléments séparés de « , ». Chaque élément est dédié à une langue et comporte un coefficient qui précise le niveau de préférence de la langue en question. Ainsi, l'en-tête suivant :

```
Accept-Language: fr,en-us;q=0.7,en;q=0.3
```

précise que l'auteur de cet ouvrage préfère les documents exprimés en français, l'absence de coefficient donnant de fait la plus forte priorité au français, puis ceux exprimés en anglais, dans la variante américaine, mais avec une priorité moindre, les documents en pur style anglais.

Pour avoir une idée plus précise de la configuration de votre navigateur, il est possible de retrouver ses en-têtes dans le tableau super global `$_SERVER` disponible en PHP. Ce tableau est en outre affiché avec l'instruction `phpinfo()`, voir figure 12-1.



PHP Variables

Variable	
_SERVER["DOCUMENT_ROOT"]	/var/www
_SERVER["HTTP_ACCEPT"]	text/xml,application/xml,application/xhtml+xml,text/html;
_SERVER["HTTP_ACCEPT_CHARSET"]	ISO-8859-1,utf-8;q=0.7,*;q=0.7
_SERVER["HTTP_ACCEPT_ENCODING"]	gzip,deflate
_SERVER["HTTP_ACCEPT_LANGUAGE"]	fr,en-us;q=0.7,en;q=0.3
_SERVER["HTTP_CONNECTION"]	keep-alive

Figure 12–1
Vérification des préférences du navigateur à l'aide de `phpinfo()`

À RETENIR **Modifier ses préférences dans Internet Explorer et Mozilla**

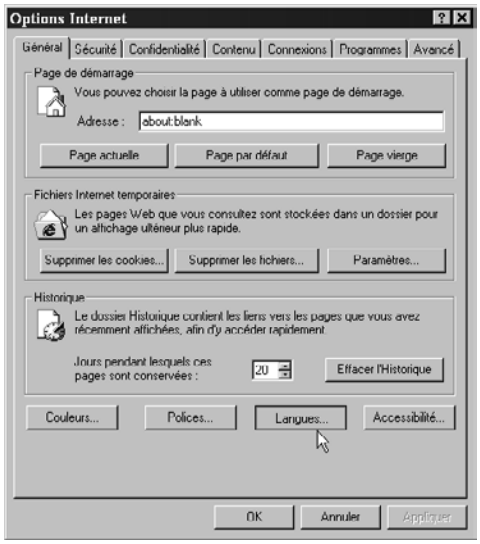
Chaque navigateur est livré avec un certain nombre de paramètres préfixés, c'est le cas notamment de la langue préférée le plus souvent en relation directe avec la langue choisie lors de l'installation.

Cependant, dans tous les cas, ce paramètre peut être modifié et affiné. Pour Mozilla et Internet Explorer la configuration n'est pas très compliquée et peut être ponctuellement modifiée.

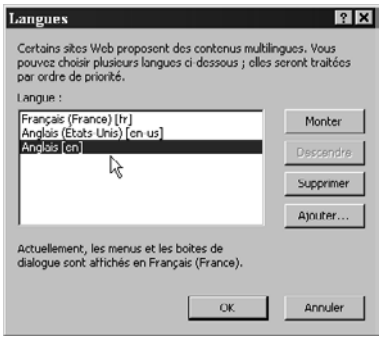
Avec Internet Explorer :

1 Ouvrir les options Internet

2 Dans l'onglet Général, cliquer sur le bouton Langues



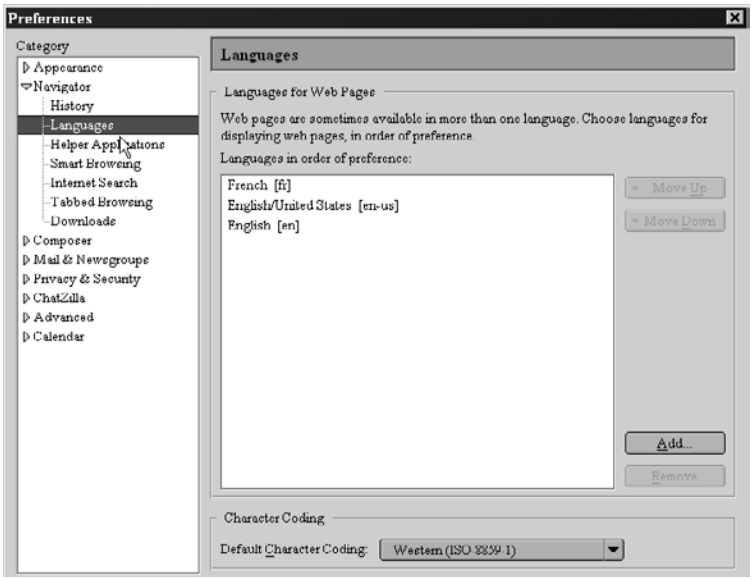
3 Modifier les priorités ou les langues préférées



Avec Mozilla :

4 Aller dans Préférences

5 Dans l'onglet Navigateur, sous rubrique Langues, modifier vos langues préférées



Il faut noter que ce même principe est retenu pour préciser au serveur les types d'encodage acceptés par le navigateur ainsi que les formats reconnus (JPEG, GIF, ZIP...).

On le voit, tout serveur web dispose donc des moyens techniques pour délivrer le contenu disponible le plus adapté. Encore faut-il le faire...

Sélectionner les ressources avec Apache

Contrairement à ce qu'on pourrait penser le support de cette adaptation dynamique est le plus souvent disponible en standard avec Apache, et il est selon toute probabilité disponible chez votre hébergeur. Cette fonctionnalité est en effet apportée par un module classique de la distribution Apache (`mod_negociation`).

Pour bénéficier de la négociation disponible dans Apache, nous allons référencer cette page avec une URL légèrement différente :

http://www.phpsaloon.com/hello

CONFIGURATION **Vérifier la configuration d'Apache**

L'intégralité de la configuration d'Apache est le plus souvent contenue dans le fichier `httpd.conf`.

Dans ce fichier, on pourra vérifier que la négociation de contenu est disponible en s'assurant que le module `mod_negotiation` est bien chargé au démarrage d'Apache :

```
LoadModule negotiation_module /usr/lib/apache/1.3/mod_negotiation.so
```

Par ailleurs, l'activation de cette négociation est subordonnée à l'utilisation de l'option `Multiviews` pour le répertoire des documents (ou tout répertoire parent). Ce point peut se vérifier en recherchant l'option `directory` correspondant :

```
<Directory /var/www/>
    Options Indexes Includes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Enfin, on peut être amené à contrôler les correspondances entre les langues (telles que désignées avec la notation ISO) et les extensions associées, celles-ci sont précisées à l'aide de l'instruction `Addlanguage` :

```
Addlanguage fr .fr
Addlanguage en-us .enus
```

▸ http://httpd.apache.org/docs/

Le principe retenu est particulièrement simple. Il suffit de respecter une convention de nommage pour les fichiers et de tronquer l'URL demandée avant l'extension. Considérons un premier exemple, le traditionnel « Hello world ! ». Par défaut, l'URL de la page serait du type :

▸ http://www.phpsaloon.com/hello.html

À partir de là, Apache va rechercher dans le répertoire courant (la racine dans notre exemple) les différentes versions de la ressource demandée. Naturellement cette recherche se fondera sur les préférences linguistiques.

Encore faut-il avoir défini ces alternatives mais la chose est excessivement simple. Pour désigner un document HTML en français on pourra par exemple le nommer :

```
hello.fr.html
```

Une version allemande pourrait quant à elle être disponible via un fichier nommé :

```
hello.de.html
```

En pratique, Apache va rechercher l'ensemble des ressources dont le nom avant toute extension est `hello`. Fort de cette liste, il appliquera les préférences utilisateurs en délivrant le contenu le plus adapté. Ce peut-être, faute de mieux, le fichier par défaut `hello.html`.

La correspondance entre le langage (façon ISO, donc par exemple `en-us`) et le supplément d'extension (`fr`, `de`...) est définie dans la configuration Apache. Il est tout à fait possible de définir des extensions personnelles pour tel ou tel dialecte ou variante d'une langue.

Ce mécanisme simple peut par exemple s'appliquer à notre squelette XUL (le fichier site.xul) et permet ainsi de proposer automatiquement l'interface applicative dans la langue du visiteur, ou au pire en anglais.

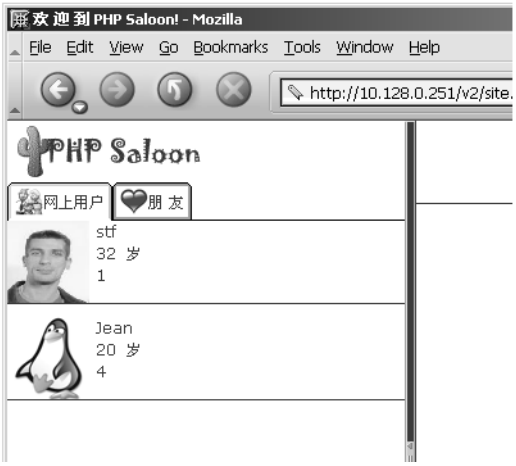


Figure 12–2 PHP Saloon en chinois

ALTERNATIVES Les tables de correspondances dans Apache

Apache permet de procéder différemment et de manière plus raffinée en associant à chaque ressource une table de correspondances, on parle de `type-map`.
Cette méthode permet de préciser les codages utilisés, la langue et ses variantes mais requiert naturellement plus d'effort puisqu'il s'agit de créer explicitement ces tables de correspondances.

- http://httpd.apache.org/docs/content-negotiation.html

Il faut toutefois rester prudent en matière de nommage des ressources. En effet, Apache prend en compte l'intégralité des préférences de l'utilisateur, ce qui inclut les préférences en matière de format et d'encodage. Avec une URL demandant la ressource `hello`, Apache peut ainsi être amené à choisir entre :

- `hello.html`
- `hello.fr.html`

mais aussi :

- `hello.gif`
- `hello.jpg`
- `hello.fr.jpg`
- `hello.cgi`

Dans ce cas, ce sont les préférences en matière de format qui prennent le pas sur la langue. Naturellement, ce sont les documents HTML qui héritent de la plus grande priorité chez tous les navigateurs, cependant quelques effets étranges peuvent apparaître en ce qui concerne les images ou même les CGI.

En cas de doute, il est toujours possible de vérifier la réalité des préférences émises par le navigateur avec `phpinfo()`. Voici par exemple les résultats obtenus avec Mozilla 1.6 et Internet Explorer 6 :

Préférences Mozilla

```
Accept : text/xml, application/xml,
    application/xhtml+xml,
    text/html;q=0.9,
    text/plain;q=0.8,
    image/png, image/jpeg, image/gif;
    q=0.2,*/*;q=0.1
```

Préférences Internet Explorer

```
Accept : image/gif, image/x-xbitmap,
    image/jpeg, image/pjpeg,
    application/vnd.ms-powerpoint, application/vnd.ms-excel,
    application/msword, application/x-shockwave-flash,
    */*
```

On notera que si Internet Explorer est approximatif et se contente d'indiquer les formats binaires supportés, Mozilla pour sa part est beaucoup plus pointilleux avec, en particulier, une priorité donnée aux documents XML sur les documents HTML...

PHP et Gettext

Apache permet donc déjà un premier niveau d'internationalisation. Hélas, celui-ci se révèle plus adapté pour les fichiers statiques que pour les fichiers dynamiques. En effet, la méthode requiert le clonage puis la traduction des fichiers. Or, s'il est naturel de dupliquer le contenu en autant de langues que nécessaire, il n'en va pas de même pour le code PHP (ce problème serait identique avec d'autres langages de développement comme Perl ou Python).

La modularité en question

Par ailleurs, ce mécanisme n'est mis en œuvre que dans la mesure où Apache intervient. Dans une application comme PHP Saloon, que nous avons voulue à juste titre modulaire, nombre de fichiers ne sont pas obtenus par l'intermédiaire d'Apache mais directement avec des instructions PHP, celles-ci ignorent tout du fonctionnement précédent.

Il va donc nous falloir, pour ce qui est de PHP, imiter Apache en identifiant les préférences de l'utilisateur puis en recherchant la bonne ressource.

Pour cela, nous allons définir deux fonctions : l'une, `lpreference()`, chargée de décoder l'en-tête `Accept-language`, l'autre, `localize()` chargée de trouver la ressource disponible la plus adaptée.

Contenu du fichier language.php

```
<?
function flip($a) {
   $b = array();
   foreach($a as $k => $v)
      foreach($v as $k2 => $v2)
         $b[$k2][$k] = $v2;
   return $b;
}
function lpreference() {
   $prefs = explode(',' , $_SERVER['HTTP_ACCEPT_LANGUAGE']);



   while (list($k, $v) = each($prefs)) {

      $prefs[$k] = explode(';' , $v);
      if (count($prefs[$k]) == 1)
         $prefs[$k][1] = 1;
      else {
         $prefs[$k][1] = explode('=', $prefs[$k][1]);
         $prefs[$k][1] = $prefs[$k][1][1];
      }
   }
```

Fonction utilitaire qui réorganise un tableau à deux dimensions, les lignes deviennent colonnes, les colonnes deviennent lignes. Cette manipulation un peu fastidieuse est requise car la fonction `array_multisort()` ne sait opérer un tri que dans un sens donné.

L'en-tête contient les préférences linguistiques. Nous le découpons avec `explode` pour isoler chaque préférence (elles sont séparées par des « , »).

Pour chaque langue, nous isolons l'éventuel coefficient de préférence.

ALTERNATIVE **Ouvrir les URL avec `fopen()`**

Puisque l'ouverture de fichier avec PHP ne nous permet pas de bénéficier des mécanismes d'internationalisation d'Apache, une solution pourrait être de ne plus accéder à aucun fichier en direct, mais au contraire de passer systématiquement par Apache. En effet, la fonction `fopen()` disponible dans PHP permet d'ouvrir non pas seulement des fichiers mais aussi des URL. Cette fonction étant utilisée de manière sous-jacente dans toutes les opérations sur les fichiers dans PHP (`include` par exemple), cette possibilité est disponible partout. Voilà qui pourrait donc résoudre notre problème ! Cela serait sans compter les problèmes de performances. Car effectuer une sous-requête HTTP pour chaque fichier inclus ou manipulé va rapidement se transformer en gouffre et plomber les performances. En conclusion, il est possible de confier l'intégralité de l'internationalisation à Apache, mais à un coût prohibitif.

```
   $f = flip($prefs);
   array_multisort($f[1],SORT_NUMERIC, SORT_DESC, $f[0]);
   return flip($f);
}
function basefile($file) {
   preg_match('/(.+)\.(.*)$/',$file,$matches);
   return $matches;
}
function localize($file) {
   static $prefs = false;
   if (! $prefs) $prefs = lpreference();
   $base = basefile($file);
   foreach($prefs as $langue) {
      $variant = $base[1] . '.' . $langue[0] . '.'. $base[2];
      if (is_file($variant))
         return $variant;
   }
   return $file;
}
?>
```

Puis tout est trié en fonction du coefficient.

Fonction utilitaire qui va nous donner le nom du fichier privé de son extension. Il s'agit d'une version améliorée du couple `basename()` / `dirname()`.

Pour chaque langue, il faut vérifier si le fichier correspondant existe.

Ces deux fonctions peuvent être testées simplement avec le code suivant :

```
<pre><?
include ( 'language.php');
print_r( lpreference() );
print ( localize ( 'hello.html' );
?></pre>
```

Pour les besoins du test, on pourra créer différentes versions du fichier `hello.html` utilisé dans les illustrations précédentes en plaçant dans chaque fichier le nom de la langue auquel il se réfère :

Contenu du fichier hello.en-us.html

```
Hello world américanisé ;) !
```

La figure 12-3 présente un résultat obtenu alors qu'un fichier `hello.en-us.html` est disponible en anglais mais pas sa variante française.

Dans PHP Saloon, ces fonctions vont nous permettre de charger les modèles de documents XML puis de sélectionner les règles de transformation en tenant compte des préférences linguistiques. Il serait tout à fait possible de procéder de même pour tous les fichiers manipulés. Cependant :

- il faut admettre que cette opération d'aiguillage réalisée en PHP et pas directement par Apache a un coût ;
- par ailleurs, une large partie des inclusions de fichiers dans PHP Saloon ne vise qu'à disposer de fonctions PHP liées à la logique métier de l'application et totalement indépendantes de la langue ;

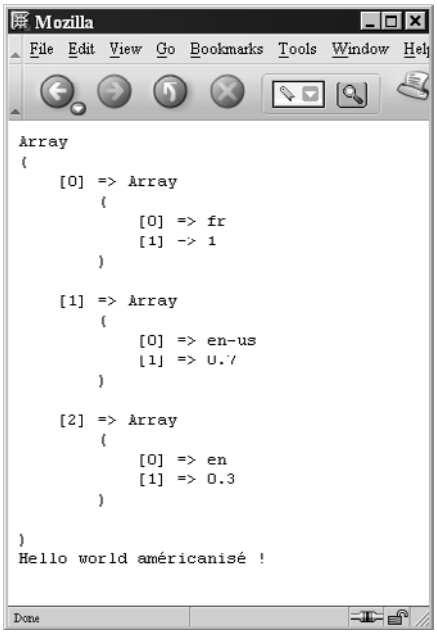


Figure 12–3 Sélection d'une ressource en fonction des préférences linguistiques

- enfin, comme évoqué précédemment, certains fichiers comportent trop de code PHP pour que leur duplication aux fins de traduction soit acceptable en termes de maintenance.

Sélection adaptée des éléments textuels de l'application avec GNU/Gettext

Ce dernier point va trouver sa réponse dans l'extension GNU/Gettext de PHP. Jusqu'à présent notre méthode n'a pas varié : des documents dupliqués pour chaque langue et sélectionnés judicieusement le moment venu.

Avec GNU/Gettext, nous allons au contraire nous intéresser à ces fichiers, qui principalement faits de code dynamique et non de contenus statiques, ne peuvent être clonés. Dans ces fichiers, certaines chaînes de caractères manipulées seront visibles pour l'utilisateur. Elles doivent donc être traduites.

GNU/Gettext est l'outil qui va nous permettre de gérer la sélection automatique de la bonne version de chaque chaîne en fonction de la langue. Dans nos fichiers PHP, une fonction `gettext()` sera intercalée partout où une chaîne doit être traduite. Considérons à titre d'exemple le code PHP suivant :

```
print "Hello world !";
```

OUTIL **GNU/Gettext**

GNU/Gettext est un environnement d'internationalisation complet.

Le programme xgettext repère et extrait l'ensemble des chaînes à traduire (celles qui font usage de la fonction `gettext()`). La quasi-totalité des langages est supportée, dont PHP.

xgettext produit un modèle d'annuaire dans lequel la version originale est précisée mais aucune traduction n'est spécifiée. Ce modèle sera confié à chaque traducteur qui pourra utiliser poEdit pour réaliser une version traduite de l'annuaire.

Ensuite `msgfmt` va optimiser l'annuaire pour que sa consultation (qui sera nécessaire pour chaque chaîne à traduire) ne soit pas trop pénalisante. Le résultat est un annuaire dans un format binaire.

En général, ces annuaires sont regroupés au même endroit sur le système, sous GNU/Linux il s'agit le plus souvent de `/usr/share/locale`, ce répertoire étant divisé en sous-repertoires par langue.

▸ http://www.gnu.org/software/gettext/

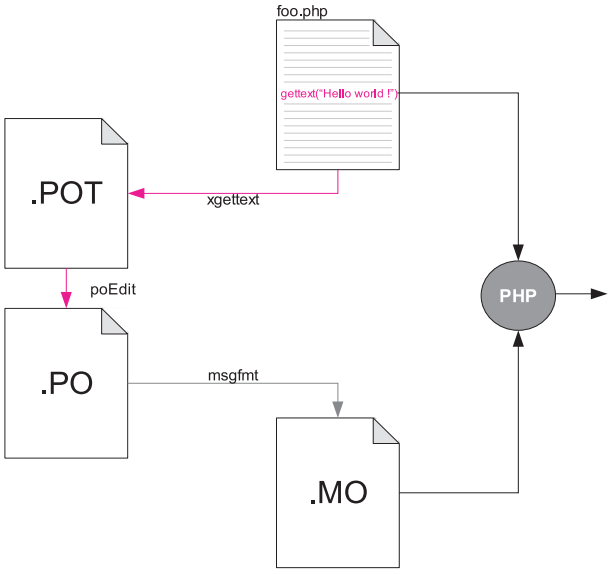


Figure 12–4 Processus d'internationalisation avec Gettext

Pour l'internationaliser, nous le modifions comme suit :

```
print ( gettext ( "Hello world !" ) );
```

En fonction des préférences linguistiques, la fonction `gettext()` renverra non pas la chaîne originale, mais une version internationalisée.

Pour que la méthode soit opérationnelle, il faut naturellement construire un annuaire de toutes les chaînes concernées et proposer des traductions de cet annuaire. Pour cela, GNU/Gettext propose un ensemble d'outils. Ces outils vont extraire les chaînes de caractères du code source, permettre d'affiner l'annuaire, puis de le traduire et enfin de le stocker de manière optimale. En production chaque invocation de la fonction `gettext()` aboutira en une consultation de l'annuaire.

Le mécanisme général est donc relativement simple, par ailleurs la sélection de la langue préférée repose sur un autre outil standard du monde Unix/Linux : les « locales ».

La notion de « locale » tente de factoriser en un seul outil l'ensemble des paramètres linguistiques. La langue naturellement, mais aussi le formatage du temps, des dates et des nombres. L'ensemble de ses paramètres est modifiable avec la fonction `setlocale()` disponible dans PHP.

OUTIL **poEdit**

Les annuaires utilisés par GNU/Gettext sont tous des fichiers texte (seules les versions figées pour distribution sont transformées pour améliorer les performances). Dans l'absolu toutes les opérations de traduction peuvent donc être réalisées manuellement avec un simple éditeur de textes.

Toutefois des outils existent pour rendre l'opération plus simple et moins longue. C'est le cas de poEdit, la référence en la matière, qui dispose également d'une fonction de traduction automatique capitalisant sur les traductions déjà réalisées (on sait qu'en effet un nombre assez important de mots et d'expressions reviennent d'un logiciel à l'autre). Une option qui ne dispense pas de tout travail, mais qui peut simplifier réellement l'internationalisation.

PoEdit est disponible sous Windows et tous les Unix.

- http://poedit.sourceforge.net/

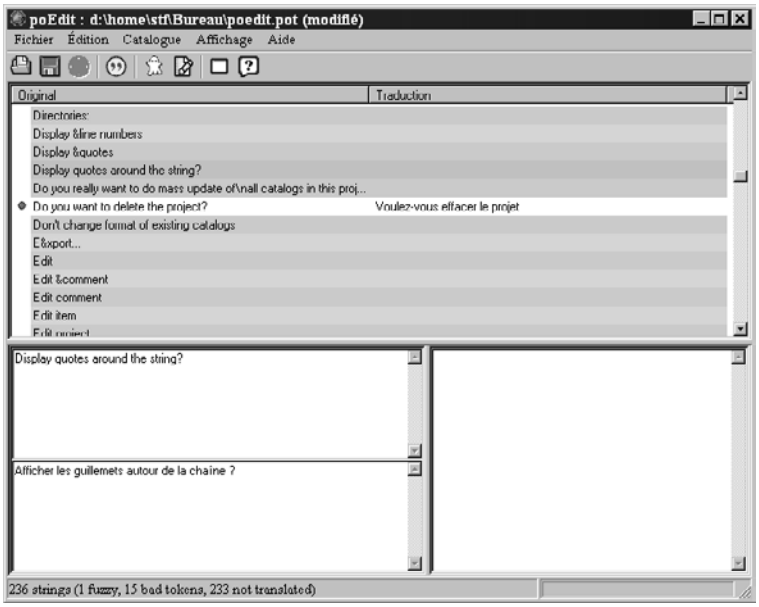


Figure 12–5 Interface principale de poEdit sous Windows

Cette fonction requiert deux paramètres, le premier précise la nature de l'élément qui sera modifié (par exemple le formatage des nombres), et le second, la langue retenue.

Pour obtenir une version française des chaînes de caractères, on pourra donc écrire :

```
setlocale ( LC_ALL , 'fr_FR.iso-8859-15' );
```

Ou de manière moins précise :

```
setlocale ( LC_ALL , 'fr' );
```

Dans ce cas, ni la variante linguistique (on pourrait par exemple souhaiter `fr_CA` pour le Québec), ni l'encodage attendu ne sont précisés.

Pour conserver les messages par défaut, tout en retrouvant un formatage de la date francisée (avec par exemple les jours avant le mois et non l'inverse comme c'est le cas dans le monde anglophone) on pourra utiliser :

```
setlocale ( LC_TIME , 'fr_FR.iso-8859-15 );
```

Ce système est mis en œuvre à tous les niveaux des systèmes Unix et vaut donc pour les messages d'erreur du système et des outils utilisés, PHP n'étant que l'un de ces environnements de plus à mettre en œuvre.

Dans PHP Saloon, il faudra donc identifier dans nos différents contrôleurs, les éventuelles chaînes à traduire et produire le catalogue de traduction ad hoc. Par chance, ces chaînes ne sont pas nombreuses, ce qui est la conséquence directe de nos choix architecturaux. Avec MVC, nous avons clairement segmenté l'application.

Il demeure que dans PHP Saloon comme pour la quasi-totalité des applications plusieurs mécanismes d'internationalisation vont cohabiter, chacun disposant de son mécanisme propre pour déterminer les préférences utilisateurs. Ce constat implique de porter une attention particulière à la cohérence entre les langues choisies :

- par Apache ;
- par nos fonctions PHP ;
- et sélectionnées avec `setlocale()`.

À défaut, nos visiteurs pourraient bien expérimenter un joyeux mélange. Ce point est notamment particulièrement sensible lorsque des variantes linguistiques entrent en jeu. Par exemple, `pt_BR` pour le Brésil ou encore `zh_TW` et `zh_CN` pour Taiwan et la Chine.

ALTERNATIVE **Forcer une langue donnée**

Dans ce chapitre nous avons évoqué la situation la plus délicate qui consiste à proposer d'emblée le contenu le plus adapté aux attentes du visiteur.
Toutefois le plus souvent, cette approche doit être complétée en proposant de choisir explicitement la langue d'affichage et ceci indépendamment des préférences de navigation.
Dans ce cas, on pourra placer des liens pour chaque langue (en général des images de drapeaux) et forcer les locales en positionnant par exemple un cookie (ce qui nécessitera de revoir un peu nos fonctions).
Il convient par ailleurs de ne pas négliger les éléments directement pris en charge par Apache en référençant dans les pages de chaque langue les fichiers localisés et non les versions génériques (`index.en.html` et non `index` par exemple).

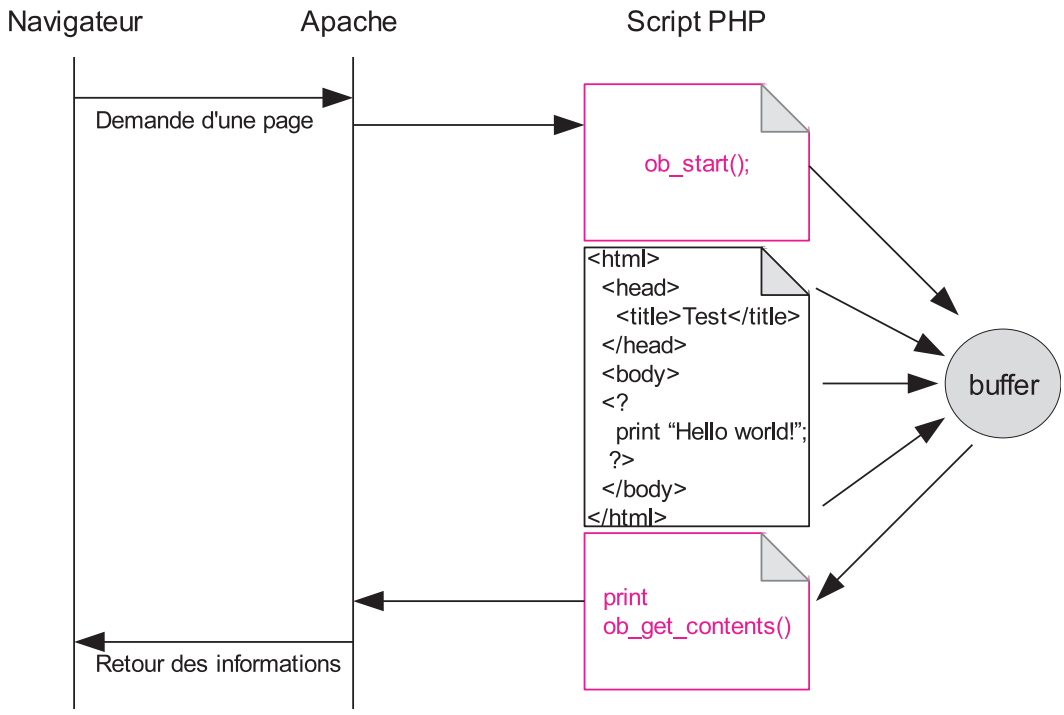
En résumé…

L'internationalisation d'une application web n'est pas chose facile. Cependant, sauf à vouloir se cantonner à un profil de visiteurs locaux, l'adaptation pour les principales langues du monde et le support de l'anglais notamment sont indispensables. Cela vaut pour les sites commerciaux comme pour les autres, on peut penser par exemple aux sites communautaires très en vogue.

Ainsi que nous venons de l'expérimenter, PHP peut à la fois tirer profit des outils mis à disposition par le serveur web utilisé et mettre en œuvre une localisation plus fine du contenu.

Cette internationalisation ne fait que mettre en exergue l'importance de nos choix précédents en matière d'organisation du code, de séparation des différents rôles comme le préconise l'architecture MVC. Qu'aurait-il été possible de faire si, comme c'est le cas dans nombre d'applications PHP disponibles sur le Web, tous les éléments (présentation, contenu et données métier) avaient été mélangés dans un même document ?

chapitre 13


Navigateur
Apache
Script PHP
Demande d'une page
ob_start();
<html>
<head>
<title>Test</title>
</head>
<body>
<?
print “Hello world!”;
?>
</body>
</html>
buffer
print
ob_get_contents()
Retour des informations

Optimisations et fonctions avancées

Internationalisation, adaptation dynamique au navigateur client, PHP Saloon a désormais tout pour plaire.
Restent toutefois des optimisations fort utiles à mettre en œuvre pour améliorer au quotidien l'exploitation du code et le simplifier, avec à la clé une meilleure qualité.

SOMMAIRE

MOTS-CLÉS

- auto_preprend_file
- stream_socket_server
- output buffering

Mutualisation du code commun avec les inclusions automatiques

Dans une application comme PHP Saloon le code développé ne peut rester monolithique. L'application est découpée en modules, en fichiers, ce qui est d'ailleurs essentiel si l'on souhaite isoler les différents composants mis en œuvre.

Il en résulte l'inclusion régulière de modules standards en début de fichier. Ainsi, si l'on observe le code PHP écrit pour gérer l'envoi de messages et les fils de discussion, on ne trouve pas moins de six fichiers inclus :

```
<?
  require_once 'inc/vue.php';
  require_once 'inc/i/icontroleur.php';
  require_once 'inc/controleurbase.php';
  require_once 'inc/utils.php';
  require_once 'inc/sessionvalide.php';
  require_once 'inc/connecte.php';
...
```

La plupart de ces éléments vont être constamment inclus, quel que soit le module en cours. C'est le cas des interfaces implantées par nos composants, mais aussi des fonctions utilitaires ou encore de la gestion des sessions.

Naturellement, il est possible de regrouper l'ensemble de ses dépendances dans un seul et même fichier que nous appelons le `prerequis.php`. Celui-ci importera l'ensemble des éléments généralement nécessaires et nous pouvons alors simplifier le code de nos fichiers PHP en remplaçant les multiples instructions `include` par une seule opération :

```
require_once 'prerequis.php';
```

Mais PHP propose une solution encore plus élégante. Grâce à la variable de configuration `auto_preprend_file`, il est possible de faire en sorte que notre fichier de prérequis soit automatiquement inclus en tête de tous les fichiers PHP. Cette variable est disponible dans le fichier `php.ini` :

```
auto_prepend_file = "/var/www/inc/prerequis.php"
```

Avec cette technique nous sommes sûrs que dès la première ligne de nos fichiers PHP, l'environnement logiciel minimal sera disponible.

CONFIGURATION Modification des paramètres PHP dans Apache et par répertoire

La modification des options de configuration de PHP au sein du fichier php.ini n'est pas toujours réalisable. C'est notamment le cas chez la plupart des hébergeurs fournisseurs de solutions mutualisées.

Toutefois, il est aussi possible de fixer certaines options en utilisant l'instruction `php_value` disponible dans Apache dès lors que PHP est configuré en tant que module. Ce qui est le plus souvent le cas.

Dans cette situation, l'instruction `php_value` peut être utilisée dans le fichier de configuration principal d'Apache (mais celui-ci est rarement plus accessible que `php.ini`) et surtout dans les fichiers .htaccess qui peuvent être créés pour un répertoire donné :

```
php_value auto_prepend_file
  /var/www/header.php
```

Le plus souvent on couplera l'utilisation d'`auto_prepend_file` à une adaptation de la variable `include_path` de manière à ne spécifier aucun chemin absolu pour accéder aux fichiers manipulés.

```
include_path = ".;/usr/share/phpsaloon/include;/var/www/inc"
auto_prepend_file = "prerequis.php"
```

Comme on est en droit de l'attendre, un équivalent de `auto_prepend_file` est disponible pour ajouter des éléments à la fin des fichiers PHP. Dans ce cas la variable de configuration est `auto_append_file`.

Pour PHP Saloon, nous pouvons en faire un usage tout à fait judicieux en plaçant le traitement de la vue dans ce fichier automatiquement inclus. En effet, par définition la vue intervient à la fin de chaque fichier PHP et de manière systématique. Plutôt que de dupliquer ce code il est possible de l'adapter pour le rendre générique et enfin le placer dans un fichier qui sera inclus pour nous.

Le code concerné est relativement simple :

```
$vue = new vue();
if (contenu_prefere() == 'xul')
      header('Content-type: text/xml');
print $vue->transform($controleur, 'echanges.xsl');
```

Cependant, il n'est pas générique. En effet, chaque portion de code diffère en opérant sur une feuille XSLT différente. On peut ainsi constater que, selon toute logique, le code précédent effectue le rendu des échanges de messages.

Pour contourner ce problème, nous allons imposer une convention : désormais les noms des fichiers PHP et de leur transformation XSLT associée sont identiques. Par exemple, la transformation XSLT du fichier `echanges.php` est disponible dans le fichier `echanges.xsl`.

Cette règle de programmation toute simple va nous permettre d'adapter légèrement le code initial de manière à le rendre générique :

```
$vue = new vue();
  if (contenu_prefere() == 'xul')
        header('Content-type: text/xml');
  print $vue->transform($controleur,
basename($_SERVER['PHP_SELF'],'.php') . '.xsl');
```

Nous utilisons pour cela la variable `$_SERVER['PHP_SELF']` qui contient le nom du script PHP courant. Cette nouvelle version du code peut être isolée dans un fichier `rendu.php` et auto-inclus en configuration PHP comme suit :

```
auto_append_file = "rendu.php"
```

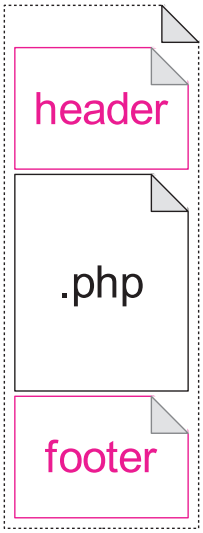


Figure 13–1 Agrégation automatique d'un en-tête et d'un pied de page dans PHP

Avec ces deux directives nous avons simplifié sensiblement le noyau applicatif de PHP Saloon. Il faut toutefois rester précautionneux avec ces mécaniques d'inclusion.

- Premièrement, inclure à tout va risque d'être pénalisant en termes de performance : car utilisé ou pas, le code PHP doit être analysé et compilé à la volée, ce qui engendre des opérations parfois plus coûteuses que l'exécution elle-même.
- Deuxièmement, le code ainsi inclus doit faire l'objet d'une attention toute particulière, car si une simple erreur s'y produit, l'intégralité des pages du site seront rendues inaccessibles (toutes victimes de la même erreur).

Contrôle et traitement a posteriori des documents produits

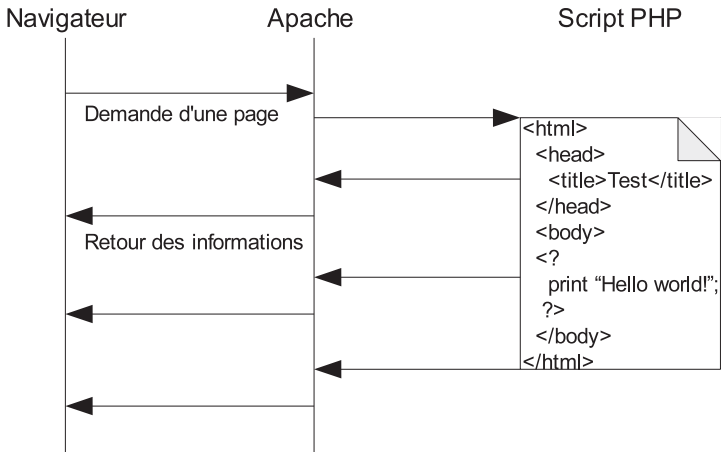


Figure 13–2 Flot des données en mode normal

L'inclusion automatique d'en-têtes et de pieds de page apporte une simplification en termes d'organisation du code, mais ne procure aucun gain en termes de contrôle sur la production du résultat. En effet, dès qu'un élément est produit : soit parce qu'il s'agit directement de contenus statiques non PHP, soit via les instructions `print` ou `echo` par exemple, l'interpréteur PHP perd le contrôle sur l'information. Celle-ci est transférée dans les mains d'Apache qui, le plus souvent, va la transmettre sans attendre au navigateur du visiteur. Ce fonctionnement explique d'ailleurs pourquoi la modification des en-têtes HTTP avec l'instruction `header()` ne peut plus fonctionner si des éléments ont déjà été produits pour l'utilisateur.

PHP propose toutefois de contrôler ce flot en modifiant le parcours. Il est ainsi possible de stocker le contenu destiné au navigateur avant de le diffuser, par exemple pour lui faire subir un traitement.

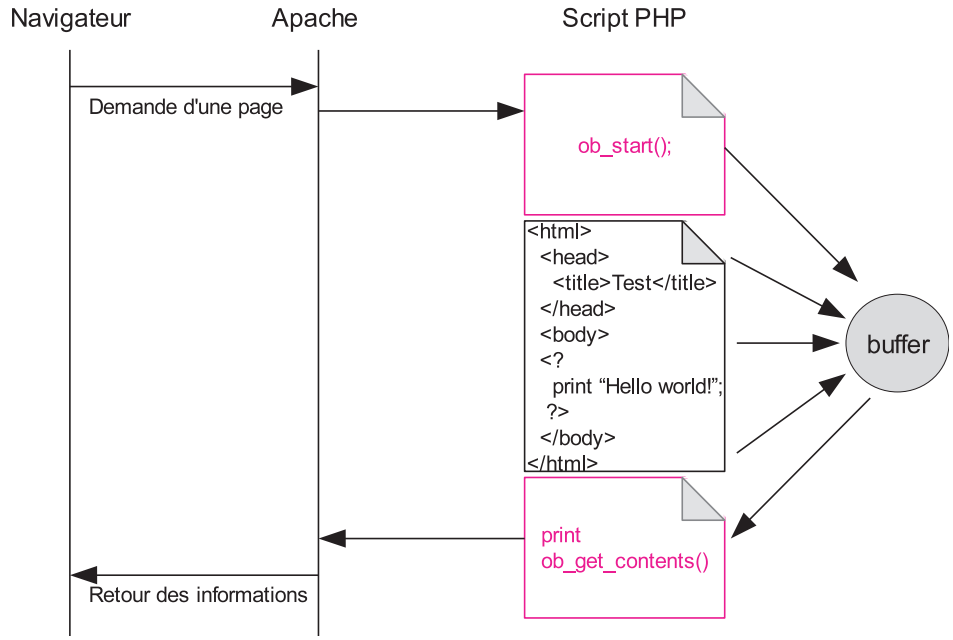


Figure 13–3 Flot des données avec mise en tampon

Pourquoi procéder de la sorte dans PHP Saloon ?

1. Pour gagner en performance et diffuser un contenu compressé, donc plus rapide à transporter sur Internet. Le navigateur, disposant du contenu plus rapidement pourra afficher la page dans un délai bien plus court. D'où un meilleur confort pour l'utilisateur.
2. Pour simplifier la création de documents qui seront ensuite repris dans leur ensemble pour une transformation globale, à la manière d'un modèle.

Compression des pages à la volée

Pour envoyer nos pages compressées, nous allons utiliser les fonctions de mise en tampon évoquées précédemment. Dans l'en-tête, nous utiliserons `ob_start()` qui marquera le début du stockage. Il convient naturellement dans notre cas de s'assurer que rien n'a déjà été envoyé au navigateur. Dans le pied de page, la fonction `ob_get_contents()` permettra d'obtenir tout le contenu produit. Il faut alors le compresser, puis adapter les en-têtes pour indiquer qu'une compression a été effectuée.

ALTERNATIVES mod_gzip
pour doper les performances

PHP dispose en propre de tous les mécanismes nécessaires à la compression des pages. Toutefois, et en particulier dans le cadre de sites mixtes, constitués à la fois de pages statiques et de pages dynamiques, il peut être judicieux de laisser la compression des échanges à Apache.
Dans ce cas c'est l'intégralité des contenus textes, statiques ou non, qui seront compressés. Apache dispose pour cela d'un module ad hoc, il s'agit de `mod_gzip`. Celui-ci ne fait pas partie intégrante de la distribution Apache mais le projet est téléchargeable depuis SourceForge.

- http://sourceforge.net/projects/mod-gzip/

PHP nous permet de réduire davantage ce processus très simple, puisque la fonction `ob_gzhandler()` effectue tout simplement toutes les opérations du pied de page. Celle-ci peut être directement passée en paramètre à `ob_start()` et sera alors appelée par PHP à la fin du script.

Finalement, il suffit donc de légèrement modifier notre fichier d'en-tête en ajoutant simplement :

```
ob_start("ob_gzhandler");
```

Nous sommes là dans l'usage le plus élémentaire des fonctions de mise en tampon qui peuvent être utilisées notamment pour reprendre du code existant en ajoutant des éléments ou en modifiant le contenu produit sans toucher au code original.

Découplage complet entre logique métier et vue

Ces fonctions peuvent aussi être utilisées pour pousser encore plus loin le découplage entre le contrôleur et la vue. Ainsi, dans PHP Saloon, il nous est impossible d'ignorer l'existence des traitements qui seront réalisés par la vue. D'ailleurs, il appartient au contrôleur de produire non pas simplement un document XML mais de le produire dans un format et selon une interface donnée (en l'occurrence sous la forme d'un objet DOM).

Modèle utilisé pour l'identification

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<phpsaloon>
  <formulaire>
    <connecte>
      <pseudo name="form[pseudo]" question="Nom de code"
              type="string">
        <![CDATA[foo]]>
      </pseudo>
      <motdepasse name="form[motdepasse]" question="Mot de passe"
                  type="secretstring">
        <![CDATA[bar]]>
      </motdepasse>
    </connecte>
  </formulaire>
  <info type="message">
    Pas inscrit(e)? <a href="inscription.php">Cliquez-ici</a>!
  </info>
</phpsaloon>
```

En utilisant les fonctions de mise en tampon nous pourrions laisser le contrôleur produire son document XML comme il le souhaite. Au terme du script PHP, le contenu produit serait alors récupéré dans le tampon et traité pour être rendu.

Un exemple possible consiste à revoir le cas du formulaire de connexion à PHP Saloon. Nous avons choisi d'ouvrir un modèle en XML, de l'analyser, puis de modifier l'arbre XML résultant. En réalité nous pourrions procéder tout autrement en nous contentant de produire un document XML texte sans même analyser quoi que ce soit. Charge à la vue de terminer le travail comme bon lui semble.

Création d'un document XML texte de façon simplifiée

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<phpsaloon>
   <formulaire>
      <connecte>
         <pseudo name="form[pseudo]" question="Nom de code"
                type="string">
<?
      // code PHP ad hoc (déterminera le nom du pseudo à placer ici
?>
         </pseudo>
         <motdepasse name="form[motdepasse]" question="Mot de passe"
                    type="secretstring">
<?
   // idem pour le mot de passe
?>
         </motdepasse>
      </connecte>
   </formulaire>
   <info type="message">
<?
   // éventuel message d'erreur
?>
   </info>
</phpsaloon>
```

Cette méthode permet de tirer profit de modèles statiques dans lesquels un code PHP minimaliste insère les éléments dynamiques. Ce document est produit en ignorant tout de ce qui sera opéré comme traitement par la suite.

Pour aboutir au résultat escompté, il suffirait alors de modifier le pied de page proposé au début de ce chapitre en ajoutant une phase d'analyse du tampon.

Pied de page modifié

```
   $controleur = new DOMDocument();
   $controleur->loadXML(ob_get_contents());
   $vue = new vue();
   if (contenu_prefere() == 'xul')
         header('Content-type: text/xml');
   print $vue->transform($controleur,
basename($_SERVER['PHP_SELF'],'.php') . '.xsl');
```

Dans le cas de PHP Saloon, le gain n'est pas forcément évident dans la mesure où nous avions déjà adopté une architecture (MVC) très modulaire et que la phase d'analyse XML est, quoi qu'il arrive, indispensable.

Malgré tout, on observe que dans le cadre d'un découpage des responsabilités, notamment entre équipes de développement, cette méthode permet d'obtenir encore plus de latitude. En effet, côté contrôleur, la production du document XML de référence n'est pas tributaire d'un modèle de données imposé (en l'occurrence DOM) et de la même façon, côté vue, la disparition de cette contrainte permettra d'adapter l'implantation en fonction de l'expérience et des évolutions d'usage. On pourrait par exemple imaginer utiliser un processeur XSLT DOM dans une première phase, puis adopter un outil différent par la suite, voire laisser les transformations à la charge d'un filtre Apache, cela en toute transparence pour la partie contrôleur.

Optimisation de la modularité avec les flots personnalisés

Comme on le voit, PHP propose des mécanismes sophistiqués pour manipuler les flots de données qui sont produits au cours de l'exécution des scripts. PHP 5 étend encore ces fonctionnalités en proposant tout un éventail de nouvelles possibilités pour manipuler l'information avec encore plus de facilité.

Ainsi, nous avons pu constater avec l'internationalisation de PHP Saloon que l'accès sélectif aux fichiers de telle ou telle langue n'était pas sans difficulté. Il nous a fallu définir notre propre fonction de sélection à intercaler partout entre le document à manipuler et les fonctions d'inclusion d'ouverture. Par exemple pour charger un modèle XML :

```
include (localize ('identification.xml'));
```

Dans le cas présent ce n'est pas dramatique, mais il peut rapidement en être autrement, notamment au sein des règles de transformation avec des instructions `xsl:include` qui vont passer au travers des mailles de notre internationalisation.

Par chance, PHP va nous permettre de créer notre propre protocole pour manipuler les fichiers (des flots de données comme les autres). Il existe naturellement un certain nombre de protocoles standards, même si dans PHP Saloon nous n'en n'avons pas nécessairement eu l'utilité. Voici quelques exemples :

CONFIGURATION **Activer le support des streams étendus**

Pour que les différents protocoles réseau soient disponibles avec l'ensemble des fonctions PHP comme `fopen()`, il peut être nécessaire de modifier la valeur de la variable `allow_url_fopen` dans le fichier php.ini. Pour que le support soit actif, la variable doit avoir la valeur TRUE.
Le support de ces streams peut aussi être désactivé à la compilation.

```
include 'foo.php';
include 'file://foo.php';
include 'http://www.stephanemariel.com/foo.php';
include 'http://ftp.stephanemariel.com/pub/foo.php';
include 'compress.zlib://foo.php.gz
```

Dans ces exemples, le protocole `compress.zlib` permet d'accéder à des fichiers compressés directement. La fonction `stream_get_wrappers()` permet d'obtenir la liste des protocoles disponibles.

Exemple de protocoles disponibles avec PHP 5 en ligne de commande

```
v1:/var/www/v2# php -r 'print_r(stream_get_wrappers());'
Array
(
    [0] => php
    [1] => file
    [2] => http
    [3] => ftp
    [4] => compress.zlib
)
```

Tous ces protocoles sont disponibles et applicables partout où des fichiers et des flots sont manipulés, avec `include`, `require`, `fopen`. Enfin, la bibliothèque XML utilisée par PHP pour manipuler et créer les documents DOM est compatible avec ces possibilités, au niveau des interfaces PHP, comme dans l'exemple suivant :

```
<?php
  $dom = new domdocument();
  $dom->load("compress.zlib://xml/identification.xml.gz");
?>
```

Mais aussi au niveau des instructions XSLT. On pourrait donc imaginer, dans PHP Saloon, de charger directement les modules XSLT compressés :

```
<xsl:include href="compress.zlib://page/standard.xsl" />
```

On notera par ailleurs que c'est le protocole `file` qui est utilisé par défaut pour accéder à un fichier local, les deux exemples suivants sont donc identiques :

```
include 'foo.php';
include 'file://foo.php';
```

Pour PHP Saloon, nous pourrions donc simplifier la localisation en définissant une version remaniée du protocole `file`, par exemple nommé `lfile` (pour localized files). Ce protocole prendrait automatiquement en charge la sélection des fichiers demandés en fonction des préférences linguistiques.

À RETENIR Au-delà des protocoles, les filtres

Les possibilités de PHP en matière de flot ne se limitent pas aux protocoles. Il est possible de définir des filtres. Les filtres ne s'occupent pas du transport, comme le font les protocoles mais transforment les données à la volée.

La liste des filtres disponibles est accessible grâce à la fonction `stream_get_filters()` et de nouveaux filtres peuvent être définis d'une manière analogue aux flots, mais en utilisant la fonction `stream_filter_register()`.

Pour chaque flot manipulé, par exemple avec `fopen`, les fonctions `stream_append_filter()` ou `stream_prepend_filter()` permettront de modifier la suite de filtres appliqués.

Nous pourrions alors écrire :

```
include 'lfile://xml/identification.xml';
```

Et dans les fichiers XSL :

```
<xsl:include href="lfile://page/standard.xsl" />
```

Dans les deux cas, le fichier ouvert serait celui correspondant aux préférences indiquées par le navigateur.

Mais peut-être la définition d'un nouveau protocole n'est-elle pas aussi simple qu'il y paraît ? Bien au contraire, il suffit de définir une classe avec quelques méthodes clés (assez naturelles au demeurant) et d'utiliser la fonction `stream_wrapper_register()`.

Fonction	Rôle
`stream_open`	Appelée lors de l'ouverture du flot.
`stream_close`	Appelée lors de la fermeture du flot.
`stream_read`	En charge de la lecture des données.
`stream_write`	En charge de l'écriture.
`stream_eof`	Indique si la fin du flot est atteinte.
`stream_tell`	Retourne la position dans le flot de données.
`stream_seek`	Positionne le pointeur à un endroit précis dans le flot de données.

Ces méthodes constituent le minimum vital, mais des méthodes complémentaires peuvent être requises notamment lorsque la notion de répertoire est indispensable.

On le voit, le potentiel offert par PHP 5 est considérable, on pourrait par exemple songer à définir des protocoles permettant un couplage transparent vers les bases de données.

Suppression d'Apache

Avec les streams, PHP 5 permet la manipulation sans limite des flots de données associés notamment aux fichiers. Mais cette notion de flot s'étend assez naturellement aux données réseau et aux connexions. PHP 5 va donc proposer, sur ce terrain aussi, de nombreuses améliorations et notamment deux nouvelles instructions clés :

- `stream_socket_client()`
- `stream_socket_server()`

Cette deuxième instruction permet ni plus ni moins de transformer votre PHP en un serveur, par exemple pour délivrer du XML, et de répondre à une requête SOAP. Aucune limite en la matière puisque PHP vous laisse définir votre protocole et l'agencement des échanges. Dans la plupart des cas, on souhaitera se caler sur HTTP, quitte à n'implanter qu'une sous-partie.

L'exemple suivant permet de se faire une idée de la méthode à suivre pour créer un serveur et illustre la simplicité avec laquelle cela est possible :

```
<?
  $ecoute = stream_socket_server('tcp://0.0.0.0:42/', $errno,
                                 $errstr);

if ($ecoute)
   while ($connexion = stream_socket_accept($ecoute)) {

     fwrite ($connexion, "<!-- Hello this is the PHP Super
                     ➥ Server.\n");
     fwrite ($connexion, "May I help you ? -->\n");
     $xml = '';
     while (($ligne = fgets($connexion)) && strcmp(trim($ligne),
            "."))
        $xml .= $ligne;
     $dom = new domDocument();
     $dom->loadXML($xml);
     $xsl = new domDocument();
     $xsl->load('test.xsl');
     $xslproc = new xsltprocessor();
     $xslproc->importStylesheet($xsl);
     fwrite($connexion, $xslproc->transformToXML($dom));
     fclose($connexion);

     print "Hourra! Et un client de servi!\n";
   }
print $errstr;
?>
```

Écoutons sur un port donné (le Web serait 80 mais nous avons choisi un port exotique) en utilisant le protocole TCP. 0.0.0.0 permet d'écouter sur toutes les cartes réseaux disponibles. Au besoin, il est possible de préciser l'adresse IP d'une seule carte.

Nous allons traiter en boucle tous les clients qui se présentent. Dès que c'est le cas, une connexion est construite pour nous à partir de la socket d'écoute.

Nous commençons par une petit message de bienvenue...

Puis nous allons lire tout ce qui nous est envoyé par le client.

En espérant qu'il s'agisse de XML.

Nous allons transformer tout ça.

Et renvoyer le résultat à notre client.

Le travail est terminé, nous fermons la connexion. Il reste à attendre le prochain client.

Naturellement dans la réalité on ne se contenterait pas d'une petite transformation XSLT, mais notre serveur pourrait jouer les intermédiaires entre deux applications métier et se muer en traducteur entre le XML d'un premier logiciel et celui du second.

Ceux qui douteraient encore de la réalité de cet exemple pourront consulter les copies d'écran du test de ce serveur en ligne de commande.

Figure 13–4
Lancement du serveur sur une machine

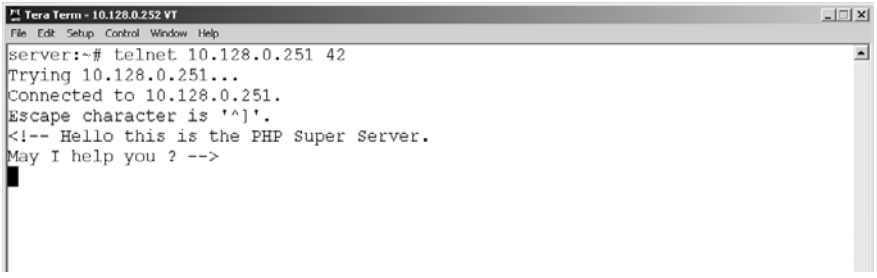


Figure 13–5
Connexion au serveur depuis une autre machine

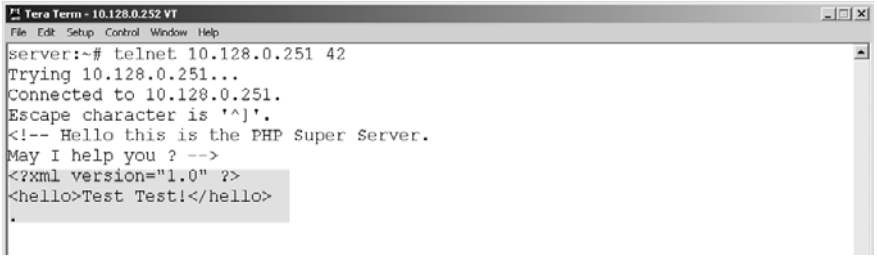


Figure 13–6
Envoi d'une requête XML

Figure 13–7
Réponse du serveur

Naturellement, PHP nous permet avec `stream_socket_client()` d'exploiter notre serveur de manière moins primitive, comme le montre cet exemple et le résultat obtenu.

```
<?
  $connexion = stream_socket_client('tcp://10.128.0.251:42/', $errno,
$errstr);
  if ($connexion) {
   fgets($connexion);
   fgets($connexion);
   fputs($connexion, "<?xml version=\"1.0\" ?>
<hello>Hello tous!</hello>\n.\n");
   while(! feof($connexion) )
      print fgets($connexion);
  }
?>
```

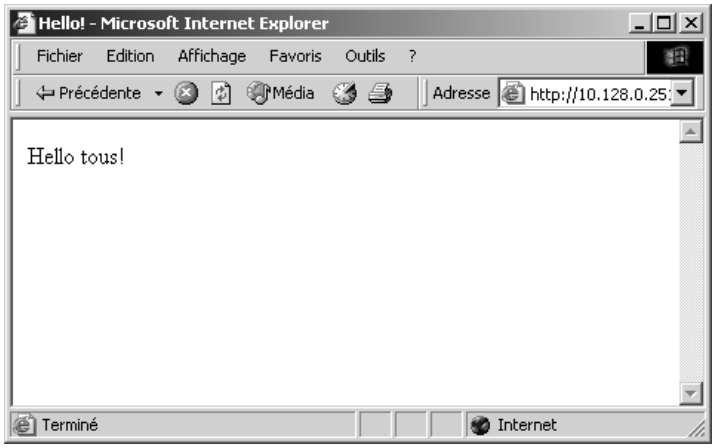


Figure 13–8 Appel du serveur depuis un script PHP et affichage du résultat dans un navigateur

Pour faciliter l'écriture de clients, PHP intègre par ailleurs l'extension CURL. Les fonctions de cette extension évitent d'avoir à réinterpréter soi-même les protocoles les plus courants (comme HTTP, FTP...) et simplifient sensiblement l'interaction avec des services distants.

ALTERNATIVES NanoWeb

Faire un serveur complet à la main n'est pas forcément de tout repos. Pour ceux qui souhaiteraient se passer réellement d'Apache tout en bénéficiant d'un serveur entièrement conçu autour de PHP, il existe NanoWeb qui répond totalement à cet objectif tout en conservant des performances acceptables.

- http://nanoweb.si.kz/

En résumé…

Organisation du code, contrôle fin des flots de sortie, possibilité de fonctionnement autonome, PHP 5 accroît encore les capacités de PHP. Autant d'aptitudes qui répondent de mieux en mieux aux exigences actuelles : tout d'abord en termes fonctionnels, puis avec une adéquation réelle entre PHP 5 et l'univers des services web, des échanges XML, en termes de qualité et de sécurité, et enfin avec des options raffinées pour maîtriser le périmètre applicatif au plus juste. De quoi s'interroger légitimement sur l'opportunité de recourir à des solutions plus lourdes et indubitablement plus coûteuses, y compris pour les gros projets.

Votre serveur PHP à domicile

annexe
A

Configuration PHP en local

Il est bien sûr possible de tester PHP chez soit, sans se préoccuper d'Internet. Dans ce cas, rendez-vous directement à la section 2.

Avec la démocratisation des accès haut débit, câble ou ADSL, on peut être tenté d'héberger directement son application web à domicile, sur son ordinateur personnel. Si cette solution vous tente, voici quelques éléments clés avant de démarrer.

Avantages et inconvénients

Avant toute chose il est important de bien peser le pour et le contre d'un hébergement à domicile. Transformer son salon en salle machine n'est évidemment pas sans conséquence.

Cette boutade met en avant un premier point, souvent négligé. Un serveur web, avec ou sans PHP, doit rester opérationnel 24 heures sur 24. Cela implique que l'ordinateur qui sera utilisé pour héberger l'application devra rester allumé en permanence.

Cet état de fait a comme première conséquence notable une consommation électrique non négligeable. Songez que le seul processeur peut consommer jusqu'à 100 W ! De plus le corollaire de cette forte consommation est la chaleur et par voie de conséquence l'importante nécessité de refroidissement de l'ordinateur. Quiconque s'est approché d'un micro-ordinateur aura pu constater que le bruit est une réelle nuisance. Naturellement, il est possible d'améliorer les choses, mais dans la plupart des cas il faudra apprendre à vivre avec ce doux ronronnement.

Si ces nuisances ne constituent pas vraiment un point en faveur de cette solution, elles mettent en avant l'un de ses avantages clés : le serveur est à portée de main pour réaliser toutes les opérations de mise à jour et de modification. Cette proximité facilite énormément le travail tout en procurant une impression de contrôle et de sécurité certaine.

Cette latitude en termes de maintenance se retrouve évidemment au niveau de la configuration logicielle. Il devient possible de construire des environnements logiciels bien plus complexes ou moins « tendances » que ceux disponibles en standard chez les hébergeurs de services mutualisés.

Compte tenu de la jeunesse de PHP 5, l'utilisation d'un ordinateur personnel peut notamment permettre de disposer d'un hébergement sans attendre la migration nécessairement progressive des hébergeurs.

On le voit, la liberté est totale, mais comme souvent, elle s'accompagne d'une responsabilité élargie, notamment en termes de sécurité. Il vous appartiendra d'assurer les mises à jour, la sécurisation quotidienne de votre serveur, en appliquant les patchs de sécurité par exemple. Ce travail quotidien peut être rapidement fastidieux, d'autant qu'il ne constitue pas l'objectif premier de l'installation.

Enfin, dernier point, à ce jour les liaisons ADSL et câble sont asymétriques, c'est-à-dire que (dans le cas des solutions grand public) le débit en réception est sensiblement plus important que le débit en émission. Ce détail technique n'a que peu d'importance pour un usage classique. En effet, nous téléchargeons tous bien plus de données que nous n'en émettons.

Cependant, les possibilités s'inversent quand on se place du point de vue du visiteur d'un site hébergé à domicile. En effet, celui-ci est limité par la capacité d'émission, le plus souvent de 256 Kbit/s, alors qu'en téléchargement la capacité dépasse souvent 2 Mbit/s, soit 10 fois plus !

Si la mise à disposition de fichiers était dans vos intentions, sachez que cela risque d'être très laborieux, pour vos utilisateurs mais aussi pour vous, car le moindre téléchargement saturera votre capacité d'émission, et même les quelques éléments émis pour obtenir une page web seront paralysés, sauf action adéquate. Votre propre consommation Internet risque donc de souffrir.

Malgré l'ensemble des contraintes qui peuvent de prime abord paraître conséquentes, l'amélioration constante de la qualité des connexions haut débit rend réellement viable l'hébergement d'un service classique à domicile, pour peu que les téléchargements soit déportés sur un autre site.

Adresse IP et nom

Avant même de vous lancer à la conquête de PHP 5 et du Web, il faudra vous préoccuper des problèmes d'adressage et de nom.

Principe

En règle générale, pour accéder à un serveur, nous faisons appel à son nom : cela nous paraît logique et naturellement le plus adapté. Cependant, au sein du réseau, les serveurs ne sont pas désignés par ce nom, mais par une adresse numérique plus simple à manipuler. Cette adresse est utilisée pour trouver la route entre un serveur et votre propre ordinateur. Pour assurer la correspondance entre les deux, le réseau Internet dispose donc d'un système d'annuaire, le DNS (*Domain Name Server*).

Jusque-là tout est simple. Hélas, cette mécanique est lourde et suppose que la correspondance entre un nom et son adresse ne change que très rarement. Or, avec l'explosion du nombre d'Internautes, les besoins en adresses sont de plus en plus difficiles à satisfaire (celles-ci sont en effet composées de quatre nombres compris entre 0 et 255, le nombre d'adresses disponibles est donc limité). Pour résoudre ce problème, les fournisseurs d'accès mutualisent leur pool d'adresses entre tous leurs clients. Après tout, nous ne nous connectons pas tous au même instant.

Cette pratique ne pose aucun problème en soit, notamment pour les connexions par modem, cependant un service, pour être contacté doit pouvoir être localisable de manière fiable à chaque instant. Le changement périodique d'adresse imposé par le fournisseur d'accès est donc problématique.

Naturellement, de plus en plus de fournisseurs commencent à proposer une adresse IP fixe. Néanmoins, cette option reste encore minoritaire. Que faire alors si vous n'êtes pas l'heureux possesseur d'une adresse IP fixe ?

Il existe par chance des services simples qui proposent un accès à leur annuaire (et donc à un nom) et un outil pour effectuer automatiquement les mises à jour en cas de changement. Ces annuaires, configurés pour ce type de fonctionnement sauront donc à chaque instant vous retrouver, vous et votre adresse IP.

Selon que vous soyez sous Linux ou Windows (il existe aussi des outils équivalents sur Mac) différents outils permettent cette mise à jour dynamique. La plupart du temps, ceux-ci peuvent se connecter à plusieurs services d'annuaire. N'hésitez donc pas à explorer les possibilités de chacun. Dans tous les cas, les logiciels de mise à jour doivent être lancés dès le démarrage et rester actifs.

SERVICE DynDNS

Les services d'annuaire dynamique sont légions, le plus connu est DynDNS à tel point que ce terme est devenu générique et désigne globablement un service d'annuaire dynamique.
Certaines sociétés proposent une prestation plus complète et vous permettront d'obtenir un nom plus séduisant, mais le principe reste le même.

▸ www.dyndns.org

Microsoft Windows

Bali DynDNS est un logiciel simple et efficace, il est disponible en français et supporte plusieurs services parmi les plus connus.

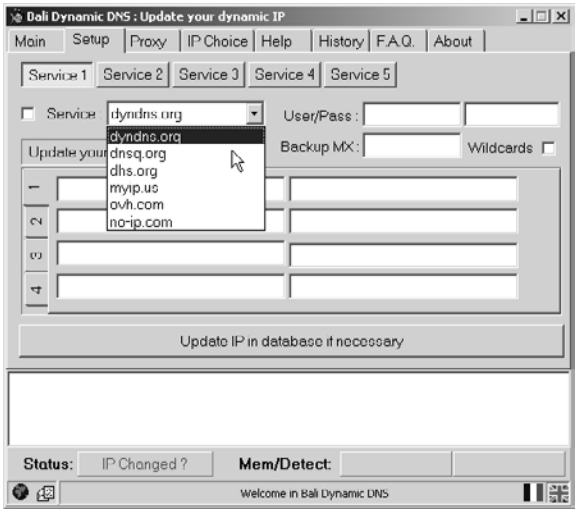


Figure A–1 Fenêtre de configuration de Bali DynamicDNS

Plusieurs services peuvent être utilisés simultanément mais la plupart du temps un seul sera utilisé. Lorsque le logiciel est lancé une icône est accessible dans la barre des tâches.

- http://www.baliciel.com/softwares/baliddns.htm

Linux

Sous Linux, le logiciel ez-ipupdate est particulièrement efficace. Il peut être lancé simplement en arrière-plan et gère tout seul. Le fichier de configuration est simple et le logiciel livré avec des exemples.

```
service-type=easydns
user=XXXX
host=mon-nom-a-moi.com
interface=eth1
wildcard=on
cache-file=/tmp/ez-ipupdate.cache
# uncomment this once you have everything working how
you want and you are
# ready to have ez-ipupdate running in the background
all the time. to stop it
# you can use "killall -QUIT ez-ipupdate" under linux.
daemon
```

Il est aussi possible de ne pas lancer ez-update sous forme de démon et le coupler aux scripts appelés par le client DHCP directement lorsque l'adresse est attribuée. La procédure dépend alors de votre client.

- http://ez-ipupdate.com/

Installation proprement dite

Une fois votre machine correctement accessible par tous, encore faut-il avoir un serveur web installé, deux installations types sont proposées, sous Microsoft Windows et sous Linux.

Sous Microsoft Windows

Installer Apache

- http://httpd.apache.org

L'installation d'Apache sous Windows est réellement un jeu d'enfant. Un installeur est disponible et l'installation se résume à une séquence de boîtes de dialogue élémentaires.

Figure A–2
Démarrage de l'installation d'Apache sous Microsoft Windows

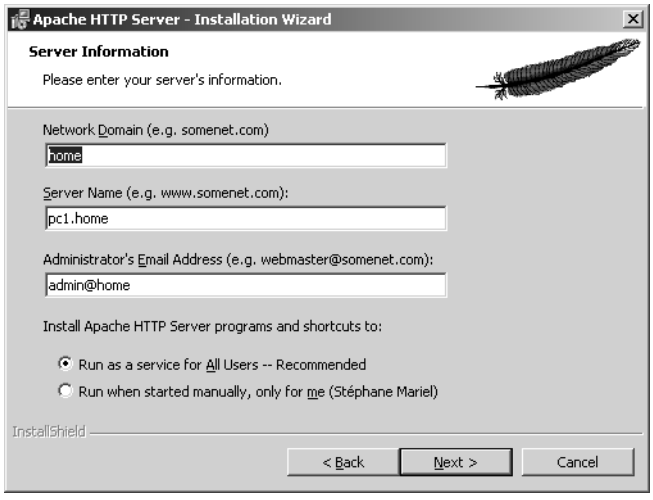


Figure A–3
Fin de l'installation d'Apache et sélections des options

En phase de test, le plus simple est cependant de ne pas utiliser la possibilité offerte d'intégrer Apache en tant que service et de se contenter de démarrer celui-ci à la main.

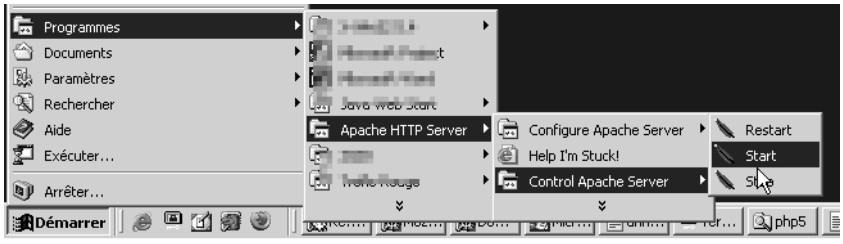


Figure A–4
Lancement d'Apache depuis le menu Démarrer

À RETENIR Localhost

On a vu précédemment que pour pouvoir nommer une machine il faut un annuaire. Toutefois, il existe un nom, disponible par défaut et qui désigne la machine locale, que celle-ci soit connectée au réseau ou non. Ce nom est `localhost`, il est utilisé par tous les systèmes d'exploitation.

L'installation réalisée, il est possible de vérifier immédiatement le bon fonctionnement du serveur avec n'importe quel navigateur. Une page par défaut apparaît alors. La racine du site web est positionnée sur le répertoire `c:\Program files\Apache Group\Apache\htdocs`.

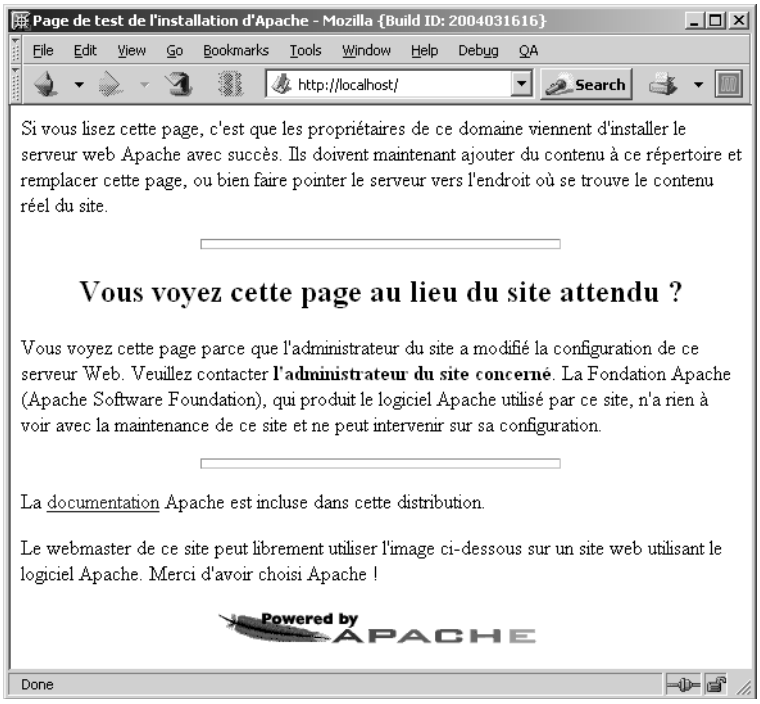


Figure A–5 Test d'Apache sous Microsoft Windows

Installer PHP 5

▸ http://ww.php.net

PHP dispose lui aussi d'un installeur. Celui-ci n'est pas disponible pour PHP 5, mais surtout, l'installation réalisée n'est pas satisfaisante. Il est préférable de télécharger le fichier `.zip` comprenant l'intégralité des extensions, le programme PHP en ligne de commande et les modules PHP.

La procédure consiste à décompresser cette archive dans un répertoire. Contrairement à certaines pratiques décrites sur Internet, il est vraiment déconseillé d'utiliser la racine de votre disque. Le répertoire d'installation d'Apache est beaucoup plus indiqué. Il suffit donc de créer un sous-répertoire `php` à l'intérieur puis de tout décompresser.

Là encore on évitera de suivre les modes d'emploi fantaisistes parfois proposés. Aucune des librairies PHP n'est à copier dans le répertoire système de votre ordinateur. Seul le fichier `php.ini-recommended` doit être copié dans ce répertoire (sous Windows 2000 et XP, il s'agit de `c:\winnt`) et renommé `php.ini`.

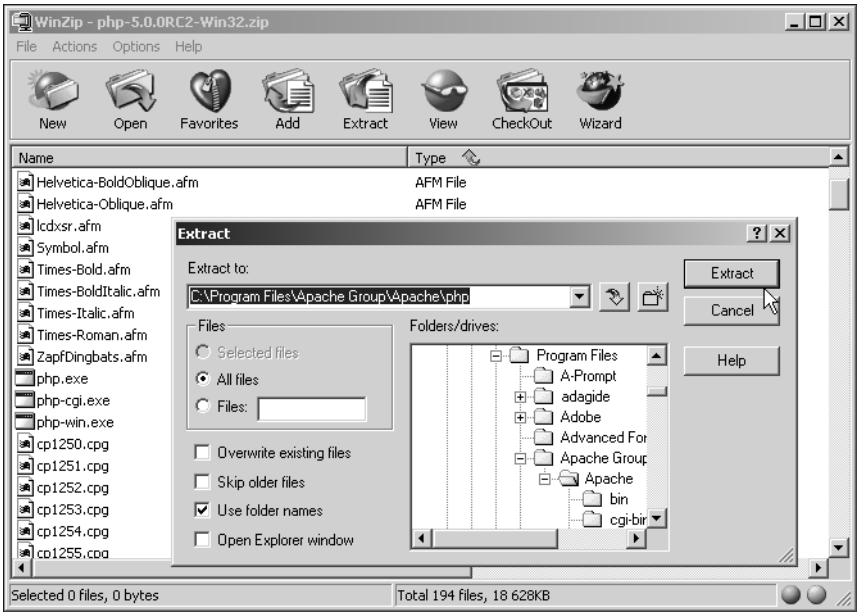


Figure A–6
Extraction des binaires PHP pour Microsoft Windows

Avant de tester notre installation, quelques modifications doivent être apportées, d'une part au fichier de configuration Apache et d'autre part au fichier de configuration de PHP.

Dans le premier cas, il s'agit de s'assurer qu'Apache saura utiliser PHP. Voici les quatres modifications clés à apporter au fichier `httpd.conf` (celui-ci est situé dans le répertoire `C:\Program Files\Apache Group\Apache\conf` et peut être modifié et testé très simplement en utilisant les raccourcis du menu Démarrer) :

1. Ajouter la ligne suivante à la liste des modules :

```
LoadModule php5_module php/php5apache.dll
```

2. De même pour :

```
AddModule mod_php5.c
```

3. Déclarer l'interpréteur PHP pour les fichiers avec l'extension PHP dans la section `<IfModule mod_mime.c>`

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

4. Ajouter `index.php` comme alternative dans la section `<IfModule mod_dir.c>`

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

Enfin pour `php.ini`, il s'agit de préciser le répertoire où sont situées les extensions. Dans PHP Saloon, nous utiliserons `gd`, `sqlite`, `dom` et `xslt` entre autres. Pour simplifier, nous allons demander le chargement automatique de celles-ci, ce qui nous évitera d'utiliser `dl()` dans le code de PHP Saloon. Pour ce faire, il suffit de modifier le chemin des extensions :

```
extension_dir = "c:/program files/apache group/apache/php/ext"
```

Puis de lister les extensions à charger par défaut :

```
extension = php_gd2.dll
extension = php_xsl.dll
```

DOM et SQLite sont chargés d'emblée, il suffit donc d'ajouter GD et les transformations XSLT. Ces quelques opérations réalisées, PHP est prêt à être testé. Il ne reste qu'à démarrer Apache et tester avec un fichier tout bête :

```
<? phpinfo() ?>
```

On constate alors l'affichage du détail de la configuration PHP et avec soulagement, qu'il s'agit bien de la version 5 !

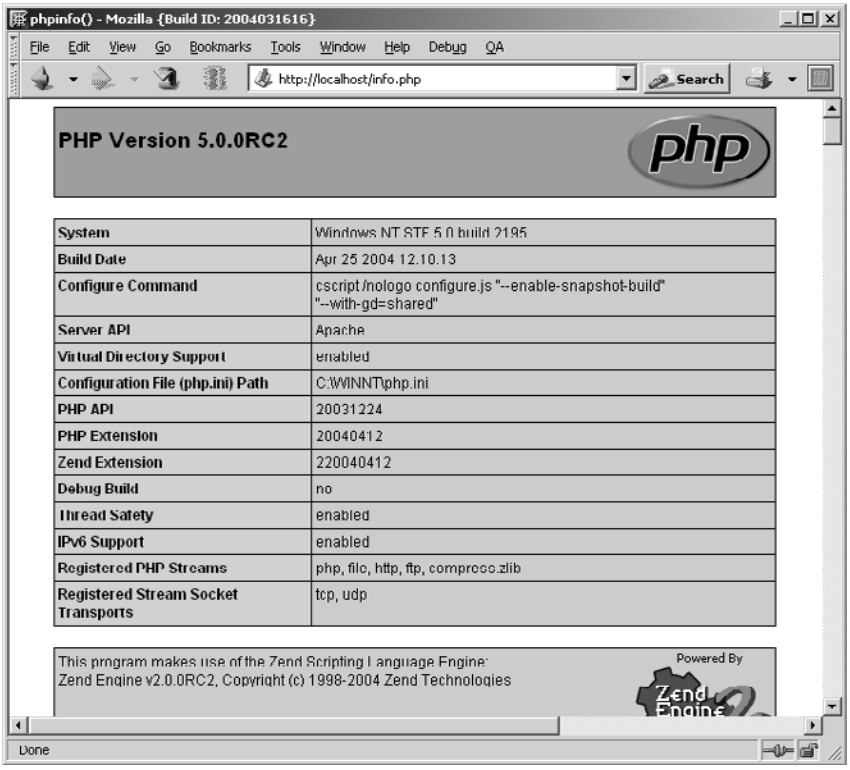


Figure A–7
Test d'intégration entre Apache et PHP 5

Sous Linux

Installer Apache

Le plus souvent, Apache est déjà installé par les distributions Linux. Si tel n'est pas le cas, la procédure sera encore plus simple et expéditive que sous Microsoft Windows.

Si vous utilisez Debian, la commande suivante suffira :

```
apt-get install apache
```

Vous pouvez aussi utiliser `synaptic`.

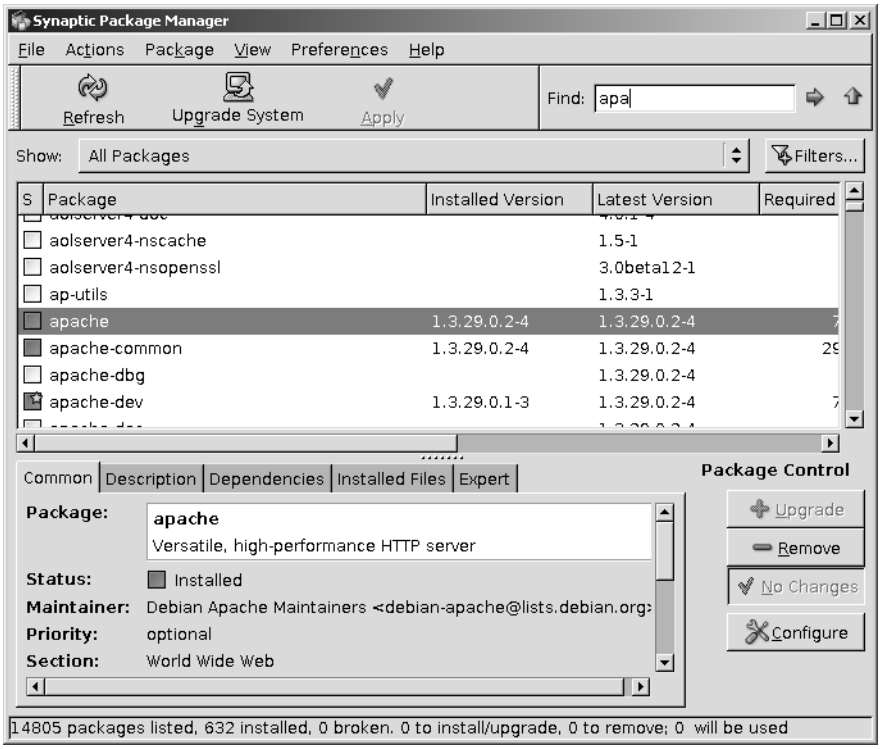


Figure A–8
Installation d'Apache sous Debian avec synaptic

Pour la distribution Mandrakelinux, l'utilisation de `rpmdrake` et une simple case à cocher devraient résoudre le problème. La plupart du temps, la racine du site web est positionnée sur le répertoire `/var/www`.

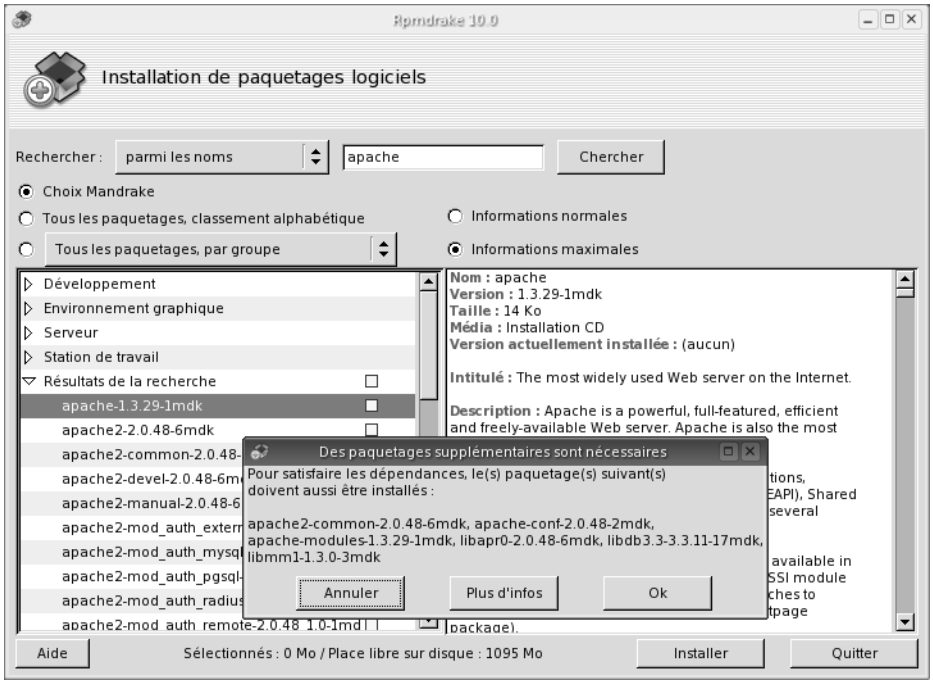


Figure A–9
Installation d'Apache avec rpmdrake sous Mandrakelinux

Installer PHP 5

Pour PHP 5, les choses sont un peu moins simples, car à ce jour aucune distribution n'intègre encore le package associé à PHP 5. La solution la plus simple consiste alors à compiler soit même.

L'opération n'est pas complexe en soi, mais faute d'habitude on peut être amené à tâtonner un peu pour retrouver les outils nécessaires à la compilation ou à l'activation des extensions PHP souhaitées. Il s'agit bien sûr des outils de développements traditionnels (compilateur, en-têtes systèmes) mais les en-têtes de librairies déjà installées seront le plus souvent nécessaires (par exemple pour GD si l'on souhaite obtenir le support des images JPEG, PNG...). Toutefois, dans la plupart des cas l'outil « `configure` » livré avec PHP saura indiquer les éléments manquants.

La procédure de compilation en elle-même est sans surprise, voici la séquence d'opération utilisée pour PHP Saloon :

```
./configure --with-sqlite --with-apxs --with-xsl --with-gd --with-zlib
--with-jpeg-dir=/usr
make
```

D'autres extensions peuvent naturellement être ajoutées. Pour plus de détail, consulter le fichier `INSTALL` ou l'aide de `configure`. Au terme de la compilation, l'installation est automatisée, il suffit d'invoquer la commande :

```
make install
```

La configuration d'Apache est automatiquement modifiée et PHP est prêt à être utilisé. Comme sous Windows, on pourra s'en assurer avec un simple `phpinfo()`. Il peut toutefois être utile d'ajouter `index.php` comme alternative à `index.html` (voir l'installation Windows).

Tester PHP Saloon

Que faire une fois ces installations réalisées ? Une toute dernière opération : copier le code de PHP Saloon à la racine de votre site web. Quelques ajustements en termes de droits sur les fichiers peuvent être nécessaires sous Linux mais, sauf complication imprévue, PHP Saloon devrait être directement opérationnel.

Pear et les extensions standards de PHP 5

annexe B

Une des richesses de PHP est de ne pas exister seul en tant que langage, mais, au contraire, de proposer en standard un environnement complet doté de multiples extensions qui couvrent l'intégralité des domaines de développement.

En outre, avec Pear, PHP dispose d'un répertoire encore plus ouvert de classes et d'extensions, à la manière de CPAN pour Perl.

Les extensions standards

Ces extensions sont intégrées à la distribution PHP. Ce qui veut dire que, sous réserve d'être activées au moment de la compilation de l'interpréteur PHP, celles-ci sont disponibles sans autre formalité.

Cependant, cette réserve est importante, car dans le cas des hébergements mutualisés, l'utilisateur ne compile pas lui-même PHP, mais utilise une version partagée par tous les clients de l'hébergeur. En fonction des choix de celui-ci, certaines extensions dites standards peuvent donc s'avérer indisponibles.

Pour éviter les désagréments après coup, une fois la commande d'un hébergement passée, il est important de se renseigner au préalable. La plupart des hébergeurs sont transparents sur leurs choix et proposent une page d'information utilisant la fonction `phpinfo()` de PHP pour lister les extensions activées.

Certaines extensions sont par nature directement intégrées à PHP, c'est le cas des opérations sur les tableaux.

Le tableau suivant liste les extensions disponibles.

Extension	Domaine d'action
apache	Quand PHP est utilisé sous forme de module pour le serveur Web Apache, cette extension fournit des fonctions permettant d'accéder à des fonctionnalités spécifiques d'Apache.
arrays	Cette extension regroupe l'intégralité des opérations sur les tableaux (fusion, tri, transformations diverses).
bcmath	BC (Binary Calculator) est une extension qui permet d'effectuer des calculs avec une précision sans limite (traditionnellement, les entiers sont stockés sur 32 bits uniquement et les réels le sont avec une précision donnée qui peut être insuffisante).
bz2	bzip2 est un utilitaire et une librairie de compression. Cette extension permet de manipuler ce type d'archives compressées.
calendar	L'histoire a connu un certain nombre de calendriers, et encore aujourd'hui, certains pays ont recours à leur propre calendrier. L'extension calendar permet d'effectuer des conversions entre tous les calendriers.
com/.NET	Les amateurs de la plate-forme Microsoft Windows pourront avec cette extension invoquer les composants COM disponibles sur leur OS préféré, voire charger des modules .NET. Attention, la version disponible dans PHP 5 n'est pas celle de la version 4.
classes/objects	Toutes les fonctions d'information sur les objets et les classes sont réunies dans cette extension.
clibpdf	Une des extensions disponibles en PHP pour créer dynamiquement et manipuler les documents PDF.
ctype	Cette extension propose des fonctions qui sauront ranger vos chaînes de caractères dans différentes catégories (numériques, majuscules et minuscules). Même si elle peut paraître redondante avec certaines instructions PHP plus connues, cette extension est en réalité très rapide.
curl	Avec fopen() et les streams, PHP est en mesure de se connecter sans difficulté à tout type de serveur distant, mais une fois la connexion effectuée, l'interprétation du protocole (HTTP, FTP ou autre) peut être complexe à réaliser manuellement. L'extension curl propose toute une série de fonctions pour simplifier le travail et ainsi accéder, le plus simplement du monde, à tout type de serveurs depuis PHP. Si vous souhaitez agréger différentes sources de contenus, cette extension peut vous être très utile.
dba	Cette extension permet d'accéder aux bases de données de type DBM stockées sous forme de fichiers.
date/time	Disponible en standard, cette extension intègre une foule de fonctions pour manipuler les dates et le temps en général.
dbase	dbase est un autre système pour stocker des enregistrements dans un fichier. Cette extension fournit l'accès à de tels fichiers.
dbx	dbx est ce qu'on appelle un *abstraction layer*, c'est-à-dire une interface commune à plusieurs systèmes, ici des bases de données. Ainsi, au lieu d'utiliser les instructions spécifiques à MySQL ou SQLite, les fonctions de l'extension dbx permettent d'attaquer indifféremment des bases des deux types. La plupart des bases compatibles avec PHP le sont également avec dbx.
dio	Cette extension permet d'effectuer des entrées et sorties à un niveau plus bas que les streams PHP comme le propose la section 6 du standard POSIX. À réserver aux experts pour des cas bien particuliers.
directories	Toutes les fonctions de manipulation des répertoires sont réunies ici.
dom	La fameuse extension DOM, dont PHP Saloon fait un large usage, et qui implémente l'ensemble de l'API définie par le W3C pour les documents HTML et XML.

Extension	Domaine d'action
errors/logging	Cette extension regroupe les fonctions qui pemettent d'identifier et de rapporter les erreurs lors de l'exécution d'un programme PHP.
exif	EXIF (Exchangeable Image File Format) est un standard pour stocker les informations essentielles d'une image (notamment JPEG). De nombreux appareils photo numériques se servent de ce standard.
fam	Disponible uniquement sous Microsoft Windows, cette extension permet de suivre les modifications de certains fichiers automatiquement.
frontbase	Une extension pour accéder aux bases FrontBase.
fdf	Cette extension donne accès à l'interface d'Adobe pour manipuler les fichiers FDF utilisés par Adobe Forms.
filepro	Une extension pour accéder aux bases FilePro en lecture uniquement.
filesystem	Disponible par défaut, cette extension regroupe toutes les fonctions d'accès aux fichiers et, plus généralement, au système de fichiers lui-même.
ftp	Cette extension permet d'accéder à un serveur FTP depuis un programme PHP.
functions	Avec cette extension, il est possible de manipuler toutes les fonctions de manière avancée.
image	GD est une bibliothèque graphique dont une version est intégrée à PHP. Elle permet de manipuler différents formats d'image et d'en créer dynamiquement via des fonctions intégrées à cette extension.
gettext	Nous l'avons utilisée dans PHP Saloon. GNU Gettext est un outil d'internationalisation.
gmp	À l'instar de BC, GMP est une bibliothèque permettant de manipuler des valeurs en précision arbitraire, mais uniquement pour les entiers, dans le cas de GMP.
http	Manipulant cookies et headers, cette extension est à même de modifier les paramètres de la connexion HTTP.
hwapi	Une extension pour accéder au système HyperWave.
iconv	Cette extension permet de disposer des fonctionnalités de conversion entre encodages pour les caractères.
imap	Contrairement à ce que le nom pourrait laisser croire, les fonctions de l'extension imap ne se limitent pas à ce seul protocole mais supportent aussi POP3, pour la relève du courrier, et NNTP, pour l'accès aux forums de discussion. La plupart des outils PHP de lecture d'e-mails en ligne reposent sur cette extension.
informix	Une extension pour accéder aux bases Informix.
ingres_ii	Une extension pour accéder aux bases Ingres.
interbase	Une extension pour accéder aux bases InterBase.
ircg	Cette extension s'interface à IRC (Internet Relay Chat, un système de chat en réseau,) via le logiciel IRCG.
ldap	Cette extension permet d'interroger et de modifier les annuaires LDAP, notamment Active Directory.
mail	Cette extension consiste en réalité en une fonction principale qui permet d'envoyer un e-mail depuis PHP.
math	Disponible par défaut, cette extension regroupe l'ensemble des fonctions et constantes mathématiques.

Extension	Domaine d'action
multibyte string	Traditionnellement, en Europe occidentale, les caractères sont stockés sur un seul octet. Toutefois, cela n'est pas possible pour différentes langues. Dans ce cas, il est fait usage de chaînes de caractères où plusieurs octets sont associés à chaque caractère. Cette extension permet de manipuler de telles chaînes.
mcrypt	Cette extension permet d'accéder aux fonctions et algorithmes de chiffrement disponibles dans la librairie mcrypt.
mcve	MCVE est le successeur de Red Hat CCVS et n'est pas disponible sous Microsoft Windows. Elle permet à n'importe quel Unix, dont Linux, d'obtenir un numéro d'autorisation bancaire pour la vente à distance sans passer par un intermédiaire.
mhash	mhash permet de calculer des signatures ou des clés et de vérifier ainsi l'authenticité de documents.
mime_magic	Les fonctions de cette extension permettent de déterminer le contenu d'un fichier à la manière de la commande file sous Unix.
ming	Ming est une bibliothèque Open Source qui permet de créer des documents Macromedia Flash.L'extension PHP permet d'accéder à ces fonctionnalités.
misc	Un ensemble de fonctions plus ou moins hétéroclites, dont les fonctions de colorisation du code PHP.
mnogosearch	Cette extension s'interface avec le moteur de recherche mnoGoSearch. Ce moteur peut être utilisé sur un intranet.
msession	Lorsqu'un site est très populaire, il peut s'avérer nécessaire d'équilibrer la charge entre plusieurs serveurs. Dans ce cas, les sessions traditionnelles de PHP ne peuvent fonctionner, car les requêtes des utilisateurs sont traitées par plusieurs serveurs. L'extension msession permet de s'interfacer avec un serveur de session et ainsi contourner le problème.
msql	Une extension pour accéder aux bases MSQL.
mssql	Une extension pour accéder aux bases Microsoft SQL Server.
mysql	L'extension probablement la plus connue de PHP. Elle permet d'accéder aux bases MySQL, quelle que soit la version du serveur.
mysqli	Cette extension permet de tirer profit des améliorations de l'interface d'interrogation, disponible dans les serveurs MySQL version 4.1 ou supérieure.
ncurses	Cette extension n'est utile qu'à ceux qui souhaitent développer des applications en mode console sous Unix. Elle permet de piloter le terminal en utilisant la célèbre librairie nCurses.
network	Cette extension intègre une très large partie des fonctions réseau disponibles sous les systèmes Unix. Pour plus de fonctionnalités, il est préférable de regarder l'extension sur les flots (streams).
oci8	Cette extension permet de se connecter aux bases Oracle (version 8 et versions supérieures).
odbc	Cette extension permet de se connecter à toute base de données disposant d'un driver ODBC.
openssl	Cette extension permet d'utiliser directement l'ensemble des possibilités de la librairie SSL OpenSSL pour crypter des connexions.
oracle	Il s'agit de l'extension historique pour se connecter à Oracle. Sauf recours à une version préhistorique d'Oracle, l'extension oci8 est à préférer.
output control	Cette extension permet de contrôler de manière précise le flot de sortie produit par l'exécution du programme PHP (incluant les éléments statiques).

Extension	Domaine d'action
overload	Cette extension définit une unique fonction overload() qui permet d'activer le traitement dynamique des attributs et des méthodes (avec les fonctions __set(), __get() et __call()).
ovrimos	Cette extension permet de se connecter aux bases Ovrimos.
pcntl	Cette extension est destinée en priorité aux applications autonomes. Elle permet de contrôler (avec des signaux) ou de créer de nouveaux processus (avec fork()). L'ensemble est proche des fonctionnalités équivalentes sous Unix/Linux.
pcre	Cette extension dote PHP de fonctions compatibles avec les expressions régulières Perl. Celles-ci permettent de rechercher des motifs dans les chaînes de caractères.
pdf	Une deuxième extension pour créer des documents PDF (fondée sur PDFlib).
pfpro	Cette extension permet de traiter les paiements par carte bancaire, à l'aide du service PayFlow de Verisign.
pgsql	Cette extension permet de se connecter aux bases PostgreSQL.
php options/infos	Cette extension regroupe l'ensemble des fonctions utilitaires, permettant d'affiner ou d'obtenir la configuration courante de PHP, notamment la fonction phpinfo().
posix	Cette extension donne accès à l'ensemble des fonctions usuelles définies dans la norme POSIX.1, bien connues sous Unix/Linux.
printer	Cette extension permet de contrôler l'impression sous Microsoft Windows.
program execution	Cette extension permet d'exécuter des programmes localement sur le serveur depuis le script PHP.
pspell	Cette extension permet de tirer profit de l'outil Aspell et ainsi de vérifier l'orthographe d'un mot et éventuellement d'obtenir des suggestions de corrections.
readline	readline est une extension simplifiant le traitement des arguments passés en ligne de commande. Cette extension est donc dédiée aux applications autonomes.
recode	L'extension recode permet de convertir des chaînes ou des fichiers d'un encodage à un autre. Il peut être utile d'utiliser iconv ou mstring.
regexps	Cette extension définit un jeu de fonctions sur des expressions régulières propres à PHP. Il peut être préférable d'utiliser pcre.
session	Cette extension regroupe l'ensemble des opérations et du support des sessions.
shmop	Cette extension permet d'utiliser les fonctionnalités de mémoire partagée et de communication interprocessus sur les systèmes Unix/Linux.
simplexml	Alternative à DOM, SimpleXML permet de manipuler simplement les documents XML sous la forme d'une hiérarchie d'objets.
snmp	SNMP (Simple Network Management Protocol) est, comme son nom l'indique, un protocole utilisé pour l'administration d'un réseau et sa supervision.
soap	Cette extension permet de créer et de gérer des échanges XML conformes à SOAP.
sockets	Cette extension permet de manipuler directement les sockets (points de connexion). Dans la plupart des cas, il est possible de se contenter des opérations sur les flots.

Extension	Domaine d'action
spl	SPL (Standard PHP Library) propose un certain nombre d'interfaces et de design patterns prêts à l'emploi dans PHP 5.
sqlite	Cette extension permet d'accéder aux bases SQLite.
streams	Cette extension regroupe l'ensemble des opérations et fonctions permettant de manipuler les flots, qu'il s'agisse de fichiers ou de connexions.
strings	Cette extension regroupe l'ensemble des fonctions standards agissant sur les chaînes de caractères.
sybase	Cette extension permet d'accéder aux bases Sybase.
sybase_ct	Cette extension permet d'accéder aux bases Sybase (variante CT).
tidy	Tidy est un analyseur de code HTML. Rigoureux, il met en évidence les non-conformités et peut reformuler le code, en tenant compte des standards à respecter.
tokenizer	Cette extension permet d'accéder au cœur de l'analyseur de code inclus dans l'interpréteur PHP. Cela permet d'analyser soi-même un code PHP de manière très simple (pour générer de la documentation ou des statistiques).
unified odbc	Cette extension n'est pas réellement une interface vers ODBC, mais une interface native vers des systèmes de gestion de bases de données dont l'interface de programmation est compatible avec ODBC. Il est toutefois possible, en plus de ces bases, d'utiliser réellement ODBC si l'on ajoute le pilote iodbc.
url	Cette extension regroupe l'ensemble des fonctions agissant sur les URL.
variables	Cette extension regroupe l'ensemble des fonctions permettant de manipuler ou d'obtenir des informations sur les variables PHP et leur contenu.
w32api	Cette extension, réservée aux systèmes sous Microsoft Windows, permet d'utiliser directement depuis PHP les fonctions standards définies dans l'interface Win32.
wddx	WDDX est un protocole qui normalise sous forme textuelle (XML) un contenu, de manière à le transporter sans déperdition sur le réseau.
xml	Cette extension permet d'analyser un document XML. Les possibilités offertes sont de plus bas niveau que celles proposées par DOM.
xmlrpc	Cette extension permet d'invoquer des procédures à distance en échangeant les données sous forme de documents XML.
xsl	Cette extension permet de réaliser les transformations XSL de documents XML depuis PHP.
yaz	Cette extension permet d'utiliser les outils yaz et d'interroger les serveurs conformes à la norme ANSI Z39.50
yp	Sous Unix/Linux, NIS (autrefois appelé YP pour *Yellow Pages*) est le système historique d'annuaires utilisables pour l'identification, la résolution des noms et des groupes d'individus (à la manière d'un *Active Directory* sous Microsoft Windows). Cette extension permet d'accéder à ces annuaires.
zlib	Cette extension permet de manipuler ou d'accéder à des documents compressés avec gzip de manière transparente avec la plupart des opérations sur les fichiers.

Pear (PHP Extension and Application Repository)

On le voit, le nombre d'extensions compatibles avec PHP est impressionnant. En réalité, le manuel en ligne de PHP en intègre davantage. Cependant l'arrivée de PHP 5 est l'occasion de rationaliser l'explosion du nombre de ces extensions.

Désormais, seul le noyau dur des extensions PHP sera inclus dans la distribution PHP, le reste résidant dans une sorte de gigantesque bibliothèque. Cette bibliothèque est Pear.

À l'origine, Pear n'est qu'un répertoire de petits modules, souvent des classes réalisées par des développeurs ou des utilisateurs de PHP et mises en commun pour la communauté des utilisateurs.

Depuis ses modestes débuts, Pear a beaucoup évolué et se transforme de plus en plus en un équivalent de CPAN, qui est la source de référence pour les modules Perl.

Pear propose donc une multitude de composants qui peuvent vous éviter de réinventer la roue. À ceci s'ajoutent donc depuis PHP 5 les extensions qui, sans être inutiles, répondent à des besoins plus spécifiques que ceux de la distribution.

Enfin, la force de Pear est de disposer d'un installeur simple d'emploi. Pour disposer d'une extension ou d'un composant, inutile de recompiler PHP, il suffit de demander, et l'installeur Pear s'occupe de tout.

Aussi, quels que soient vos objectifs, il ne faut pas hésiter à explorer la base Pear. Certes, Pear est encore en cours de mûrissement et tous les composants n'ont pas la maturité attendue, mais on aurait tort de mettre l'ensemble de Pear de côté, alors que celui-ci a fini par revêtir une importance capitale au sein de la stratégie PHP.

Document Object Model (DOM)

annexe C

La représentation des documents avec DOM

Lorsqu'un document doit être analysé chacun est libre de choisir à la fois la méthode mise en œuvre, et la façon de stocker en mémoire le résultat de l'analyse réalisée.

À l'échelle du Web cette liberté confine au chaos en matière de développement. Le W3C a donc proposé un modèle permettant de représenter et manipuler les documents XML et HTML. Ce modèle se nomme DOM.

DOM ne préjuge en rien de la méthode utilisée pour analyser un document mais se contente de définir une interface et un modèle de représentation des informations qui devra être accessible via l'interface.

Le modèle proposé par DOM repose sur une représentation arborescente de la structure des documents analysés. C'est cette représentation arborescente qui devra être manipulable via l'interface de programmation.

Cependant chaque développeur reste libre de stocker l'information comme bon lui semble (une liste chaînée par exemple) tant que l'implantation de l'interface respecte DOM et expose à l'utilisateur la représentation arborescente définie dans la recommandation. Que celle-ci colle à l'implantation ou pas n'a aucune importance pour l'utilisateur de DOM.

La recommandation officielle du W3C et le copyright associé

Dans cette annexe nous allons reprendre la spécification en IDL (Interface Definition Language) de l'API Document Object Model (DOM) Level 3 Core (Annexe F).
Ces éléments, dont certains sont traduits en français, ne sauraient être confondus avec la recommandation officielle qui demeure la seule référence, téléchargeable librement sur les sites précisés ci-dessous.

Pour chaque interface la mention suivante est applicable :

```
/*
* Copyright (c) 2004 World Wide Web Consortium,
*
* (Massachusetts Institute of Technology, European Research Consortium for
* Informatics and Mathematics, Keio University). All Rights Reserved. This
* work is distributed under the W3C(r) Software License [1] 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.
*
* [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
*/
```

Par ailleurs, le lecteur est invité à prendre connaissance du texte intégral de la licence mentionnée :

```
W3C® SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
This work (and included software, documentation such as READMEs, or other related items) is being provided by the
copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee)
agree that you have read, understood, and will comply with the following terms and conditions.
Permission to copy, modify, and distribute this software and its documentation, with or without modification, for
any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies
of the software and documentation or portions thereof, including modifications:
   1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
   2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the
W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any
redistributed or derivative code.
   3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend
you provide URIs to the location from which the code is derived.)
THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR
PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS,
TRADEMARKS OR OTHER RIGHTS.
COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF
ANY USE OF THE SOFTWARE OR DOCUMENTATION.
The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the
software without specific, written prior permission. Title to copyright in this software and any associated
documentation will at all times remain with copyright holders.
```

- http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/
- http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/idl/dom.idl

Types élémentaires

Au-delà des nœuds qui, dans une représentation arborescente vont constituer l'essentiel des objets manipulés, DOM définit quelques types élémentaires, et en réalité un type clé : `DOMString`.

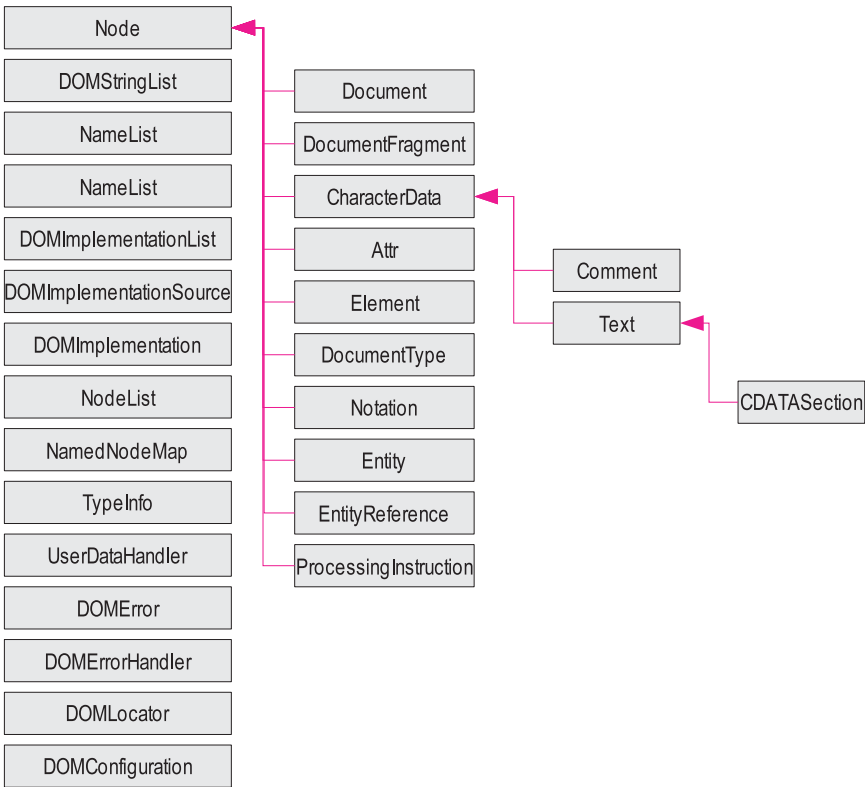
Nombre de méthodes dans l'API DOM vont manipuler des chaînes de caractères. Pour s'assurer la plus large audience, DOM utilise l'encodage UTF-16 pour chaque caractère (qui occupe donc 2 octets). Ce type de caractères est désormais très bien supporté par la plupart des langages, y compris JavaScript.

Par ailleurs DOM propose les types suivants :

- `DOMTimeStamp` pour stocker une date (absolue ou relative). Celle-ci est exprimée en millisecondes ;
- `DOMUserData` qui permet de lier une arborescence DOM à des données utilisateurs extérieures à l'arbre. Ce type de donnée représente un pointeur générique ;
- `DOMObject` pour manipuler des références d'objets DOM.

Interfaces fondamentales

DOM dispose d'implantations dans de nombreux langages, PHP naturellement mais aussi Java et JavaScript entre autres. Le modèle objet n'étant pas nécessairement supporté par tous ces langages, l'API DOM ne définit pas des objets mais des interfaces. Naturellement dans le cas des langages orientés objet, on définira le plus souvent une hiérarchie de classes cohérente avec celles des interfaces DOM.



COMPRENDRE **L'OMG IDL**

Dans la suite les interfaces de programmation DOM ne seront pas décrites en utilisant la syntaxe de PHP. En effet PHP n'étant pas un langage fortement typé, nos descriptions ne seraient pas assez précises. Toutefois le langage retenu, issu des travaux de l'OMG (Object Management Group) et de CORBA ne devrait pas poser problème.

- http://www.omg.org/technology/documents/formal/corba_2.htm

Dans l'absolu, l'interface `Node` est la plus importante et est suffisante pour tout faire. Cependant, dans la pratique on utilisera des interfaces plus sophistiquées en fonction de la nature du nœud manipulé comme `Element` ou `Document`.

Enfin, il faut noter que DOM est aujourd'hui disponible dans sa troisième mouture, certains attributs ou méthodes sont donc apparus au fil du temps, dans les versions deux ou trois, ces évolutions sont précisées dans les interfaces qui suivent.

Pour plus de détails

Cette annexe reprend les interfaces de l'API DOM de manière succinte. Dans la plupart des cas, les noms des méthodes et des attributs sont caractéristiques du rôle de chacun et se suffisent à eux-mêmes.
Toutefois, certaines interfaces peuvent nécessiter des explications approfondies. Le plus sage est alors de revenir à la recommandation qui explicite pour chaque attribut ou méthode le rôle et les contraintes.

- http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

DOMException

En cas d'échec, lors de situations exceptionnelles, les méthodes de l'API DOM peuvent être amenées à déclencher une exception. DOM ne définit qu'une exception : `DOMExeption` (en PHP cette classe dérive naturellement de la classe `Exception`).

Toutefois l'implantation est libre de déclencher des erreurs additionnelles. En outre, les langages non orientés objet peuvent se contenter d'utiliser le mécanisme traditionnel de traitement des erreurs à base de code de retour.

```
exception DOMException {
  unsigned short   code;
};
```

Tableau C–1 Constantes définies dans DOM (niveau 1)

Nom	Valeur	Signification
INDEX_SIZE_ERR	1	Index ou intervalle invalide.
DOMSTRING_SIZE_ERR	2	Le texte proposé n'est pas stockable dans la chaîne.
HIERARCHY_REQUEST_ERR	3	Nœud impossible à insérer à cet endroit.
WRONG_DOCUMENT_ERR	4	Nœud utilisé dans un document incompatible.
INVALID_CHARACTER_ERR	5	Caractère invalide (par exemple dans un nom d'élément).
NO_DATA_ALLOWED_ERR	6	Aucune donnée ne peut être insérée.
NO_MODIFICATION_ALLOWED_ERR	7	Modification interdite
NOT_FOUND_ERR	8	Nœud introuvable dans ce contexte.
NOT_SUPPORTED_ERR	9	Fonctionnalité non prise en charge par l'implantation.
INUSE_ATTRIBUTE_ERR	10	Ajout d'un attribut déjà associé à un élément par ailleurs.

Tableau C–2 Constantes définies dans DOM (niveau 2)

Nom	Valeur	Signification
INVALID_STATE_ERR	11	L'objet n'est plus utilisable.
SYNTAX_ERR	12	La chaîne manipulée est erronnée.
INVALID_MODIFICATION_ERR	13	Modification du type de l'objet interdite.
NAMESPACE_ERR	14	Modification apportée à l'objet non conforme aux contraintes de l'espace de noms.
INVALID_ACCESS_ERR	15	Paramètre ou méthode non pris en charge.

Tableau C–3 Constantes définies dans DOM (niveau 3)

Nom	Valeur	Signification
VALIDATION_ERR	16	Le nœud est invalide apres utilisation de `insertBefore()` ou `removeChild()`.
TYPE_MISMATCH_ERR	17	Type d'objet non conforme à celui attendu.

DOMStringList– DOM③

Cette interface décrit la manipulation d'une liste de chaînes de caractères. La taille de la liste est disponible via l'attribut `length` et chaque chaîne peut être obtenue individuellement avec la fonction `item()`.

```
interface DOMStringList {
readonlyattribute unsigned long   length;
  DOMString          item     (in unsigned long index);
  boolean            contains (in DOMString str);
};
```

NameList– DOM③

L'utilisation de listes dans DOM est assez fréquente, en PHP cela peut paraître déroutant tant la notion de tableau est riche. Ici il s'agit de manipuler une liste ordonnée de couples : nom , espace de noms.

```
interface NameList {
  readonly attribute unsigned long   length;
  DOMString          getName         (in unsigned long index);
  DOMString          getNamespaceURI (in unsigned long index);
  boolean            contains        (in DOMString str);
  boolean            containsNS      (in DOMString namespaceURI,
                                      in DOMString name);
};
```

DOMImplementation

L'interface `DOMImplementation` permet de créer ou de manipuler des informations transversales indépendantes des documents manipulés par ailleurs.

En outre, l'interface `DOMImplementation` permet de s'assurer de la disponibilité de certaines fonctionnalités dans l'implantation DOM utilisée.

```
interface DOMImplementation {
  boolean            hasFeature        (in DOMString feature,
                                        in DOMString version);

  // Méthodes disponibles à compter de DOM niveau 2 :
  DocumentType       createDocumentType(in DOMString qualifiedName,
                                        in DOMString publicId,
                                        in DOMString systemId) raises(DOMException);
  Document           createDocument    (in DOMString namespaceURI,
                                        in DOMString qualifiedName,
                                        in DocumentType doctype) raises(DOMException);
  // Méthodes disponibles à compter de DOM niveau 3 :
  DOMObject          getFeature        (in DOMString feature,
                                        in DOMString version);
};
```

DOMImplementationSource

Dans le prolongement de l'interface `DOMImplementation`, `DOMImplementationSource` permet d'obtenir une ou plusieurs implantations compatibles avec certaines fonctionnalités minimales.

```
interface DOMImplementationSource {
  DOMImplementation     getDOMImplementation     (in DOMString features);
  DOMImplementationList getDOMImplementationList (in DOMString features);
};
```

DOMImplementationList

Encore une liste, mais consacrée aux objets `DOMImplementation`, et notamment utilisée dans le cadre de `DOMImplementationSource`.

```
interface DOMImplementationList {
  readonly attribute unsigned long   length;
  DOMImplementation item(in unsigned long index);
};
```

Node

L'interface `Node` est naturellement l'interface fondamentale de l'API DOM. Elle fournit les fonctions essentielles pour parcourir et manipuler l'arbre représentant chaque document analysé.

```
interface Node {
  readonly attribute DOMString       nodeName;
           attribute DOMString       nodeValue;
                                     // l'affectation et la lecture peuvent lever une
                                     // exception DOMException
```

```
  readonly attribute unsigned short nodeType;
  readonly attribute Node            parentNode;
  readonly attribute NodeList        childNodes;
  readonly attribute Node            firstChild;
  readonly attribute Node            lastChild;
  readonly attribute Node            previousSibling;
  readonly attribute Node            nextSibling;
  readonly attribute NamedNodeMap    attributes;
  // Attributs modifiés pour DOM niveau 2
  readonly attribute Document        ownerDocument;
  // Attributs disponibles compter de DOM niveau 2
  readonly attribute DOMString       namespaceURI;
           attribute DOMString       prefix;
                                     // l'affectation peut lever une exception DOMException
  readonly attribute DOMString       localName;
  // Attributs disponibles compter de DOM niveau 3
  readonly attribute DOMString       baseURI;
           attribute DOMString       textContent;
                                     // l'affectation et la lecture peuvent lever une
                                     // exception DOMException
  boolean            hasChildNodes           ();
  Node               cloneNode               (in boolean deep);
  // Méthodes modifiées pour DOM niveau 3
  Node               insertBefore            (in Node newChild,
                                              in Node refChild) raises(DOMException);
  Node               replaceChild            (in Node newChild,
                                              in Node oldChild) raises(DOMException);
  Node               removeChild             (in Node oldChild) raises(DOMException);
  Node               appendChild             (in Node newChild) raises(DOMException);
  void               normalize               ();
  // Méthodes disponibles à compter de DOM niveau 2
  boolean            isSupported             (in DOMString feature,
                                              in DOMString version);
  boolean            hasAttributes           ();

  // Méthodes disponibles à compter de DOM niveau 3
  unsigned short     compareDocumentPosition (in Node other) raises(DOMException);
  boolean            isSameNode              (in Node other);
  DOMString          lookupPrefix            (in DOMString namespaceURI);
  boolean            isDefaultNamespace      (in DOMString namespaceURI);
  DOMString          lookupNamespaceURI      (in DOMString prefix);
  boolean            isEqualNode             (in Node arg);
  DOMObject          getFeature              (in DOMString feature,
                                              in DOMString version);
  DOMUserData        setUserData             (in DOMString key,
                                              in DOMUserData data,
                                              in UserDataHandler handler);
  DOMUserData        getUserData             (in DOMString key);
};
```

Tableau C–4 Constantes associées aux différents types de nœuds DOM

Type	Code associé
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11
NOTATION_NODE	12

Tableau C–5

Nom	Valeur associée
DOCUMENT_POSITION_DISCONNECTED	0x01
DOCUMENT_POSITION_PRECEDING	0x02
DOCUMENT_POSITION_FOLLOWING	0x04
DOCUMENT_POSITION_CONTAINS	0x08
DOCUMENT_POSITION_CONTAINED_BY	0x10
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC	0x20

NodeList

`NodeList` définit une liste ordonnée de nœuds. Ce type de liste est souvent utilisée comme valeur de retour par les méthodes DOM.

```
interface NodeList {
  Node              item(in unsigned long index);
  readonly attribute unsigned long   length;
};
```

Document

Autre interface fondamentale de DOM, `Document` étend l'interface `Node` pour proposer des méthodes de plus au niveau (comme la création d'éléments ou de sections `Text`).

```
interface Document : Node {
  readonly attribute DOMImplementation implementation;
  readonly attribute Element           documentElement;
  // Attributs modifiés pour DOM niveau 3
  readonly attribute DocumentType    doctype;
  // Attributs disponibles à compter de DOM niveau 3
  readonly attribute DOMString       inputEncoding;
  readonly attribute DOMString       xmlEncoding;
           attribute boolean         xmlStandalone;
                                     // l'affectation peut lever une exception DOMException
           attribute DOMString       xmlVersion;
                                     // l'affectation peut lever une exception DOMException
           attribute boolean         strictErrorChecking;
           attribute DOMString       documentURI;
  readonly attribute DOMConfiguration domConfig;
  Element            createElement                (in DOMString tagName) raises(DOMException);
  DocumentFragment   createDocumentFragment       ();
  Text               createTextNode               (in DOMString data);
  Comment            createComment                (in DOMString data);
  CDATASection       createCDATASection           (in DOMString data) raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                                     in DOMString data) raises(DOMException);
  Attr               createAttribute              (in DOMString name) raises(DOMException);
  EntityReference    createEntityReference        (in DOMString name) raises(DOMException);
  NodeList           getElementsByTagName         (in DOMString tagname);
  // Méthodes disponibles à compter de DOM niveau 2
  Node               importNode                   (in Node importedNode,
                                                     in boolean deep) raises(DOMException);
  Element            createElementNS              (in DOMString namespaceURI,
                                                     in DOMString qualifiedName) raises(DOMException);
  Attr               createAttributeNS            (in DOMString namespaceURI,
                                                     in DOMString qualifiedName) raises(DOMException);
  NodeList           getElementsByTagNameNS       (in DOMString namespaceURI,
                                                     in DOMString localName);
  Element            getElementById               (in DOMString elementId);
  // Méthodes disponibles à compter de DOM niveau 3 :
  Node               adoptNode                    (in Node source) raises(DOMException);
  void               normalizeDocument            ();
  Node               renameNode                   (in Node n,
                                                     in DOMString namespaceURI,
                                                     in DOMString qualifiedName) raises(DOMException);
};
```

DocumentFragment

L'interface `DocumentFragment` permet de manipuler des morceaux de document. Il serait bien sûr possible d'utiliser l'interface `Document` classique. Cependant, en fonction de l'implantation, celle-ci peut s'avérer gourmande en ressources.

DocumentFragrent laisse donc à l'implantation la possibilité de proposer une version allégée, utilisée spécifiquement pour composer des morceaux de documents entre eux.

Cette interface ne définit aucune méthode complémentaire à celles définies dans l'interface Node.

```
interface DocumentFragment : Node {
};
```

NamedNodeMap

Outre les listes, qui permettent d'accéder à des collections d'objets ordonnés, l'interface NamedNodeMap permet de manipuler une collection de nœuds en proposant un accès par leur nom (y compris l'espace de noms).

```
interface NamedNodeMap {
  readonly attribute unsigned long   length;
  Node               getNamedItem      (in DOMString name);
  Node               setNamedItem      (in Node arg) raises(DOMException);
  Node               removeNamedItem   (in DOMString name) raises(DOMException);
  Node               item              (in unsigned long index);
  // Méthodes disponibles à compter de DOM niveau 2
  Node               getNamedItemNS    (in DOMString namespaceURI,
                                        in DOMString localName) raises(DOMException);
  Node               setNamedItemNS    (in Node arg) raises(DOMException);
  Node               removeNamedItemNS (in DOMString namespaceURI,
                                        in DOMString localName) raises(DOMException);
};
```

CharacterData

CharacterData represente une interface qu'on pourrait qualifier d'abstraite. En effet, dans un document DOM aucun nœud de ce type n'existe. Par contre, cette interface sera complétée de manière à définir les interfaces Text et Comment notamment.

```
interface CharacterData : Node {
           attribute DOMString       data;
                                     // l'affectation et la lecture peuvent lever une
                                     // exception DOMException
  readonly attribute unsigned long   length;
  DOMString          substringData   (in unsigned long offset,
                                      in unsigned long count) raises(DOMException);
  void               appendData      (in DOMString arg) raises(DOMException);
  void               insertData      (in unsigned long offset,
                                      in DOMString arg) raises(DOMException);
  void               deleteData      (in unsigned long offset,
                                      in unsigned long count) raises(DOMException);
  void               replaceData     (in unsigned long offset,
                                      in unsigned long count,
                                      in DOMString arg) raises(DOMException);
};
```

Attr

Cette interface permet de manipuler les propriétés d'un nœud correspondant à un attribut dans le document original.

```
interface Attr : Node {
  readonly attribute DOMString       name;
  readonly attribute boolean         specified;
           attribute DOMString       value;
                                     // l'affectation peut lever une exception DOMException
  // Attributs disponibles à partir de DOM niveau 2
  readonly attribute Element         ownerElement;
  // Attributs disponibles à partir de DOM niveau 3
  readonly attribute TypeInfo        schemaTypeInfo;
  readonly attribute boolean         isId;
};
```

Element

Cette interface étend l'interface `Node` de manière à proposer des fonctions de plus haut niveau pour les nœuds qui s'avèrent être des éléments dans le document original.

```
interface Element : Node {
  readonly attribute DOMString       tagName;
  DOMString          getAttribute(in DOMString name);
  // Attributs disponibles à partir de DOM niveau 3
  readonly attribute TypeInfo        schemaTypeInfo;
  void               setAttribute         (in DOMString name,
                                           in DOMString value) raises(DOMException);
  void               removeAttribute      (in DOMString name) raises(DOMException);
  Attr               getAttributeNode     (in DOMString name);
  Attr               setAttributeNode     (in Attr newAttr) raises(DOMException);
  Attr               removeAttributeNode (in Attr oldAttr) raises(DOMException);
  NodeList           getElementsByTagName (in DOMString name);
  // Méthodes disponibles à compter de DOM niveau 2
  DOMString          getAttributeNS        (in DOMString namespaceURI,
                                            in DOMString localName) raises(DOMException);
  void               setAttributeNS        (in DOMString namespaceURI,
                                            in DOMString qualifiedName,
                                            in DOMString value) raises(DOMException);
  void               removeAttributeNS     (in DOMString namespaceURI,
                                            in DOMString localName) raises(DOMException);
  Attr               getAttributeNodeNS    (in DOMString namespaceURI,
                                            in DOMString localName) raises(DOMException);
  Attr               setAttributeNodeNS    (in Attr newAttr) raises(DOMException);
  NodeList           getElementsByTagNameNS(in DOMString namespaceURI,
                                            in DOMString localName) raises(DOMException);
  boolean            hasAttribute          (in DOMString name);
  boolean            hasAttributeNS        (in DOMString namespaceURI,
                                            in DOMString localName) raises(DOMException);
```

```
  // Méthodes disponibles à compter de DOM niveau 3
  void               setIdAttribute        (in DOMString name,
                                            in boolean isId) raises(DOMException);
  void               setIdAttributeNS      (in DOMString namespaceURI,
                                            in DOMString localName,
                                            in boolean isId) raises(DOMException);
  void               setIdAttributeNode    (in Attr idAttr,
                                            in boolean isId) raises(DOMException);
};
```

Text

Cette interface permet de manipuler les contenus texte des éléments d'un document.

```
interface Text : CharacterData {
  // Attributs disponibles à partir de DOM niveau 3
  readonly attribute boolean         isElementContentWhitespace;
  readonly attribute DOMString       wholeText;
  // Méthodes disponibles à compter de DOM niveau 3 :
  Text               splitText        (in unsigned long offset) raises(DOMException);
  Text               replaceWholeText (in DOMString content) raises(DOMException);
};
```

Comment

```
interface Comment : CharacterData {
};
```

TypeInfo– DOM③

L'interface `TypeInfo` permet de manipuler les types associés aux attributs ou éléments lorsqu'un schéma est associé au document analysé.

```
interface TypeInfo {
  readonly attribute DOMString       typeName;
  readonly attribute DOMString       typeNamespace;
  boolean            isDerivedFrom(in DOMString typeNamespaceArg,
                                   in DOMString typeNameArg,
                                   in unsigned long derivationMethod);
};
```

Nom	Valeur
DERIVATION_RESTRICTION	0x00000001
DERIVATION_EXTENSION	0x00000002
DERIVATION_UNION	0x00000004
DERIVATION_LIST	0x00000008

UserDataHandler– DOM③

On a vu avec le type `DOMUserData` que l'API DOM permet d'associer à un nœud une référence vers des informations externes. Cette interface permet de définir un handler qui sera invoqué en cas de clonage, renommage ou importantion et plus généralement une modification sur un nœud comportant une donnée utilisateur.

```
interface UserDataHandler {
  void               handle(in unsigned short operation,
                            in DOMString key,
                            in DOMUserData data,
                            in Node src,
                            in Node dst);
};
```

Tableau C–6 Types d'opération définis

Nom	Valeur
NODE_CLONED	1
NODE_IMPORTED	2
NODE_DELETED	3
NODE_RENAMED	4
NODE_ADOPTED	5

DOMError– DOM③

Cette interface décrit une erreur au sein de l'API DOM.

```
interface DOMError {
  readonly attribute unsigned short severity;
  readonly attribute DOMString       message;
  readonly attribute DOMString       type;
  readonly attribute DOMObject       relatedException;
  readonly attribute DOMObject       relatedData;
  readonly attribute DOMLocator      location;
};
```

Tableau C–7 Niveaux d'erreur définis

Nom	Valeur
SEVERITY_WARNING	1;
SEVERITY_ERROR	2;
SEVERITY_FATAL_ERROR	3;

DOMErrorHandler– DOM③

Cette interface décrit un handler qui pourra être appelé automatiquement par l'implantation DOM en cas d'erreur.

```
interface DOMErrorHandler {
  boolean            handleError(in DOMError error);
};
```

DOMLocator– DOM③

Toujours dans le cadre du traitement des erreurs cette interface permet de déterminer une position dans un document, notamment le lieu d'une erreur.

```
interface DOMLocator {
  readonly attribute long            lineNumber;
  readonly attribute long            columnNumber;
  readonly attribute long            byteOffset;
  readonly attribute long            utf16Offset;
  readonly attribute Node            relatedNode;
  readonly attribute DOMString       uri;
};
```

DOMConfiguration– DOM③

`DOMConfiguration` définit une interface permettant d'obtenir et de modifier les paramètres d'analyse propres à un document donné.

```
interface DOMConfiguration {
  readonly attribute DOMStringList   parameterNames;
  void               setParameter    (in DOMString name,
                                      in DOMUserData value) raises(DOMException);
  DOMUserData        getParameter    (in DOMString name) raises(DOMException);
  boolean            canSetParameter (in DOMString name,
                                      in DOMUserData value);
};
```

Interfaces étendues pour XML

L'API DOM est utilisable tant pour les documents HTML que les documents XML. Les interfaces suivantes permettent de manipuler plus précisément des nœuds spécifiques aux documents XML.

CDATASection

Une section CDATA dans un document XML permet de manipuler du texte qui ne respecte pas la syntaxe XML (par exemple un extrait de code ou du HTML mal formé).

```
interface CDATASection : Text {
};
```

DocumentType

Cette interface permet de manipuler la définition de type d'un document XML.

```
interface DocumentType : Node {
  readonly attribute DOMString       name;
  readonly attribute NamedNodeMap    entities;
  readonly attribute NamedNodeMap    notations;
  // Attributs disponibles à partir de DOM niveau 2
  readonly attribute DOMString       publicId;
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       internalSubset;
};
```

Notation

Cette interface est utilisée pour obtenir la notation associée à une DTD.

```
interface Notation : Node {
  readonly attribute DOMString       publicId;
  readonly attribute DOMString       systemId;
};
```

Entity

Cette interface permet d'obtenir le détail d'une entité dans un document (par exemple `&` ou `&acute;`).

```
interface Entity : Node {
  readonly attribute DOMString       publicId;
  readonly attribute DOMString       systemId;
  readonly attribute DOMString       notationName;
  // Attributs disponibles à partir de DOM niveau 3
  readonly attribute DOMString       inputEncoding;
  readonly attribute DOMString       xmlEncoding;
  readonly attribute DOMString       xmlVersion;
};
```

EntityReference

```
interface EntityReference : Node {
};
```

ProcessingInstruction

Cette interface permet d'obtenir ou de modifier le code d'une instruction de traitement. Ainsi cette interface pourrait permettre d'obtenir le code PHP enfoui dans un document HTML ou XML.

```
interface ProcessingInstruction : Node {
  readonly attribute DOMString       target;
           attribute DOMString       data;
                                     // l'affectation peut lever une exception DOMException
};
```

Organisation du code de PHP Saloon

annexe D

Pour simplifier la compréhension du code de PHP Saloon, celui-ci est organisé de la manière la plus simple possible autour de pôles homogènes. Ainsi l'ensemble du code associé au rendu de l'application a été placé dans le répertoire `style`.

L'arborescence retenue est décrite ci-dessous.

Répertoire	Rôle
`.`	Racine du code de PHP Saloon.
`./img`	L'ensemble des images utilisées dans PHP Saloon sont déposées dans ce répertoire.
`./img/connectes`	Photos des connectés.
`./inc`	Ensemble des composants de PHP Saloon et notamment les classes communes.
`./inc/i`	Interfaces des classes utilisées.
`./style`	Ce répertoire regroupe l'ensemble des feuilles de styles utilisées pour le rendu du code XML.
`./style/xul`	Transformations XSL et éléments XUL pour le rendu Mozilla/Gecko.
`./style/chtml`	Transformations XSL et code cHTML pour la plate-forme i-mode.
`./style/chtml/page`	
`./style/chtml/form`	
`./style/html`	Transformations XSL et code HTML pour les navigateurs standards.
`./style/html/form`	
`./style/html/page`	
`./xml`	Modèles XML utilisés notamment pour l'inscription et l'identification.
`./sql`	Code SQL permettant de créer le schéma initial de la base.

L'intégralité du code est disponible sur le site www.stephanemariel.com et l'application PHP Saloon est utilisable en ligne sur le site www.phpsaloon.com.

Index

Programmez intelligent avec les Cahiers du Programmeur

PHP 5

De la conception à l'exploitation, on créera une application de discussion en ligne en PHP 5 respectant les méthodes éprouvées du développement web : architecture MVC, programmation objet, conception modulaire avec les interfaces, sessions, gestion d'erreurs et exceptions, échanges XML et transformations avec DOM, XPath et SimpleXML.

On verra dans ce cahier comment concevoir le code d'une application web en utilisant les interfaces, comment prototyper un modèle de données dans SQLite, internationaliser un site dynamique grâce à Apache et PHP, générer des rendus en fonction du client (XUL/Mozilla, i-mode) avec les transformations XSLT, optimiser le code et les performances par les inclusions et compressions à la volée…

Diplômé du 3e cycle et ingénieur EPITA, **Stéphane Mariel** est un ancien enseignant chercheur de l'Institut National des Télécommunications (INT). Dirigeant/fondateur d'une société de services spécialisée dans les logiciels libres et les TIC pendant plusieurs années, il est aujourd'hui expert indépendant auprès des décideurs institutionnels et privés.

Téléchargez et testez en ligne le code source de l'étude de cas !
www.editions-eyrolles.com • www.phpsaloon.com

Sommaire

illustration Widbuck.com

EYROLLES