

La programmation WEB en PHP

Introduction à la programmation web en PHP

Programmation PHP

Rappelons ici que PHP est un langage à part entière et que s'il est surtout utilisé dans le cadre du développement d'applications pour le web, il peut être utilisé dans d'autres contextes. Le document "**PHP par l'exemple**" disponible à l'URL <http://shiva.istia.univ-angers.fr/~tahe/ressources/php.html> donne les bases du langage. On suppose ici que celles-ci sont acquises. Montrons avec un exemple simple la procédure d'exécution d'un programme PHP sous windows. Le code suivant a été sauvegardé sous le nom *coucou.php*.

```
<?php
echo "coucou";
?>
```

L'exécution de ce programme se fait dans une fenêtre Dos de Windows :

```
dos>"e:\program files\easypHP\php\php.exe" coucou.php
X-Powered-By: PHP/4.3.0-dev
Content-type: text/html

coucou
```

L'interpréteur PHP est **php.exe** et se trouve normalement dans le répertoire <php> d'installation du logiciel. On remarquera qu'ici l'interpréteur PHP envoie par défaut :

- les entêtes *HTTP X-Powered-By* et *Content-type* :
- la ligne vide séparant les entêtes HTTP du reste du document
- le document formé ici du texte écrit par la fonction *echo*

Le fichier de configuration de l'interpréteur PHP

Le comportement de l'interpréteur PHP est paramétré par un fichier de configuration appelé *php.ini* et rangé, sous windows, dans le répertoire de windows lui-même. C'est un fichier de taille conséquente puisque sous windows et pour la version 4.2 de PHP, il fait près de 1000 lignes dont heureusement les trois-quarts sont des commentaires. Examinons quelques-uns des attributs de configuration de PHP :

<code>short_open_tag = On</code>	permet d'inclure des instructions entre des balises <? >. A <i>off</i> , il faudrait les inclure entre <?php ... >
<code>asp_tags = Off</code>	à <i>on</i> permet d'utiliser la syntaxe <%=variable%> utilisée par la technologie asp (Active Server Pages)
<code>expose_php = On</code>	permet l'envoi de l'entête HTTP <i>X-Powered-By: PHP/4.3.0-dev</i> . A <i>off</i> , cet entête est supprimé.
<code>error_reporting = E_ALL & E_NOTICE</code>	fixe l'étendue du suivi d'erreurs. Ici toutes les erreurs (<i>E_ALL</i>) sauf les avertissements en cours d'exécution (<i>E_NOTICE</i>) seront signalées
<code>display_errors = Off</code>	à <i>on</i> , place les erreurs dans le flux HTML envoyé au client. Celles-ci sont donc visualisées dans le navigateur. Il est conseillé de mettre cette option à <i>off</i> .
<code>log_errors = On</code>	les erreurs seront mémorisées dans un fichier
<code>track_errors = On</code>	mémorise la dernière erreur survenue dans la variable <i>\$php_errormsg</i>
<code>error_log = E:\Program Files\EasyPHP\php\erreurs.log</code>	fixe le fichier de mémorisation des erreurs (si <i>log_errors=on</i>)
<code>register_globals = Off</code>	à <i>on</i> , un certain nombre de variables deviennent globales. Considéré comme un trou de sécurité.
<code>default_mimetype = "text/html"</code>	génère par défaut l'entête HTTP : <i>Content-type: text/html</i>
<code>include_path = ".:E:\Program Files\EasyPHP\php\pear"</code>	la liste des répertoires qui seront explorés à la recherche des fichiers requis par des directives <i>include</i> ou <i>require</i>

`session.save_path = /temp` le répertoire où seront sauvegardés les fichiers mémorisant les différentes sessions en cours.
Le disque concerné est celui où a été installé PHP. Ici */temp* désigne *e:\temp*

Ce fichier de configuration influe sur la portabilité du programme php écrit. En effet, si une application web doit récupérer la valeur d'un champ **C** d'un formulaire web, elle pourra le faire de diverses façons selon que la variable de configuration **register_globals** a la valeur *on* ou *off*:

- **off** : la valeur sera récupérée par `$HTTP_GET_VARS["C"]` ou `$_GET["C"]` ou `$HTTP_POST_VARS["C"]` ou `$_POST["C"]` selon la méthode (GET/POST) utilisée par le client pour envoyer les valeurs du formulaire
- **on** : idem ci-dessus plus `$C`, car la valeur du champ **C** a été rendue globale dans une variable portant le même nom que le champ

Si un développeur écrit un programme en utilisant la notation `$C` parce que le serveur web/php qu'il utilise a la variable *register_globals* à *on*, ce programme ne fonctionnera plus s'il est porté sur un serveur web/php où cette même variable est à *off*. On cherchera donc à écrire des programmes PHP, en évitant d'utiliser des fonctionnalités dépendant de la configuration du serveur web/php.

Configurer PHP à l'exécution

Pour améliorer la portabilité d'un programme PHP, on peut fixer soi-même certaines des variables de configuration de PHP. Celles-ci sont modifiées le temps de l'exécution du programme et seulement pour lui. Deux fonctions sont utiles dans ce processus :

`ini_get("confVariable")` rend la valeur de la variable de configuration *confVariable*
`ini_set("confVariable", "valeur")` fixe la valeur de la variable de configuration *confVariable*

Voici un exemple où on fixe la valeur de la variable de configuration *track_errors* :

```
<?php
// valeur de la variable de configuration track_errors echo
"track_errors=". ini_get("track_errors"). "¶n";
// changement de cette valeur
ini_set("track_errors","off");
// vérification
echo "track_errors=". ini_get("track_errors"). "¶n";
?>
```

A l'exécution, on a les résultats suivants :

```
Edos>"E:\Program Files\EasyPHP\php\php.exe" confl.php
Content-type: text/html

track_errors=1
track_errors=off
```

La valeur de la variable de configuration *track_errors* était initialement 1 (*~on*). On la fait passer à *off*. On retiendra que si notre application doit s'appuyer sur certaines valeurs des variables de configuration, il est prudent d'initialiser celles-ci dans le programme lui-même.

Contexte d'exécution des exemples

Les exemples de ce polycopié seront exécutés avec la configuration suivante :

- PC sous Windows 2000
- serveur Apache 1.3
- PHP 4.3

La configuration du serveur Apache est faite dans le fichier **httpd.conf**. Les lignes suivantes indiquent à Apache de charger PHP comme module intégré à Apache et de passer à l'interpréteur PHP toute requête visant un document ayant certains suffixes dont *.php*. C'est le suffixe par défaut que nous utiliserons pour nos programmes php.

```
LoadModule php4_module "E:/Program Files/EasyPHP/php/php4apache.dll" AddModule
mod_php4.c
```

```
AddType application/x-httpd-php .phtml .pwm1 .php3 .php4 .php .php2. inc
```

Par ailleurs, nous avons défini pour Apache, un alias **poly** :

```
Alias "/poly/" "e:/data/serge/web/php/poly/"
<Directory "e:/data/serge/web/php/poly"> Options Indexes
    FollowSymLinks Includes AllowOverride All
    #Order allow, deny
    Allow from all
</Directory>
```

Appelons `<poly>` le chemin `e:/data/serge/web/php/poly`. Si nous voulons demander avec un navigateur le document `doc.php` au serveur Apache, nous utiliserons l'URL `http://localhost/poly/doc.php`. Le serveur Apache reconnaîtra dans l'URL l'alias `poly` et associera alors l'URL `/poly/doc.php` au document `<poly>\doc.php`.

Unpremier exemple

Écrivons une première application web/php. Le texte suivant est enregistré dans le fichier `intro\heure.php` :

```
<html>
<head>
  <title>Une page php dynamique</title>
</head>
<body>
  <center>
    <h1>Une page PHP générée dynamiquement</h1>
    <h2>
      <?php
        $maintenant=time();
        echo date("j/m/y, h:i:s",$maintenant);
      ?>
    </h2>
    <br>
    A chaque fois que vous rafraîchissez la page, l'heure change.
  </body>
</html>
```

Si nous demandons cette page avec un navigateur, nous obtenons le résultat suivant :



La partie dynamique de la page a été générée par du code PHP :

```
<?php
  $maintenant=time();
  echo date("j/m/y, h:i:s",$maintenant);
?>
```

Que s'est-il passé exactement ? Le navigateur a demandé l'URL `http://localhost/poly/intro/heure.php`. Le serveur web (dans l'exemple Apache) a reçu cette demande et a détecté, à cause du suffixe **.php** du document demandé, qu'il devait faire suivre cette demande à l'interpréteur PHP. Celui-ci analyse alors le document `heure.php` et exécute toutes les portions de code situées entre les balises **<?php**

> et remplace chacune d'elles par les lignes écrites par les instructions PHP *echo* ou *print*. Ainsi l'interpréteur PHP va exécuter la portion de code ci-dessus et la remplacer par la ligne écrite par l'instruction *echo* :

2/10/02, 06:13:47

Une fois que toutes les portions de code PHP ont été exécutées, le document PHP est devenu un simple document HTML qui est alors envoyé au client.

On cherchera à éviter au maximum de mélanger code PHP et code HTML. Dans ce but, on pourrait réécrire l'application précédente de la façon suivante dans *intro\heure2.php* :

```
<!-- code PHP -->
<?php
// on récupère l'heure du moment
$maintenant=time();
$maintenant=date("j/m/y, h:i:s", $maintenant);
?>
<!-- code HTML -->
<html>
<head>
<title>Une page php dynamique</title>
</head>
<body>
<center>
<h1>Une page PHP générée dynamiquement</h1>
<h2>
<?php echo $maintenant ?>
</h2>
<br>
A chaque fois que vous rafraîchissez la page, l'heure change.
</body>
</html>
```

Le résultat obtenu dans le navigateur est identique :



La seconde version est meilleure que la première car il y a moins de code PHP dans le code HTML. La structure de la page est ainsi plus visible. On peut aller plus loin en mettant le code PHP et le code HTML dans deux fichiers différents. Le code PHP est stocké dans le fichier *intro\heure3.php* :

```
<!-- code PHP -->
<?php
// on récupère l'heure du moment
$maintenant=time();
$maintenant=date("j/m/y, h:i:s", $maintenant);
// on affiche la réponse include
"heure3-page1.php";
?>
```

Le code HTML est lui stocké dans le fichier *intro\heure3-page1.php* :

```
<!-- code HTML -->
<html>
<head>
```

```

<title>Une page php dynamique</title>
</head>
<body>
  <center>
    <h1>Une page PHP générée dynamiquement</h1>
    <h2>
      <?php echo $maintenant ?>
    </h2>
    <br>
    A chaque fois que vous rafraîchissez la page, l'heure change.
  </body>
</html>

```

Lorsque le navigateur demandera le document *heure3.php*, celui sera chargé et analysé par l'interpréteur PHP. A la rencontre de la ligne

```
include "heure3-page1.php";
```

l'interpréteur va inclure le fichier *heure3-page1.php* dans le code source de *heure3.php* et l'exécuter. Ainsi tout se passe comme si on avait le code PHP suivant :

```

<!-- code PHP -->
<?php
  // on récupère l'heure du moment
  $maintenant=time();
  $maintenant=date("j/m/y, h:i:s", $maintenant);
?>
<!-- code HTML -->
<html>
  <head>
    <title>Une page php dynamique</title>
  </head>
  <body>
    <center>
      <h1>Une page PHP générée dynamiquement</h1>
      <h2>
        <?php echo $maintenant ?>
      </h2>
      <br>
      A chaque fois que vous rafraîchissez la page, l'heure change.
    </body>
  </html>

```

Le résultat obtenu est le même qu'auparavant :



La solution dans laquelle les codes PHP et HTML sont dans des fichiers différents sera adoptée par la suite. Elle présente divers avantages :

- la structure des pages envoyées au client n'est pas noyée dans du code PHP. Ainsi elles peuvent être maintenues par un "web designer" ayant des compétences graphiques mais peu de compétences en PHP.
- le code PHP sert de "frontal" aux requêtes des clients. Il a pour but de calculer les données nécessaires à la page qui sera renvoyée en réponse au client.

La solution présente cependant un inconvénient : au lieu de nécessiter le chargement d'un seul document, elle nécessite le chargement de plusieurs documents d'où une possible perte de performances.

Récupérer les paramètres envoyés par un client web

2.6.1 par un POST

Considérons le formulaire suivant où l'utilisateur doit donner deux informations : un nom et un âge.

Un formulaire Web

Récupération des valeurs des champs d'un formulaire

Nom	<input type="text"/>	Age	<input type="text"/>
<input type="submit" value="Envoyer"/>			

Lorsque l'utilisateur a rempli les champs *Nom* et *Age*, il appuie ensuite sur le bouton *Envoyer* qui est de type *submit*. Les valeurs du formulaire sont alors envoyées au serveur. Celui-ci renvoie le formulaire avec de plus un tableau listant les valeurs qu'il a reçues :

Un formulaire Web

Récupération des valeurs des champs d'un formulaire

Nom	<input type="text" value="aa"/>	Age	<input type="text" value="aax"/>
<input type="submit" value="Envoyer"/>			

Valeurs récupérées

Nom	aa	Age	aax
-----	----	-----	-----

Le navigateur demande le formulaire à l'application *nomage.php* suivante :

```
<?php
// a-t-on les paramètres attendus ?
$post=isset($_POST["txtNom"]) && isset($_POST["txtAge"]); if($post){
// on récupère les paramètres txtNom et txtAge "postés" par le client
    $nom=$_POST["txtNom"];
    $age=$_POST["txtAge"];
} else {
    $nom="";
    $age="";
} //if
// affichage de la page
include "nomage-p1.php";
?>
```

Quelques explications :

- un champ de formulaire HTML appelé **champ** peut être envoyé au serveur par la méthode GET ou la méthode POST. S'il est envoyé par la méthode GET, le serveur peut le récupérer dans la variable `$_GET["champ"]` et dans la variable `$_POST["champ"]` s'il est envoyé par la méthode POST.
- l'existence d'une donnée peut être testée avec la fonction `isset(donnée)` qui rend *true* si la donnée existe, *false* sinon.
- l'application *nomage.php* construit trois variables : *\$nom* pour le nom du formulaire, *\$age* pour l'âge et *\$post* pour indiquer s'il y a eu des valeurs "postées" ou non. Ces trois variables sont transmises à la page *nomage-p1.php*. On notera que si celle-ci participe à l'élaboration de la réponse au client, celui-ci n'en a pas connaissance. Pour lui, c'est l'application *nomage.php* qui lui répond.
- la première fois qu'un client demande l'application *nomage.php* on a *\$post* à faux. En effet, lors de ce premier appel, aucune valeur de formulaire n'est transmise au serveur.

La page *nomage-p1.php* est la suivante :

```
<html>
<head>
  <title>Formulaire web</title>
</head>
<body>
  <center>
    <h3>Un formulaire Web</h3>
    <h4>Récupération des valeurs des champs d'un formulaire</h4>
    <hr>
    <form name="frmPersonne" method="post">
      <table>
        <tr>
          <td>Nom</td>
          <td><input type="text" value="<?php echo $nom ?>" name="txtNom" size="20"></td>
        </tr>
        <tr>
          <td>Age</td>
          <td><input type="text" value="<?php echo $age ?>" name="txtAge" size="3"></td>
        </tr>
      </table>
      <input type="submit" name="cmdEffacer" value="Envoyer">
    </form>
  </center>
</body>
</html>

<?php
// y-at-il eu des valeurs postées ?
if ($post) {
?>
  <h4>Valeurs récupérées</h4>
  <table border="1">
    <tr>
      <td>Nom</td><td><?php echo $nom ?></td>
      <td width="10"></td>
      <td>Age</td><td><?php echo $age ?></td>
    </tr>
  </table>
<?php } ?>
```

L'application *nomage-p1.php* présente le formulaire *frmPersonne*. Celui est défini par la balise :

```
<form name="frmPersonne" method="post">
```

L'attribut *action* de la balise n'étant pas définie, le navigateur transmettra les données du formulaire à l'URL qu'il a interrogée pour l'avoir, c.a.d. l'application *nomage.php*.

Distinguons les deux cas d'appel de l'application *nomage.php* :

1. C'est la première fois que l'utilisateur l'appelle. L'application *nomage.php* appelle donc l'application *nomage-p1.php* en lui fournissant les valeurs (*\$nom*, *\$age*, *\$post*)=("", "", faux). L'application *nomage-p1.php* affiche alors un formulaire vide.
2. L'utilisateur remplit le formulaire et utilise le bouton *Envoyer* (de type *submit*). Les valeurs du formulaire (*txtNom*, *txtAge*) sont alors "postées" (*method="post"* dans *<form>*) à l'application *nomage.php* (attribut *action* non défini dans *<form>*). L'application *nomage.php* calcule (*\$nom*, *\$age*, *\$post*)=(*txtNom*, *txtAge*, vrai) et les transmet à l'application *nomage-p1.php* qui affiche alors un formulaire déjà rempli ainsi que le tableau des valeurs récupérées.

par un GET

Dans le cas où les valeurs du formulaire sont transmises au serveur par un GET, l'application *nomage.php* devient l'application *nomage2.php* suivante :

```
<?php
// a-t-on les paramètres attendus ?
$get=isset($_GET["txtNom"]) && isset($_GET["txtAge"]); if($get){
// on récupère les paramètres txtNom et txtAge "GETtés" par le client
$nom=$_GET["txtNom"];
$age=$_GET["txtAge"];
} else {
$nom="";
$age="";
} //if
// affichage de la page
include "nomage-p2.php";
?>
```

L'application *nomage-p2.php* est identique à l'application *nomage-p1.php* aux détails près suivants :

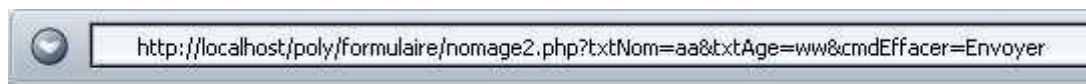
- la balise *form* est modifiée :

```
<form name="frmPersonne" method="get">
```

- l'application récupère maintenant une variable *\$get* plutôt que *\$post*:

```
<?php
// y-at-il eu des valeurs envoyées au serveur ? if ($get)
{
?>
```

À l'exécution, lorsque des valeurs sont saisies dans le formulaire et envoyées au serveur, le navigateur reflète dans son champ URL le fait que les valeurs ont été envoyées par la méthode GET :



Récupérer les entêtes http envoyés par un client web

Lorsqu'un navigateur fait une demande à un serveur web, il lui envoie un certain nombre d'entêtes HTTP. Il est parfois intéressant d'avoir accès à ceux-ci. On peut s'aider dans un premier temps du tableau associatif `$_SERVER`. Celui-ci contient diverses informations qui lui sont données par le serveur web, dont entre-autres les entêtes HTTP fournis par le client. Considérons le programme suivant qui affiche toutes les valeurs du tableau `$_SERVER` :

```
<?php
// affiche les variables liées au serveur web
// on envoie du texte simple
header("Content-type: text/plain");
// parcours du tableau associatif $_SERVER reset($_SERVER);
while (list($clé,$valeur)=each($_SERVER)) { echo "$clé
: $valeur\n";
} //while
?>
```

Sauvegardons ce code dans *headers.php* et demandons cette URL avec un navigateur :



```

COMSPEC : C:\\WINNT\\system32\\cmd.exe
DOCUMENT_ROOT : e:/program files/easyphp/www
HTTP_ACCEPT : text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
HTTP_ACCEPT_CHARSET : ISO-8859-1, utf-8;q=0.66, *;q=0.66
HTTP_ACCEPT_ENCODING : gzip, deflate, compress;q=0.9
HTTP_ACCEPT_LANGUAGE : en-us, en;q=0.50
HTTP_CACHE_CONTROL : max-age=0
HTTP_CONNECTION : keep-alive
HTTP_HOST : localhost
HTTP_KEEP_ALIVE : 300
HTTP_USER_AGENT : Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.0.1) Gecko
PATH : E:\\Perl\\bin;C:\\WINNT\\system32;C:\\WINNT;C:\\WINNT\\System32\\Wbem;C:\\
REMOTE_ADDR : 127.0.0.1
REMOTE_PORT : 1095
SCRIPT_FILENAME : e:/data/serge/web/php/poly/headers/headers.php
SERVER_ADDR : 127.0.0.1
SERVER_ADMIN : admin@localhost
SERVER_NAME : localhost
SERVER_PORT : 80
SERVER_SIGNATURE : <ADDRESS>Apache/1.3.24 Server at <A HREF=\\\"mailto:admin@loca:

SERVER_SOFTWARE : Apache/1.3.24 (Win32)
SystemRoot : C:\\WINNT
WINDIR : C:\\WINNT
GATEWAY_INTERFACE : CGI/1.1
SERVER_PROTOCOL : HTTP/1.1
REQUEST_METHOD : GET
QUERY_STRING :
REQUEST_URI : /poly/headers/headers.php
SCRIPT_NAME : /poly/headers/headers.php
PATH_TRANSLATED : e:/data/serge/web/php/poly/headers/headers.php
PHP_SELF : /poly/headers/headers.php

```

Nous récupérons un certain nombre d'informations dont les entêtes HTTP envoyés par le navigateur. Ceux-ci sont les valeurs associées aux clés commençant par HTTP. Détaillons certaines des informations obtenues ci-dessus :

HTTP_ACCEPT	types de documents acceptés par le client web
HTTP_ACCEPT_CHARSET	types de caractères acceptés dans les documents
HTTP_ACCEPT_ENCODING	types d'encodages acceptés pour les documents
HTTP_ACCEPT_LANGUAGE	types de langues acceptées pour les documents
HTTP_CONNECTION	type de connexion avec le serveur. <i>Keep-Alive</i> : le serveur doit maintenir la liaison ouverte lorsqu'il aura donné sa réponse
HTTP_KEEP_ALIVE	? durée maximale d'ouverture de la liaison
HOST	machine hôte interrogée par le client
HTTP_USER_AGENT	identité du client
REMOTE_ADDR	adresse IP du client
REMOTE_PORT	port de communication utilisé par le client
SERVER_PROTOCOL	protocole HTTP utilisé par le serveur
REQUEST_METHOD	méthode d'interrogation utilisée par le client (GET ou POST)
QUERY_STRING	requête <i>?param1=val1&param2=val2&...</i> placée derrière l'URL demandée (méthode GET)

En modifiant légèrement le code du programme précédent, nous pouvons ne récupérer que les entêtes HTTP :

```

<?php
// affiche les variables liées au serveur web
// on envoie du texte simple

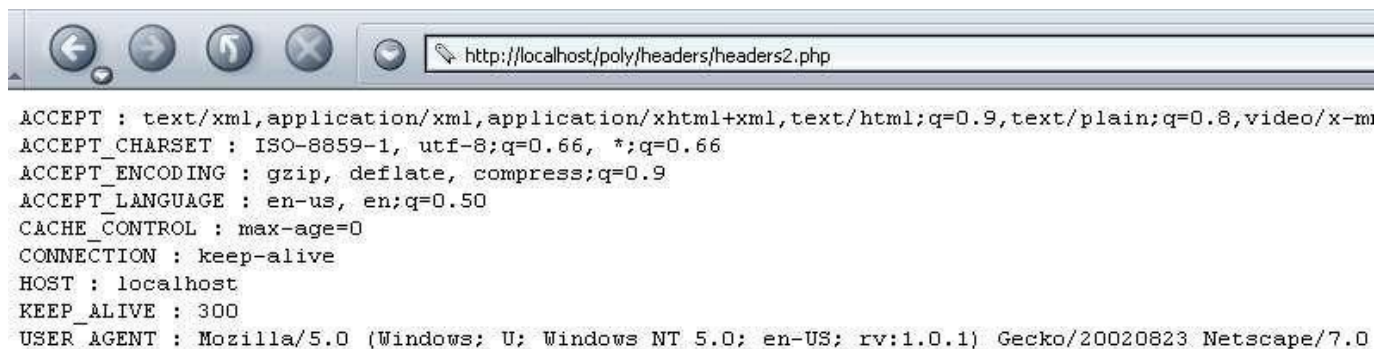
```

```

header("Content-type: text/plain");
// parcours du tableau associatif $_SERVER reset($_SERVER);
while (list($clé, $valeur)=each($_SERVER)) {
    // entête HTTP ?
    if (strtolower(substr($clé, 0, 4))=="http")
        echo substr($clé, 5).": $valeur\n";
} //while
?>

```

Le résultat obtenu dans le navigateur est le suivant :



```

ACCEPT : text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-m
ACCEPT_CHARSET : ISO-8859-1, utf-8;q=0.66, *;q=0.66
ACCEPT_ENCODING : gzip, deflate, compress;q=0.9
ACCEPT_LANGUAGE : en-us, en;q=0.50
CACHE_CONTROL : max-age=0
CONNECTION : keep-alive
HOST : localhost
KEEP_ALIVE : 300
USER_AGENT : Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.0.1) Gecko/20020823 Netscape/7.0

```

Si on veut un entête HTTP précis, on écrira par exemple `$_SERVER["HTTP_ACCEPT"]`.

Récupérer des informations d'environnement

Le serveur web/php s'exécute dans un environnement qu'on peut connaître via le tableau `$_ENV` qui mémorise diverses caractéristiques de l'environnement d'exécution. Considérons l'application *env1.php* suivante :

```

<?php
// affiche les variables liées au serveur web
// on envoie du texte simple
header("Content-type: text/plain");
// parcours du tableau associatif $_ENV reset($_ENV);
while (list($clé, $valeur)=each($_ENV)) { echo
    "$clé : $valeur\n";
} //while
?>

```

Elle donne le résultat suivant dans un navigateur (vue partielle) :



```

ALLUSERSPROFILE : C:\\Documents and Settings\\All Users
APPDATA : C:\\Documents and Settings\\serge\\Application Data
BLASTER : A220 I7 D1 H7 P330 T6
CLASSPATH : \\\"C:\\Program Files\\JavaSoft\\JRE\\1.3.1_02\\lib\\ext\\QTJava.zip\\\"
CommonProgramFiles : C:\\Program Files\\Fichiers communs
COMPUTERNAME : TAHE
ComSpec : C:\\WINNT\\system32\\cmd.exe
HOMEDRIVE : C:
HOMEPATH : \\
INCLUDE : C:\\Program Files\\Microsoft Visual Studio .NET\\FrameworkSDK\\include\\
LIB : C:\\Program Files\\Microsoft.Net\\Odbc.Net\\
LOGONSERVER : \\\\TAHE
NetSamplePath : C:\\PROGRA~1\\MICROS~1.NET\\FRAMEW~1\\Samples
netsdk : c:\\PROGRA~1\\MICROS~1.NET\\FRAMEW~1\\
NUMBER_OF_PROCESSORS : 1
OS : Windows_NT

```

On voit par exemple ci-dessus que le serveur web/php s'exécute sous l'OS Windows NT.

Exemples

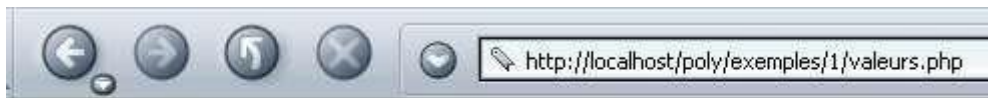
2.9.1 Génération dynamique de formulaire - 1

Nous prenons comme exemple la génération d'un formulaire n'ayant qu'un contrôle : un combo. Le contenu de ce combo est construit dynamiquement avec des valeurs prises dans un tableau. Dans la réalité, elles sont souvent prises dans une base de données. Le formulaire est le suivant :



Choisissez un nombre

Si sur l'exemple ci-dessus, on fait *Envoyer*, on obtient la réponse suivante :



Vous avez choisi le nombre quatre

Le code HTML du formulaire initial, une fois celui-ci généré, est le suivant :

```
<html>
<head>
  <title>Génération de formulaire</title>
</head>
<body>
  <h2>Choisissez un nombre</h2>
  <hr>
  <form name="frmvaleurs" method="post" action="valeurs.php">
    <select name="cmbValeurs" size="1">
      <option>un</option>
    <option>deux</option>
    <option>trois</option>
    <option>quatre</option>
    <option>cinq</option>
    <option>six</option>
    <option>sept</option>
    <option>huit</option>
    <option>neuf</option>
    <option>dix</option>
    </select>
    <input type="submit" value="Envoyer" name="cmdEnvoyer">
  </form>
</body>
</html>
```

L'application PHP est composée d'une page principale *valeurs.php* qui est appelée aussi bien pour obtenir le formulaire initial (la liste des valeurs) que pour traiter les valeurs de celui-ci et fournir la réponse (la valeur choisie). L'application génère deux pages différentes :

- celle du formulaire initial qui sera générée par le programme *valeurs-p1.php*
- celle de la réponse fournie à l'utilisateur qui sera générée par le programme *valeurs-p2.php*

L'application *valeurs.php* est la suivante :

```
<?php
```

```
// le tableau des valeurs
$valeurs=array("un","deux","trois","quatre","cinq","six","sept","huit","neuf","dix");

// a-t-on les paramètres attendus
$requeteVide=! isset($_POST["cmbValeurs"]);

// on récupère le choix de l'utilisateur if
($requeteVide){
    // requête initiale include
    "valeurs-p1.php";
}else{
    // réponse à un POST
    $choix=$_POST["cmbValeurs"];
    include "valeurs-p2.php";
}
?>
```

Elle définit le tableau des valeurs et appelle *valeurs-p1.php* pour générer le formulaire initial si la requête du client était vide ou *valeurs-p2.php* pour générer la réponse si on avait une requête valide. Le programme *valeurs1.php* est le suivant :

```
<html>
<head>
<title>Génération de formulaire</title>
</head>
<body>
<h2>Choisissez un nombre</h2>
<hr>
<form name="frmvaleurs" method="post" action="valeurs.php">
    <select name="cmbValeurs" size="1">
        <?php
            for($i=0;$i<count($valeurs);$i++){
                echo "<option>$valeurs[$i]</option>\n";
            }
        <?php
    </select>
    <input type="submit" value="Envoyer" name="cmdEnvoyer">
</form>
</body>
</html>
```

La liste des valeurs du combo est générée dynamiquement à partir du tableau *\$valeurs* transmis par *valeurs.php*. Le programme *valeurs-p2.php* génère la réponse :

```
<html>
<head>
<title>réponse</title>
</head>
<body>
<h2>Vous avez choisi le nombre <?php echo $choix ?></h2>
</body>
</html>
```

Ici, on se contente d'afficher la valeur de la variable *\$choix* là aussi transmise par *valeurs.php*.

Génération dynamique de formulaire - 2

Nous reprenons l'exemple précédent en le modifiant de la façon suivante. Le formulaire proposé est toujours le même :



La réponse est elle différente :



Choisissez un nombre

Vous avez choisi le nombre quatre

Dans la réponse, on renvoie le formulaire, le nombre choisi par l'utilisateur étant indiqué dessous. Par ailleurs, ce nombre est celui qui apparaît comme sélectionné dans la liste affichée par la réponse.

Le code de *valeurs.php* est le suivant :

```
<?php
// configuration ini_set("register_globals","off");
// le tableau des valeurs
$valeurs=array("un","deux","trois","quatre","cinq","six","sept","huit","neuf","dix");
// on récupère l'éventuel choix de l'utilisateur
$choix=$_POST["cmbValeurs"];
// on affiche la réponse
include "valeurs-pl.php";
?>
```

On remarquera qu'ici, on a pris soin de configurer PHP pour qu'il n'y ait pas de variables globales. C'est en général une saine précaution car les variables globales posent des problèmes de sécurité. Une alternative est d'initialiser toutes les variables qu'on utilise. Cela aura pour effet "d'écraser" une éventuelle variable globale de même nom.

La page du formulaire est affichée par *valeurs-pl.php* :

```
<html>
<head>
<title>Génération de formulaire</title>
</head>
<body>
<h2>Choisissez un nombre</h2>
<hr>
<form name="frmvaleurs" method="post" action="valeurs.php">
  <select name="cmbValeurs" size="1">
    <?php
    for ($i=0;$i<count($valeurs);$i++) {
      // si option courante est égale au choix, on la sélectionne
      if (isset($choix) && $choix==$valeurs[$i])
        echo "<option selected>$valeurs[$i]</option>\n";
      else echo "<option>$valeurs[$i]</option>\n";
    }
  </select>
  <input type="submit" value="Envoyer" name="cmdEnvoyer">
</form>

  <?php
  // suite page
  if(isset($choix)){
    echo "<hr>\n";
    echo "<h3>Vous avez choisi le nombre $choix</h3>\n";
  }
  <?>
</body>
</html>
```


Le programme générateur de la page s'appuie sur la variable *\$choix* passée par le programme *valeurs.php*. Remarquons ici que la structure HTML de la page commence à être sérieusement "polluée" par du code PHP. Le frontal *valeurs.php* pourrait faire davantage de travail comme le montre la nouvelle version suivante :

```
<?php
// configuration
ini_set("register_globals","off");

// le tableau des valeurs
$valeurs=array("un","deux","trois","quatre","cinq","six","sept","huit","neuf","dix");

// on récupère l'éventuel choix de l'utilisateur
$choix=$_POST["cmbValeurs"];

// on calcule la liste des valeurs à afficher
$HTMLvaleurs="";
for($i=0;$i<count($valeurs);$i++){
    // si option courante est égale au choix, on la sélectionne
    if (isset($choix) && $choix==$valeurs[$i])
        $HTMLvaleurs.="<option selected>$valeurs[$i]</option>\n";
    else $HTMLvaleurs.="<option>$valeurs[$i]</option>\n";
}

// on calcule la seconde partie de la page
$HTMLpart2="";
if(isset($choix)){
    $HTMLpart2="<hr>\n";
    $HTMLpart2.="<h3>Vous avez choisi le nombre $choix</h3>\n";
}

// on affiche la réponse include
"valeurs-p2.php";
?>
```

La page est maintenant générée par le programme *valeurs-p2.php* suivant :

```
<html>
<head>
<title>Génération de formulaire</title>
</head>
<body>
<h2>Choisissez un nombre</h2>
<hr>
<form name="frmvaleurs" method="post" action="valeurs.php">
    <select name="cmbValeurs" size="1">
        <?php
            // affichage liste de valeurs
            echo $HTMLvaleurs;
        ?>
    </select>
    <input type="submit" value="Envoyer" name="cmdEnvoyer">
</form>
    <?php
        // affichage partie 2
        echo $HTMLpart2;
    ?>
</body>
</html>
```

Le code HTML est maintenant débarrassé d'une bonne partie du code PHP. Rappelons cependant le but du découpage en un programme frontal qui analyse et traite la demande d'un client et des programmes simplement chargés d'afficher des pages paramétrées par des données transmises par le frontal : c'est de séparer le travail du développeur PHP de celui de l'infographiste. Le développeur PHP travaille sur le frontal, l'infographiste sur les pages web. Dans notre nouvelle version, l'infographiste ne peut plus, par exemple, travailler sur la partie 2 de la page puisqu'il n'a plus accès au code HTML de celle-ci. Dans la première version, il le pouvait. Aucune des deux méthodes n'est donc parfaite.

Génération dynamique de formulaire - 3

Nous reprenons le même problème que précédemment mais cette fois-ci les valeurs sont prises dans une base de données. Celle-ci est dans notre exemple une base MySQL :

- la base s'appelle **dbValeurs**
- son propriétaire est **admDbValeurs** ayant le mot de passe **mdpDbValeurs**

- la base a une unique table appelée **tvaleurs**
- cette table n'a qu'un champ entier appelé **valeur**

```
dos> mysql --database=dbValeurs --user=admDbValeurs --password=mdpDbValeurs

mysql> show tables;
+-----+
| Tables_in_dbValeurs |
+-----+
| tvaleurs             |
+-----+
1 row in set (0.00 sec)

mysql> describe tvaleurs;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| valeur | int(11) |      |      | 0        |       |
+-----+-----+-----+-----+-----+

mysql> select * from tvaleurs;
+-----+
| valeur |
+-----+
| 0      |
| 1      |
| 2      |
| 3      |
| 4      |
| 6      |
| 5      |
| 7      |
| 8      |
| 9      |
+-----+
10 rows in set (0.00 sec)
```

Dans une application utilisant une base de données, on trouvera généralement les étapes suivantes :

1. Connexion au SGBD
2. Émission de requêtes SQL vers une base du SGBD
3. Traitement des résultats de ces requêtes
4. Fermeture de la connexion au SGBD

Les étapes 2 et 3 sont réalisées de façon répétée, la fermeture de connexion n'ayant lieu qu'à la fin de l'exploitation de la base. C'est un schéma relativement classique pour toute personne ayant exploité une base de données de façon interactive. Le tableau suivant donne les instructions PHP pour réaliser ces différentes opérations avec le SGBD MySQL :

connexion au SGBD **\$connexion=mysql_pconnect(\$hote,\$user,\$pwd)**
\$connexion=mysql_connect(\$hote,\$user,\$pwd)

\$hote : nom internet de la machine sur laquelle s'exécute le SGBD MySQL. En effet il est possible de travailler avec des SGBD MySQL distants.

\$user : nom d'un utilisateur connu du SGBD MySQL

\$pwd : son mot de passe

\$connexion : la connexion créée

mysql_pconnect crée une connexion persistante avec le SGBD MySQL. Une telle connexion n'est pas fermée à la fin du script. Elle reste ouverte. Ainsi lorsqu'il faudra ouvrir une nouvelle connexion avec le SGBD, PHP cherchera une connexion existante appartenant au même utilisateur. S'il la trouve, il l'utilise. Il y a là un gain de temps.

mysql_connect crée une connexion non persistante qu'on ferme donc lorsque le travail avec le SGBD MySQL est terminé.

requêtes SQL

\$résultats=mysql_db_query(\$base,\$requête,\$connexion)

\$base : la base de MySQL avec laquelle on va travailler

\$requête : une requête SQL (insert, delete, update, select, ...)

\$connexion : la connexion au SGBD MySQL

\$résultats : les résultats de la requête - diffèrent selon que la requête est un select ou une opération de mise à jour (insert, update, delete, ...)

traitement des
résultats d'un
select

\$résultats=mysql_db_query(\$base,"select ...",\$connexion)

Le résultat d'un select est une table, donc un ensemble de lignes et de colonnes. Cette table est accessible via **\$résultats**.

\$ligne=mysql_fetch_row(\$résultats) lit une ligne de la table et la met dans **\$ligne** sous la forme d'un tableau. Ainsi **\$ligne[i]** est la colonne *i* de la ligne récupérée. La fonction **mysql_fetch_row** peut être appelée de façon répétée. A chaque fois, elle lit une nouvelle ligne de la table *\$résultats*. Lorsque la fin de la table est atteinte, la fonction rend la valeur *false*. Ainsi la table *\$résultats* peut-elle être exploitée de la façon suivante :

```
while($ligne=mysql_fetch_row($résultats)){  
    // traite la ligne courante $ligne  
}
```

traitement des
résultats d'une
requête de mise à
jour

\$résultats=mysql_db_query(\$base,"insert ...",\$connexion)

La valeur **\$résultats** est vraie ou fausse selon que l'opération a réussi ou échoué. En cas de succès, la fonction **mysql_affected_rows** permet de connaître le nombre de lignes modifiées par l'opération de mise à jour.

fermeture de la
connexion

mysql_close(\$connexion)

\$connexion : une connexion au SGBD MySQL

Le code du frontal *valeurs.php* devient le suivant :

```
<?php  
  
// configuration  
ini_set("register_globals","off");  
ini_set("display_errors","off");  
ini_set("track_errors","on");  
  
// le tableau des valeurs  
list($erreur,$valeurs)=getValeurs();  
  
// y-at-il eu erreur ?  
if($erreur){  
    // affichage page d'erreur  
    include "valeurs-err.php";  
    // fin  
    return;  
} //if  
  
// on récupère l'éventuel choix de l'utilisateur  
$choix=$_POST["cmbValeurs"];  
  
// on calcule la liste des valeurs à afficher  
$HTMLvaleurs="";  
for($i=0;$i<count($valeurs);$i++){  
    // si option courante est égale au choix, on la sélectionne if  
    (isset($choix) && $choix==$valeurs[$i])  
        $HTMLvaleurs.="<option selected>$valeurs[$i]</option>¥n"; else  
        $HTMLvaleurs.="<option>$valeurs[$i]</option>¥n";  
} //for  
  
// on calcule la seconde partie de la page  
$HTMLpart2="";  
if(isset($choix)){  
    $HTMLpart2="<hr>¥n";  
    $HTMLpart2.="<h3>Vous avez choisi le nombre $choix</h3>¥n";  
} //if  
  
// on affiche la réponse include  
"valeurs-p1.php";  
  
// fin  
return;
```

```
// -----
function getValeurs(){
// récupère les valeurs dans une base MySQL
$user="admDbValeurs";
$pwd="mdpDbValeurs";
$db="dbValeurs";
$hote="localhost";
$table="tvaleurs";
$champ="valeur";

// ouverture d'une connexion persistante au serveur MySQL
// ou sinon d'une connexion normale
($connexion=mysql_pconnect($hote,$user,$pwd)
|| ($connexion=mysql_connect($hote,$user,$pwd));
if(! $connexion)
return array("Base de données indisponible(".mysql_error()."). Veuillez recommencer
ultérieurement.");

// obtention des valeurs
$selectValeurs=mysql_db_query($db,"select $champ from $table",$connexion);
if(! $selectValeurs)
return array("Base de données indisponible(".mysql_error()."). Veuillez recommencer
ultérieurement.");

// les valeurs sont mises dans un tableau
$valeurs=array();
while($ligne=mysql_fetch_row($selectValeurs)){
$valeurs[]=$ligne[0];
}

// fermeture de la connexion (si elle est persistante, elle ne sera en fait pas fermée)
mysql_close($connexion);

// retour du résultat
return array("", $valeurs);
}

//getValeurs

```

?>

Cette fois-ci, les valeurs à mettre dans le combo ne sont pas fournies par un tableau mais par la fonction *getValeurs()*. Cette fonction :

- ouvre une connexion **persistante** (*mysql_pconnect*) avec le serveur *mySQL* en passant un nom d'utilisateur enregistré et son mot de passe.
- une fois la connexion obtenue, une requête *select* est émise pour récupérer les valeurs présentes dans la table *tvaleurs* de la base *dbValeurs*.
- le résultat du *select* est mis dans le tableau *\$valeurs* qui est rendu au programme appelant.
- la fonction rend en fait un tableau à deux résultats (*\$erreur*, *\$valeurs*) où le premier élément est un éventuel message d'erreur ou la chaîne vide sinon.
- le programme appelant teste s'il y a eu erreur ou non et si oui fait afficher la page *valeurs-err.php*. Celle-ci est la suivante :

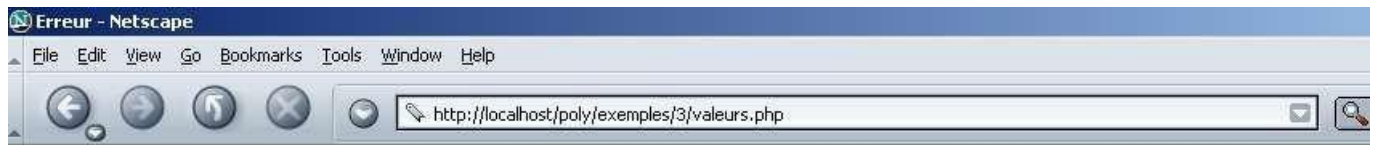
```
<html>
<head>
<title>Erreur</title>
</head>
<body>
<h3>L'erreur suivante s'est produite</h3>
<font color="red">
<h4><?php echo $erreur ?></h4>
</font>
</body>
</html>

```

- s'il n'y a pas eu erreur, le programme appelant dispose des valeurs dans le tableau *\$valeurs*. On est donc ramené au problème précédent.

Voici deux exemples d'exécution :

- **avec erreur**



L'erreur suivante s'est produite

Base de données indisponible(Can't connect to MySQL server on 'localhost' (10061)). Veuillez recommencer ultérieurement.

- sans erreur



Choisissez un nombre

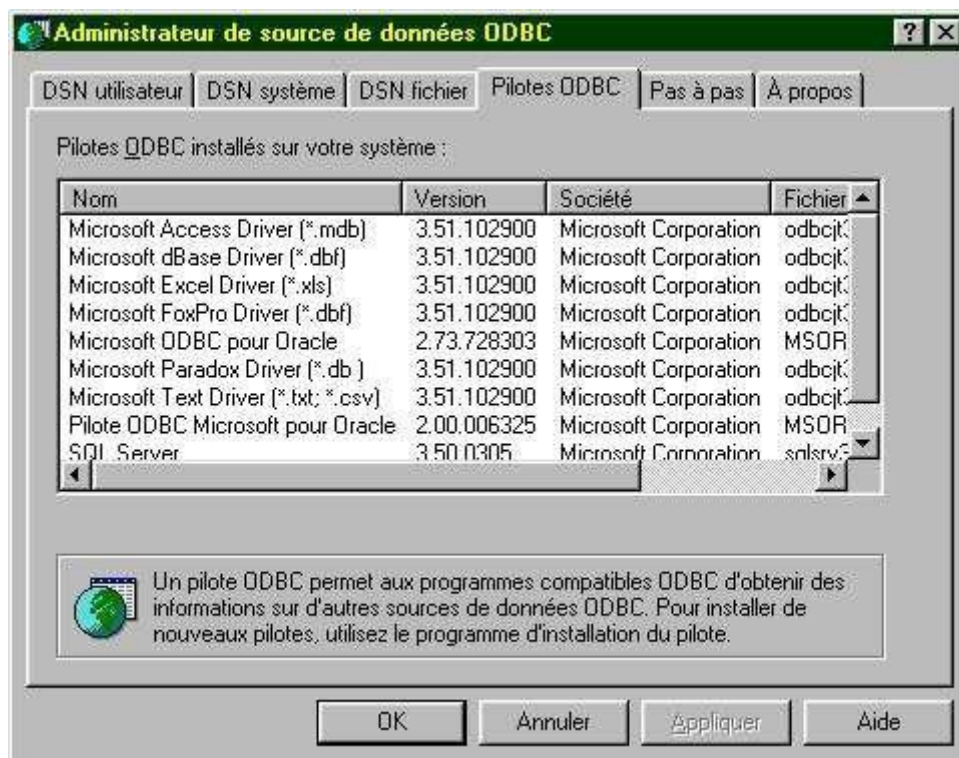
Envoyer

Vous avez choisi le nombre 5

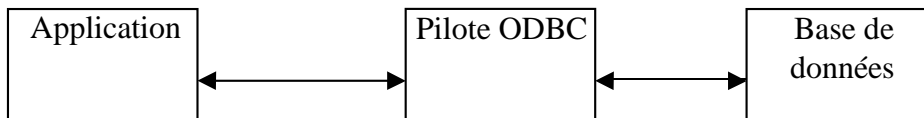
Génération dynamique de formulaire - 4

Dans l'exemple précédent, que se passerait-il si on changeait de SGBD ? Si on passait de MySQL à Oracle ou SQL Server par exemple ? Il faudrait réécrire la fonction *getValeurs()* qui fournit les valeurs. L'intérêt d'avoir rassemblé le code nécessaire à la récupération des valeurs à afficher dans la liste dans une fonction est que le code à modifier est bien ciblé et non pas éparpillé dans tout le programme. La fonction *getValeurs()* peut être réécrite de façon à ce qu'elle soit indépendante du SGBD utilisé. Il suffit qu'elle travaille avec le pilote ODBC du SGBD plutôt que directement avec le SGBD.

Il existe de nombreuses bases de données sur le marché. Afin d'uniformiser les accès aux bases de données sous MS Windows, Microsoft a développé une interface appelée ODBC (Open DataBase Connectivity). Cette couche cache les particularités de chaque base de données sous une interface standard. Il existe sous MS Windows de nombreux pilotes ODBC facilitant l'accès aux bases de données. Voici par exemple, une liste de pilotes ODBC installés sur une machine Windows :

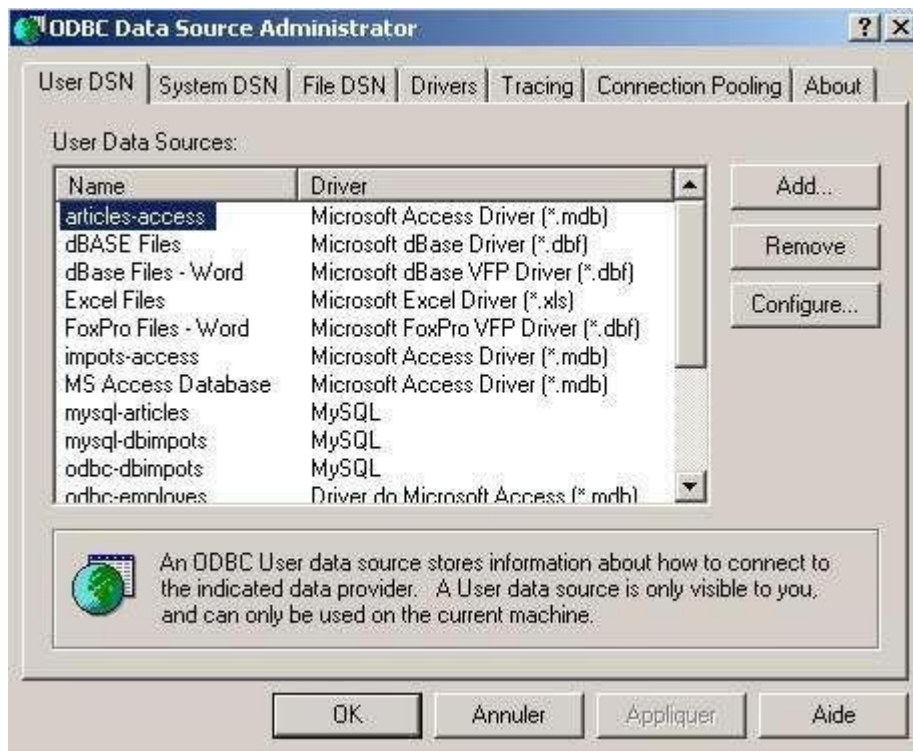


Le SGBD MySQL a lui aussi un pilote ODBC. Une application s'appuyant sur des pilotes ODBC peut utiliser n'importe quelle bases de données ci-dessus sans ré-écriture.

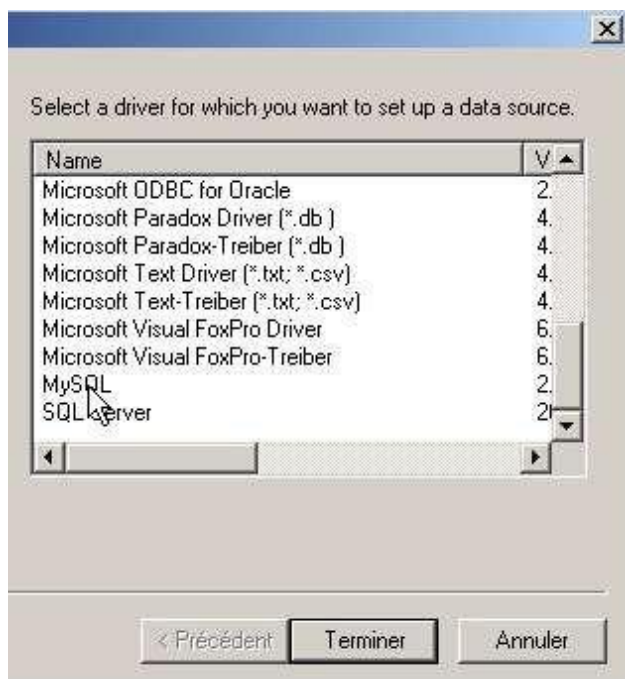


Rendons notre base MySQL *dbValeurs* accessible au travers d'un pilote ODBC. La procédure ci-dessous est celle de Windows 2000. Pour les systèmes Win9x, la procédure est très analogue. On active le gestionnaire de ressources ODBC :

Menu Démarrer/Panneau de configuration/Outils d'administration/Sources de données ODBC



On utilise le bouton [Add] pour ajouter une nouvelle source de données ODBC :



On sélectionne le pilote ODBC MySQL, on fait [Terminer] puis on donne les caractéristiques de la source de données :

Windows DSN name	le nom donné à la source de données ODBC (odbc-valeurs)
MySQL host	le nom de la machine qui héberge le SGBD MySQL qui gère la source de données (localhost)
MySQL database name	le nom de la base MySQL qui est la source de données (dbValeurs)
User	un utilisateur ayant des droits d'accès suffisants sur la base MySQL à gérer (admDbValeurs)
Password	son mot de passe (mdpDbValeurs)

PHP est capable de travailler avec des pilotes ODBC. Le tableau suivant donne les fonctions utiles à connaître :

connexion au SGBD **\$connexion=odbc_pconnect(\$dsn,\$user,\$pwd)**
\$connexion=odbc_connect(\$dsn,\$user,\$pwd)

\$dsn : nom DSN (Data Source Name) de la machine sur laquelle s'exécute le SGBD

\$user : nom d'un utilisateur connu du SGBD

\$pwd : son mot de passe

\$connexion : la connexion créée

odbc_pconnect crée une connexion persistante avec le SGBD. Une telle connexion n'est pas fermée à la fin du script. Elle reste ouverte. Ainsi lorsqu'il faudra ouvrir une nouvelle connexion avec le SGBD, PHP cherchera une connexion existante appartenant au même utilisateur. S'il la trouve, il l'utilise. Il y a là un gain de temps.

odbc_connect crée une connexion non persistante qu'on ferme donc lorsque le travail avec le SGBD est terminé.

requêtes SQL

\$requetePréparée=odbc_prepare(\$connexion,\$requete)

\$requete : une requête SQL (insert, delete, update, select, ...)

\$connexion : la connexion au SGBD

Analyse la requête *\$requete* et prépare son exécution. La requête ainsi "préparée" est référencée par le résultat **\$requetePréparée**. Préparer une requête pour son exécution n'est pas obligatoire mais améliore les performances puisque l'analyse de la requête n'est faite qu'une fois. On demande ensuite l'exécution de la requête préparée. Si on demande l'exécution d'une requête non préparée de façon répétée, l'analyse de celle-ci est faite à chaque fois, ce qui est inutile. Une fois préparée la requête est exécutée par

\$res=odbc_execute(\$requetePréparée)

rend vrai ou faux selon que l'exécution de la requête réussit ou non

traitement des
résultats d'un
select

Le résultat d'un select est une table, donc un ensemble de lignes et de colonnes. Cette table est accessible via **\$requetePréparée**.

\$res=odbc_fetch_row(\$requetePréparée) lit une ligne de la table résultat du *select*. Rend vrai ou faux selon que l'exécution de la requête réussit ou non. Les éléments de la ligne récupérée sont disponibles via la fonction **odbc_result** :

\$val=odbc_result(\$requetePréparée,i) : colonne i de la ligne qui vient d'être lue

\$val=odbc_result(\$requetePréparée,"nomColonne") : colonne nomColonne de la ligne qui vient d'être lue

La fonction *odbc_fetch_row* peut être appelée de façon répétée. A chaque fois, elle lit une nouvelle ligne de la table des résultats. Lorsque la fin de la table est atteinte, la fonction rend la valeur *false*. Ainsi la table *des résultats* peut-elle être exploitée de la façon suivante :

```
while(odbc_fetch_row($résultats)){  
    //traite la ligne courante avec odbc_result  
}  
//while
```

fermeture de la
connexion

odbc_close(\$connexion)

\$connexion : une connexion au SGBD MySQL

La fonction *getValeurs()* chargée de récupérer les valeurs dans la base de données ODBC est la suivante :

```
// -----  
function getValeurs() {  
    // récupère les valeurs dans une base MySQL  
    $user="admDbValeurs";  
    $pwd="mdpDbValeurs";  
    $db="dbValeurs";  
    $dsn="odbc-valeurs";  
    $table="tvaleurs";  
    $champ="valeur";  
  
    // ouverture d'une connexion persistante au serveur MySQL  
    // ou sinon d'une connexion normale  
    ($connexion=odbc_pconnect($dsn,$user,$pwd))  
    || ($connexion=odbc_connect($dsn,$user,$pwd)); if(!  
    $connexion)  
        return array("1 - Base de données indisponible(" . odbc_error() . "). Veuillez recommencer  
ultérieurement.");  
  
    // obtention des valeurs  
    $selectValeurs=odbc_prepare($connexion,"select $champ from $table"); if(!  
    odbc_execute($selectValeurs))  
        return array("2 - Base de données indisponible(" . odbc_error() . "). Veuillez recommencer  
ultérieurement.");  
  
    // les valeurs sont mises dans un tableau  
    $valeurs=array();  
    while(odbc_fetch_row($selectValeurs)) {  
        $valeurs[] = odbc_result($selectValeurs,$champ);  
    }  
    //while  
}
```



```
// fermeture de la connexion (si elle est persistante, elle ne sera en fait pas fermée) odbc_close($connexion);
// retour du résultat return
array("", $valeurs);
} //getValeurs
?>
```

Si on exécute la nouvelle application sans activer la base de données *odbc-valeurs*, on obtient le résultat suivant :



L'erreur suivante s'est produite

Base de données indisponible(S1000). Veuillez recommencer ultérieurement.

On remarquera que le code d'erreur renvoyé par le pilote odbc (odbc_error()=S1000) est peu explicite. Si la base *odbc-valeurs* est rendue disponible, on obtient les mêmes résultats qu'auparavant.

En conclusion, on peut dire que cette solution est bonne pour la maintenance de l'application. En effet, si la base de données doit changer, l'application n'aura elle pas à changer. L'administrateur système créera simplement une nouvelle source de données ODBC pour la nouvelle base. Toujours en vue de la maintenance, il serait bon de mettre les paramètres d'accès à la base (\$dsn, \$user, \$pwd) dans un fichier séparé que l'application chargerait au démarrage (*include*).

Récupérer les valeurs d'un formulaire

Nous avons déjà à plusieurs reprises récupéré les valeurs d'un formulaire envoyées par un client web. L'exemple suivant montre un formulaire rassemblant les composants HTML les plus courants et s'applique à récupérer les paramètres envoyés par le navigateur client. Le formulaire est le suivant :

Etes-vous marié(e)	<input type="radio"/> Oui <input checked="" type="radio"/> Non
Cases à cocher	<input type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	<input type="text" value="qqs mots"/>
Mot de passe	<input type="password" value="aaaaaaaaaaaa"/>
Boîte de saisie	<div>ligne1</div> <div>ligne2</div> <div>ligne3</div>
combo	<div>choix2</div>
liste à choix simple	<div>liste1</div> <div>liste2</div> <div>liste3</div>
liste à choix multiple	<div>liste1</div> <div>liste2</div> <div>liste3</div>
bouton	<div>Effacer</div>
envoyer	<div>Envoyer</div>
rétablir	<div>Rétablir</div>

Il arrive pré-rempli. Ensuite l'utilisateur peut le modifier :

Etes-vous marié(e) ☒ Oui ☐ Non

Cases à cocher ☒ 1 ☐ 2 ☒ 3

Champ de saisie

Mot de passe

Boîte de saisie

ligne1

ligne2

ligne3

combo

choix3

liste à choix simple

liste3

liste4

liste5

liste à choix multiple

liste1

liste2

liste3

bouton

envoyer

rétablir

S'il appuie sur le bouton [Envoyer] le serveur lui renvoie la liste des valeurs du formulaire :

R1	Oui
C1	un
C2	
C3	trois
txtSaisie	un texte
txtMdp	abcdefgh
areaSaisie	ligne1 ligne2 ligne3 ligne4 ligne5
cmbValeurs	choix3
lst1	liste5
lst2	liste3 liste5
secret	une Valeur

Le formulaire est une page HTML statique **balises.html** :

```
<html>
```

```

<head>
<title>balises</title>
<script language="JavaScript">
    function effacer() {
        alert("Vous avez cliqué sur le bouton Effacer");
    } //effacer
</script>
</head>

<body background="/images/standard.jpg">
    <form method="POST" action="parameters.php">
        <table border="0">
            <tr>
                <td>Etes-vous marié(e)</td>
                <td>
                    <input type="radio" value="oui" name="R1">Oui
                    <input type="radio" name="R1" value="non" checked>Non
                </td>
            </tr>
            <tr>
                <td>Cases à cocher</td>
                <td>
                    <input type="checkbox" name="C1" value="un">1
                    <input type="checkbox" name="C2" value="deux" checked>2
                    <input type="checkbox" name="C3" value="trois">3
                </td>
            </tr>
            <tr>
                <td>Champ de saisie</td>
                <td>
                    <input type="text" name="txtSaisie" size="20" value="qqs mots">
                </td>
            </tr>
            <tr>
                <td>Mot de passe</td>
                <td>
                    <input type="password" name="txtMdp" size="20" value="unMotDePasse">
                </td>
            </tr>
            <tr>
                <td>Boîte de saisie</td>
                <td>
                    <textarea rows="2" name="areaSaisie" cols="20">
ligne1
ligne2
ligne3
</textarea>
                </td>
            </tr>
            <tr>
                <td>combo</td>
                <td>
                    <select size="1" name="cmbValeurs">
                        <option>choix1</option>
                        <option selected>choix2</option>
                        <option>choix3</option>
                    </select>
                </td>
            </tr>
            <tr>
                <td>liste à choix simple</td>
                <td>
                    <select size="3" name="lst1">
                        <option selected>liste1</option>
                        <option>liste2</option>
                        <option>liste3</option>
                        <option>liste4</option>
                        <option>liste5</option>
                    </select>
                </td>
            </tr>
            <tr>
                <td>liste à choix multiple</td>
                <td>
                    <select size="3" name="lst2[]" multiple>
                        <option selected>liste1</option>
                        <option>liste2</option>
                        <option selected>liste3</option>
                        <option>liste4</option>
                    </select>
                </td>
            </tr>
        </table>
    </form>

```

```

        <option>liste5</option>
    </select>
</td>
</tr>
<tr>
<td>bouton</td>
<td>
        <input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()">
    </td>
</tr>
<tr>
<td>envoyer</td>
<td>
        <input type="submit" value="Envoyer" name="cmdRenvoyer">
    </td>
</tr>
<tr>
<td>r tablir</td>
<td>
        <input type="reset" value="R tablir" name="cmdR tablir">
    </td>
</tr>
</table>
<input type="hidden" name="secret" value="uneValeur">
</form>
</body>
</html>

```

Le tableau ci-dessous rappelle le r le des diff rentes balises de ce document et la valeur r cup r e par PHP pour les diff rents types de composants d'un formulaire. La valeur d'un champ de nom HTML C peut  tre envoy e par un POST ou un GET. Dans le premier cas, elle sera r cup r e dans la variable \$_GET["C"] et dans le second cas dans la variable \$_POST["C"]. Le tableau suivant suppose l'utilisation d'un POST.

Contr�le	balise HTML	valeur r�cup�r�e par PHP
formulaire	<form method="POST" >	
champ de saisie	<input type="text" name="txtSaisie" size="20" value="qqs mots">	\$_POST["txtSaisie"] : valeur contenue dans le champ txtSaisie du formulaire
champ de saisie cach�e	<input type="password" name="txtMdp" size="20" value="unMotDePasse">	\$_POST["txtmdp"] : valeur contenue dans le champ txtMdp du formulaire
champ de saisie multilignes	<textarea rows="2" name="areaSaisie" cols="20"> ligne1 ligne2 ligne3 </textarea>	\$_POST["areaSaisie"] : lignes contenues dans le champ <i>areaSaisie</i> sous la forme d'une unique cha�ne de caract�res : "ligne1\r\nligne2\r\nligne3". Les lignes sont s�par�es entre elles par la s�quence "\r\n".
boutons radio	<input type="radio" value="Oui" name="R1">Oui <input type="radio" name="R1" value="non" checked>Non	\$_POST["R1"] : valeur (=value) du bouton radio co�h� "oui" ou "non" selon les cas.
cases � co�her	<input type="checkbox" name="C1" value="un">1 <input type="checkbox" name="C2" value="deux" checked>2 <input type="checkbox" name="C3" value="trois">3	\$_POST["C1"] : valeur (=value) de la case si elle est co�h�e sinon la variable n'existe pas. Ainsi si la case C1 a �t� co�h�e, \$_POST["C1"] vaut "un", sinon \$_POST["C1"] n'existe pas.
Combo	<select size="1" name="cmbValeurs"> <option>choix1</option> <option selected>choix2</option> <option>choix3</option> </select>	\$_POST["cmbValeurs"] : option s�lectionn�e dans la liste, par exemple "choix3".
liste � s�lection uniaue	<select size="3" name="lst1"> <option selected>liste1</option> <option>liste2</option> <option>liste3</option> <option>liste4</option> <option>liste5</option> </select>	\$_POST["lst1"] : option s�lectionn�e dans la liste, par exemple "liste5".

liste à sélection multiple	<pre><select size="3" name="lst2[]" multiple> <option>liste1</option> <option>liste2</option> <option selected>liste3</option> <option>liste4</option> <option>liste5</option> </select></pre>	<p><code>\$_POST["lst2"]</code> : tableau des options sélectionnées dans la liste, par exemple ["liste3","liste5"]. On notera la syntaxe particulière de la balise HTML pour ce cas particulier : lst2[].</p>
----------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

champ caché	<pre><input type="hidden" name="secret" value="uneValeur"></pre>	<p><code>\$_POST["secret"]</code> : valeur (=value) du champ, ici "uneValeur".</p>
-------------	------------------------------------------------------------------------	------------------------------------------------------------------------------------

Dans notre exemple, les valeurs du formulaire sont adressées au programme *parameters.php* :

```
<form method="POST" action="parameters.php">
```

Le code de ce dernier est le suivant :

```
<?php
// configuration
ini_set("register_globals","off");
ini_set("display_errors","off");

// méthode d'appel
$méthode=$_SERVER["REQUEST_METHOD"];

// récupération des paramètres
// elle dépend de la méthode d'envoi de ceux-ci
if ($méthode=="GET")
    $param=$_GET;
else $param=$_POST;

$R1=$param["R1"];
$C1=$param["C1"];
$C2=$param["C2"];
$C3=$param["C3"];
$txtSaisie=$param["txtSaisie"];
$txtMdp=$param["txtMdp"];

$areaSaisie=implode("<br>",$param["areaSaisie"]);
$cmbValeurs=$param["cmbValeurs"];
$lst1=$param["lst1"];

$lst2=implode("<br>",$param["lst2"]);
$secret=$param["secret"];

// requête valide ?
$requêteValide=isset($R1) && (isset($C1) || isset($C2) || isset($C3)) &&
    isset($txtSaisie) && isset($txtMdp) && isset($areaSaisie)
    && isset($cmbValeurs) && isset($lst1) && isset($lst2) &&
    isset($secret);

// affichage page if
($requêteValide)
    include "parameters-p1.php"; else
    include "balises.html";
?>
```

Détaillons quelques points de ce programme :

- l'application s'affranchit du mode de passage des valeurs du formulaire au serveur. Dans les deux cas possibles (GET et POST), le dictionnaire des valeurs passées est référencé par *\$param*.

```
if ($méthode=="GET")
    $param=$_GET;
else $param=$_POST;
```

- à partir du contenu du champ *areaSaisie* "ligne1\r\nligne2\r\n..." on crée un tableau de chaînes par *explode("\r\n",\$param["areaSaisie"])*. On a donc le tableau [*ligne1*,*ligne2*,...]. A partir de celui-ci on crée la chaîne "*ligne1
ligne2
...*" avec la fonction *implode*.
- la valeur de la liste à sélection multiple *lst2* est un tableau, par exemple ["option3","option5"]. A partir de celui-ci, on crée une chaîne de caractères "option3
option5" avec la fonction *implode*.
- l'application vérifie que tous les paramètres ont été positionnés. Il faut se rappeler ici que n'importe quelle URL peut être appelée à la main ou par programme et qu'on n'a pas nécessairement les paramètres attendus. S'il manque des paramètres,

la page balises.html est affichée sinon la page *parameters-p1.php*. Celle-ci affiche les valeurs récupérées et calculées dans *parameters.php* dans un tableau :

```
<html>
<head>
<title>Récupération des paramètres d'un formulaire</title>
</head>
<body>
<table border="1">
<tr>
<td>R1</td>
<td><?php echo $R1 ?></td>
</tr>
<tr>
<td>C1</td>
<td><?php echo $C1 ?></td>
</tr>
<tr>
<td>C2</td>
<td><?php echo $C2 ?></td>
</tr>
<tr>
<td>C3</td>
<td><?php echo $C3 ?></td>
</tr>
<tr>
<td>txtSaisie</td>
<td><?php echo $txtSaisie ?></td>
</tr>
<tr>
<td>txtMdp</td>
<td><?php echo $txtMdp ?></td>
</tr>
<tr>
<td>areaSaisie</td>
<td><?php echo $areaSaisie ?></td>
</tr>
<tr>
<td>cmbValeurs</td>
<td><?php echo $cmbValeurs ?></td>
</tr>
<tr>
<td>lst1</td>
<td><?php echo $lst1 ?></td>
</tr>
<tr>
<td>lst2</td>
<td><?php echo $lst2 ?></td>
</tr>
<tr>
<td>secret</td>
<td><?php echo $secret ?></td>
</tr>
</table>
</body>
</html>
```

Suivi de session

2.10.1 Le problème

Une application web peut consister en plusieurs échanges de formulaires entre le serveur et le client. On a alors le fonctionnement suivant :

- **étape 1**
 1. le client C1 ouvre une connexion avec le serveur et fait sa requête initiale.
 2. le serveur envoie le formulaire F1 au client C1 et ferme la connexion ouverte en 1.
- **étape 2**
 3. le client C1 remplit le formulaire et le renvoie au serveur. Pour ce faire, le navigateur ouvre une nouvelle connexion avec le serveur.
 4. celui-ci traite les données du formulaire 1, calcule des informations I1 à partir de celles-ci, envoie un formulaire F2 au client C1 et ferme la connexion ouverte en 3.
- **étape 3**
 5. le cycle des étapes 3 et 4 se répète dans des étapes 5 et 6. A l'issue de l'étape 6, le serveur aura reçu deux formulaires F1 et F2 et à partir de ceux-ci aura calculé des informations I1 et I2.

Le problème posé est : comment le serveur fait-il pour conserver les informations I1 et I2 liées au client C1 ? On appelle ce problème le suivi de la session du client C1. Pour comprendre sa racine, examinons le schéma d'une application serveur TCP-IP servant simultanément plusieurs clients :



Dans une application client-serveur TCP-IP classique :

- le client crée une connexion avec le serveur
- échange à travers celle-ci des données avec le serveur
- la connexion est fermée par l'un des deux partenaires

Les deux points importants de ce mécanisme sont :

1. une connexion unique est créée pour chacun des clients
2. cette connexion est utilisée pendant toute la durée du dialogue du serveur avec son client

Ce qui permet au serveur de savoir à un moment donné avec quel client il travaille, c'est la connexion ou dit autrement le "tuyau" qui le relie à son client. Ce tuyau étant dédié à un client donné, tout ce qui arrive de ce tuyau vient de ce client et tout ce qui est envoyé dans ce tuyau arrive à ce même client.

Le mécanisme du client-serveur HTTP suit le schéma précédent avec cependant la particularité que le dialogue client-serveur est limité à un **unique échange** entre le client et le serveur :

- le client ouvre une connexion vers le serveur et fait sa demande
- le serveur fait sa réponse et ferme la connexion

Si au temps T1, un client C fait une demande au serveur, il obtient une connexion C1 qui va servir à l'échange unique demande-réponse. Si au temps T2, ce même client fait une seconde demande au serveur, il va obtenir une connexion C2 différente de la connexion C1. Pour le serveur, il n'y a alors aucune différence entre cette seconde demande de l'utilisateur C et sa demande initiale : dans les deux cas, le serveur considère le client comme un nouveau client. Pour qu'il y ait un lien entre les différentes connexions du client C au serveur, il faut que le client C se fasse "reconnaître" par le serveur comme un "habitué" et que le serveur récupère les informations qu'il a sur cet habitué.

Imaginons une administration qui fonctionnerait de la façon suivante :

- Il y a une unique file d'attente
- Il y a plusieurs guichets. Donc plusieurs clients peuvent être servis simultanément. Lorsqu'un guichet se libère, un client quitte la file d'attente pour être servi à ce guichet
- Si c'est la première fois que le client se présente, la personne au guichet lui donne un jeton avec un numéro. Le client ne peut poser qu'une question. Lorsqu'il a sa réponse il doit quitter le guichet et passer à la fin de la file d'attente. Le guichetier note les informations de ce client dans un dossier portant son numéro.
- Lorsqu'il arrive à nouveau son tour, le client peut être servi par un autre guichetier que la fois précédente. Celui-ci lui demande son jeton et récupère le dossier ayant le numéro du jeton. De nouveau le client fait une demande, a une réponse et des informations sont rajoutées à son dossier.
- et ainsi de suite... Au fil du temps, le client aura la réponse à toutes ses requêtes. Le suivi entre les différentes requêtes est réalisé grâce au jeton et au dossier associé à celui-ci.

Le mécanisme de suivi de session dans une application client-serveur web est analogue au fonctionnement précédent :

- lors de sa première demande, un client se voit donner un jeton par le serveur web
- il va présenter ce jeton à chacune de ses demandes ultérieures pour s'identifier

Le jeton peut revêtir différentes formes :

- **celui d'un champ caché dans un formulaire**

- le client fait sa première demande (le serveur le reconnaît au fait que le client n'a pas de jeton)
- le serveur fait sa réponse (un formulaire) et met le jeton dans un champ caché de celui-ci. A ce moment, la connexion est fermée (le client quitte le guichet avec son jeton). Le serveur a pris soin éventuellement d'associer des informations à ce jeton.
- le client fait sa seconde demande en renvoyant le formulaire. Le serveur récupère dans celui-ci le jeton. Il peut alors traiter la seconde demande du client en ayant accès, grâce au jeton, aux informations calculées lors de la première demande. De nouvelles informations sont ajoutées au dossier lié au jeton, une seconde réponse est faite au client et la connexion est fermée pour la seconde fois. Le jeton a été mis de nouveau dans le formulaire de la réponse afin que l'utilisateur puisse le présenter lors de sa demande suivante.
- et ainsi de suite...

L'inconvénient principal de cette technique est que le jeton doit être mis dans un formulaire. Si la réponse du serveur n'est pas un formulaire, la méthode du champ caché n'est plus utilisable.

■ celui du cookie

- le client fait sa première demande (le serveur le reconnaît au fait que le client n'a pas de jeton)
- le serveur fait sa réponse en ajoutant un cookie dans les entêtes HTTP de celle-ci. Cela se fait à l'aide de la commande HTTP **Set-Cookie** :

Set-Cookie: param1=valeur1;param2=valeur2;....

où *parami* sont des noms de paramètres et *valeursi* leurs valeurs. Parmi les paramètres, il y aura le jeton. Bien souvent, il n'y a que le jeton dans le cookie, les autres informations étant consignées par le serveur dans le dossier lié au jeton. Le navigateur qui reçoit le cookie va le stocker dans un fichier sur le disque. Après la réponse du serveur, la connexion est fermée (le client quitte le guichet avec son jeton).

- le client fait sa seconde demande au serveur. A chaque fois qu'une demande à un serveur est faite, le navigateur regarde parmi tous les cookies qu'il a, s'il en a un provenant du serveur demandé. Si oui, il l'envoie au serveur toujours sous la forme d'une commande HTTP, la commande **Cookie** qui a une syntaxe analogue à celle de la commande *Set-Cookie* utilisée par le serveur :

Cookie: param1=valeur1;param2=valeur2;....

Parmi les entêtes HTTP envoyés par le navigateur, le serveur retrouvera le jeton lui permettant de reconnaître le client et de retrouver les informations qui lui sont liées.

C'est la forme la plus utilisée de jeton. Elle présente un inconvénient : un utilisateur peut configurer son navigateur afin qu'il n'accepte pas les cookies. Ce type d'utilisateur n'a alors pas accès aux applications web avec cookie.

■ réécriture d'URL

- le client fait sa première demande (le serveur le reconnaît au fait que le client n'a pas de jeton)
- le serveur envoie sa réponse. Celle-ci contient des liens que l'utilisateur doit utiliser pour continuer l'application. Dans l'URL de chacun de ces liens, le serveur rajoute le jeton sous la forme *URL;jeton=valeur*.
- lorsque l'utilisateur clique sur l'un des liens pour continuer l'application, le navigateur fait sa demande au serveur web en lui envoyant dans les entêtes HTTP l'URL *URL;jeton=valeur* demandée. Le serveur est alors capable de récupérer le jeton.

L'API PHP pour le suivi de session

Nous présentons maintenant les principales méthodes utiles au suivi de session :

<code>session_start()</code>	démarre la session à laquelle appartient la requête en cours. Si celle-ci ne faisait pas encore partie d'une session, cette dernière est créée.
<code>session_id()</code>	identifiant de la session en cours
<code>\$_SESSION[\$var]</code>	dictionnaire mémorisant les données d'une session. Accessible en lecture et écriture
<code>session_destroy()</code>	supprime les données contenues dans la session en cours. Ces données restent disponibles pour l'échange client-serveur en cours mais seront indisponibles lors du prochain échange.

Exemple 1

Nous présentons un exemple inspiré du livre "Programmation avec J2EE" aux éditions Wrox et distribué par Eyrolles. Cet exemple permet de voir comment fonctionne une session PHP. La page principale est la suivante :



Cycle de vie d'une session PHP

ID session : e4b12bd71499318dbd6d72d0c966d078

compteur : 0

[Invalidier la session](#)

[Recharger la page](#)

On y trouve les éléments suivants :

- l'ID de la session obtenue par la fonction `session_id()`. Cet ID généré par le navigateur est envoyé au client par un cookie que le navigateur renvoie lorsqu'il demande une URL de la même arborescence. C'est ce qui maintient la session.
- un compteur qui est incrémenté au fil des demandes du navigateur et qui montre que la session est bien maintenue.
- un lien permettant de supprimer les données attachées à la session en cours. Ceci est fait par la fonction `session_destroy()`
- un lien pour recharger la page

Le code de l'application *cycledevie.php* est le suivant :

```
<?php
//cycledevie.php

// configuration ini_set("register_globals","off");
ini_set("display_errors","off");

// on démarre une session
session_start();

// faut-il l'invalider ?
$action=$_GET["action"];
if($action=="invalider"){
    // fin session
    session_destroy();
}

// gestion du compteur
if(!isset($_SESSION["compteur"]))
    // le compteur n'existe pas - on le crée
    $_SESSION["compteur"]=0;
// le compteur existe - on l'incrémente else
$_SESSION["compteur"]++;

// on récupère l'ID de la session courante
$idSession=session_id();

// on récupère le compteur
$compteur=$_SESSION["compteur"];

// on passe la main à la page de visualisation include
"cycledevie-p1.php";
?>
```

Notons les points suivants :

- dès le début de l'application, une session est démarrée. Si le client a envoyé un jeton de session, c'est la session ayant cet identificateur qui est redémarrée et toutes les données qui lui sont liées sont placées dans le dictionnaire `$_SESSION`. Sinon, un nouveau jeton de session est créé.
- si le client a envoyé un paramètre *action* ayant la valeur "invalider", les données de la session sont marquées comme étant "à supprimer" pour le prochain échange. Elles ne seront pas sauvegardées sur le serveur à la fin de l'échange, contrairement à un échange normal.
- on récupère un compteur lié à la session dans le dictionnaire `$_SESSION`, ainsi que l'ID de session (`session_id()`).
- la page à envoyer au client est générée par le programme *cycledevie-p1.php*

La page *cycledevie-p1.php* affiche la page envoyée au client :

```
<html>
<head>
  <title>Gestion de sessions</title>
</head>
<body>
  <h3>Cycle de vie d'une session PHP</h3>
  <hr>
  <br>ID session : <?php echo $idSession ?>
  <br>compteur : <?php echo $compteur ?>
  <br><a href="cycledevie.php?action=invalid" >Invalidier la session</a>
  <br><a href="cycledevie.php" >Recharger la page</a>
</body>
</html>
```

On remarquera l'URL attachée à chacun des deux liens :

1. *cycledevie.php* pour recharger la page
2. *cycledevie.php?action=invalid* pour invalider la session. Dans ce cas, un paramètre *action=invalid* est joint à l'URL. Il sera récupéré par le programme serveur *cycledevie.php* par l'instruction *\$action=\$_GET["action"]*.

Rechargeons la page deux fois de suite :

Cycle de vie d'une session PHP

ID session : e4b12bd71499318dbd6d72d0c966d078

compteur : 2

[Invalidier la session](#)

[Recharger la page](#)

Le compteur a bien été incrémenté. L'ID de session n'a pas changé. Invalidons maintenant la session :

Cycle de vie d'une session PHP

ID session :

compteur : 3

[Invalidier la session](#)

[Recharger la page](#)

On voit qu'on a perdu l'ID de la session mais que le compteur a lui été incrémenté une nouvelle fois. Rechargeons la page :

Cycle de vie d'une session PHP

ID session : e4b12bd71499318dbd6d72d0c966d078

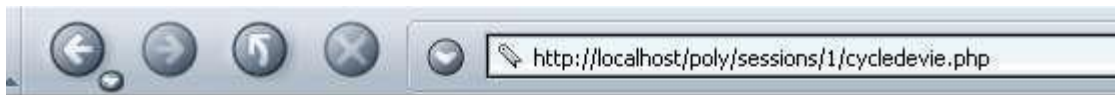
compteur : 0

[Invalidiser la session](#)

[Recharger la page](#)

On voit qu'on recommence avec le même ID de session que précédemment. Le compteur lui repart à zéro. La fonction `session_destroy()` n'a donc pas un effet immédiat. Les données de la session courante sont supprimées simplement pour l'échange client-serveur qui suit celui où est faite la suppression. L'ID session n'a pas changé, ce qui tendrait à montrer que `session_destroy()` ne démarre pas une nouvelle session avec création d'un nouvel ID. Le cookie du jeton de session a été renvoyé par le navigateur client et PHP a récupéré la session de ce jeton, session qui n'avait plus de données.

Les tests précédents ont été réalisés avec le navigateur Netscape configuré pour utiliser les cookies. Configurons-le maintenant pour ne pas utiliser ceux-ci. Cela signifie qu'il ne stockera ni ne renverra les cookies envoyés par le serveur. On s'attend alors à ce que les sessions ne fonctionnent plus. Essayons avec un premier échange :



Cycle de vie d'une session PHP

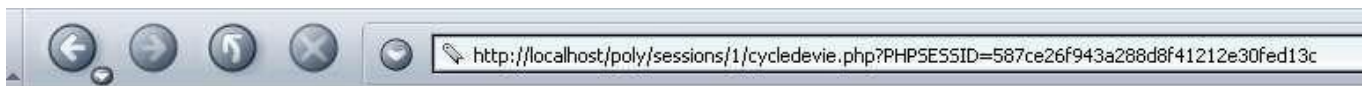
ID session : 587ce26f943a288d8f41212e30fed13c

compteur : 0

[Invalidiser la session](#)

[Recharger la page](#)

On a un ID de session, celui généré par `session_start()`. Rechargeons la page avec le lien :



Cycle de vie d'une session PHP

ID session : 587ce26f943a288d8f41212e30fed13c

compteur : 1

[Invalidiser la session](#)

[Recharger la page](#)

De façon surprenante, les résultats ci-dessus montrent qu'on a une session en cours et que le compteur est correctement géré. Comment cela est-il possible puisque les cookies ayant été désactivés, il n'y a plus d'échanges de jeton entre le serveur et le navigateur ? L'URL de la copie d'écran ci-dessus nous donne la réponse :

`http://localhost/poly/sessions/1/cycledevie.php?PHPSESSID=587ce26f943a288d8f41212e30fed13c`

C'est l'URL du lien "Recharger la page". Vérifions le code source de la page affichée par le navigateur :

```
<a href="cycledevie.php?action=invalid&PHPSESSID=587ce26f943a288d8f41212e30fed13c">Invalidier la session</a>
<a href="cycledevie.php?PHPSESSID=587ce26f943a288d8f41212e30fed13c">Recharger la page</a>
```

Rappelons que le code initial des deux liens dans la page *cycledevie-p1.php* est celui-ci :

```
<a href="cycledevie.php?action=invalid">Invalidier la session</a>
<a href="cycledevie.php">Recharger la page</a>
```

L'interpréteur PHP a donc de lui-même réécrit les URL des deux liens en y ajoutant le jeton de session. Ainsi celui-ci est bien transmis par le navigateur lorsque les liens sont activés. Ce qui explique que même sans les cookies activés, la session reste correctement gérée.

Exemple 3

Nous nous proposons d'écrire une application php qui serait cliente de l'application *compteur* précédente. Elle l'appellerait N fois de suite où N serait passé en paramètre. Notre but est de montrer un client web programmé et la façon de gérer le jeton de session. Notre point de départ sera un client web générique appelé de la façon suivante :

clientweb URL GET/HEAD

- URL : url demandée
- GET/HEAD : GET pour demander le code HTML de la page, HEAD pour se limiter aux seuls entêtes HTTP

Voici un exemple avec l'URL *http://localhost/poly/sessions/2/cycledevie.php*. Ce programme est celui déjà décrit, avec une légère différence :

```
// on fixe le chemin du cookie
session_set_cookie_params(0,"/poly/sessions/2");
// on démarre une session
session_start();
```

La fonction *session_set_cookie_params* permet de fixer certains paramètres du cookie qui va contenir le jeton de session. Le 1er paramètre est la durée de vie du cookie. Une durée de vie nulle signifie que le cookie est supprimé par le navigateur qui l'a reçu lorsque celui-ci est fermé. Le second paramètre est le chemin des URL auxquelles le navigateur doit renvoyer le cookie. Dans l'exemple ci-dessus, si le navigateur a reçu le cookie de la machine *localhost*, il renverra le cookie à toute URL se trouvant dans l'arborescence *http://localhost/poly/sessions/2/*.

```
dos>e:\php43\php.exe clientweb.php http://localhost/poly/sessions/2/cycledevie.php GET
HTTP/1.1 200 OK
Date: Wed, 09 Oct 2002 13:58:16 GMT
Server: Apache/1.3.24 (Win32)
Set-Cookie: PHPSESSID=48d5aaa0e99850b17c33a6e22d38e5c4;path=/poly/sessions/2
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
  <head>
    <title>Gestion de sessions</title>
  </head>
  <body>
    <h3>Cycle de vie d'une session PHP</h3>
    <hr>
    <br>ID session : 48d5aaa0e99850b17c33a6e22d38e5c4      <br>compteur : 0
    <br><a href="cycledevie.php?action=invalid&PHPSESSID=48d5aaa0e99850b17c33a6e22d38e5c4">Invalid
er la session</a>
    <br><a href="cycledevie.php?PHPSESSID=48d5aaa0e99850b17c33a6e22d38e5c4">Recharger la page</a>
  </body>
</html>
```

Le programme *clientweb* affiche tout ce qu'il reçoit du serveur. On voit ci-dessus la commande HTTP **Set-cookie** avec laquelle le serveur envoie un cookie à son client. Ici le cookie contient deux informations :

- **PHPSESSID** qui est le jeton de la session

- **path** qui définit l'URL à laquelle appartient le cookie. *path=/poly/sessions/2* indique au navigateur qu'il devra renvoyer le cookie au serveur à chaque fois qu'il demandera une URL commençant par */poly/sessions/2* de la machine qui lui a envoyé le cookie.
- un cookie peut définir également **une durée de validité**. Ici cette information est absente. Le cookie sera donc détruit à la fermeture du navigateur. Un cookie peut avoir une durée de validité de N jours par exemple. Tant qu'il est valide, le navigateur le renverra à chaque fois que l'une des URL de son domaine (*Path*) sera consultée. Prenons un site de vente en ligne de CD. Celui-ci peut suivre le cheminement de son client dans son catalogue et déterminer peu à peu ses préférences : la musique classique par exemple. Ces préférences peuvent être rangées dans un cookie ayant une durée de vie de 3 mois. Si ce même client revient au bout d'un mois sur le site, le navigateur renverra le cookie à l'application serveur. Celle-ci d'après les informations renfermées dans le cookie pourra alors adapter les pages générées aux préférences de son client.

Le code du client web suit.

```
<?php
// configuration
dl("php_curl.dll"); // bibliothèque CURL

// syntaxe : $0 URL GET
// il faut trois arguments
if($argc != 3){
    // msg d'erreur
    fputs(STDERR,"Syntaxe : $argv[0] URL GET/HEAD");
    // arrêt
    exit(1);
}

// le troisième argument doit être GET ou HEAD
$header=strtolower($argv[2]);
if($header!="get" && $header!="head"){
    // msg d'erreur
    fputs(STDERR,"Syntaxe : $argv[0] URL GET/HEAD");
    // arrêt
    exit(2);
}

// le premier argument est une URL
$url=strtolower($argv[1]);

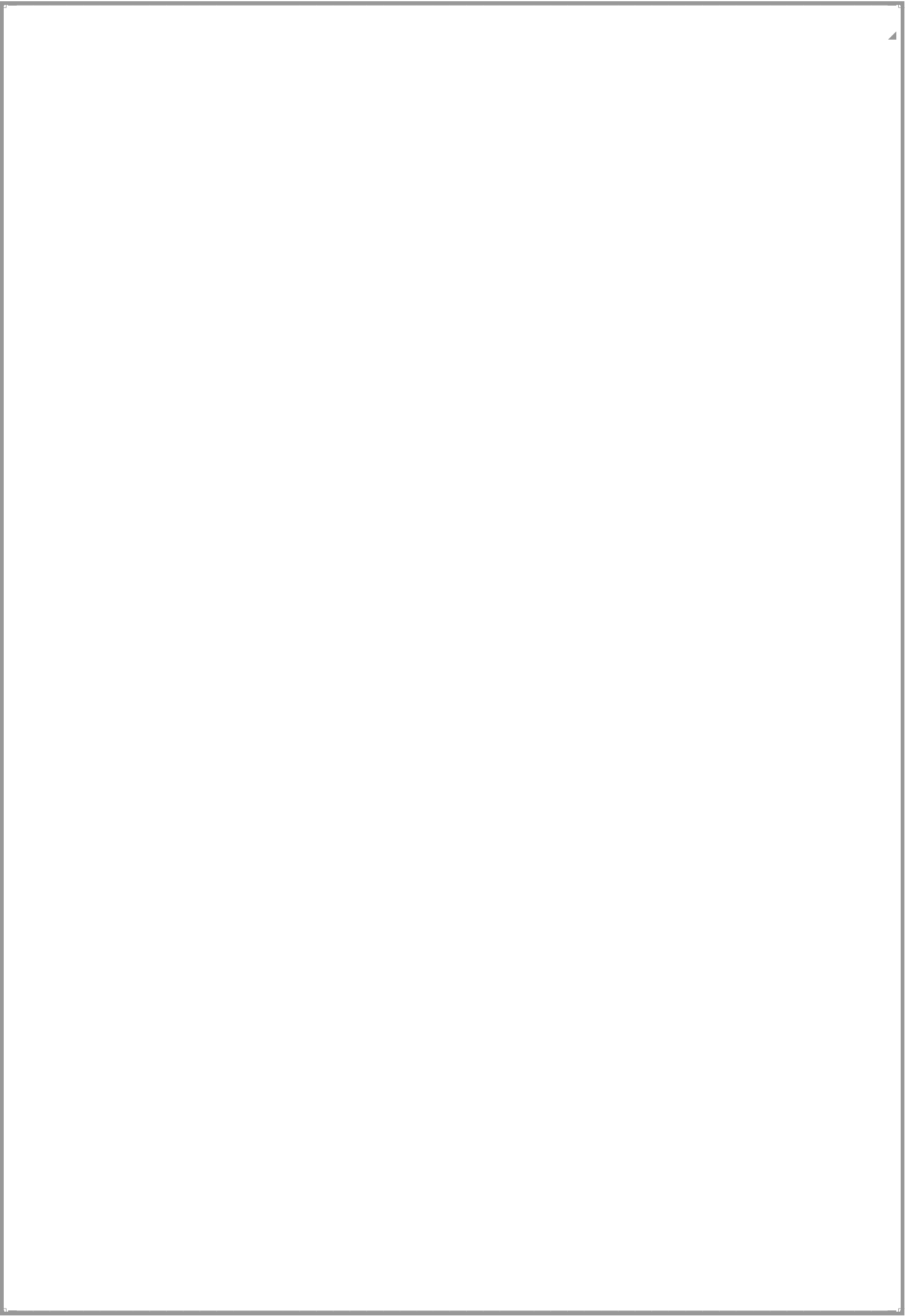
// préparation de la connexion
$connexion=curl_init($url);
// paramétrage de la connexion
curl_setopt($connexion, CURLOPT_HEADER, 1);
if($header=="head") curl_setopt($connexion, CURLOPT_NOBODY, 1);
// exécution de la connexion
curl_exec($connexion);
// fermeture de la connexion
curl_close($connexion);
// fin
exit(0);
?>
```

Le programme précédent utilise la bibliothèque CURL :

<code>\$connexion=curl_init(\$url)</code>	initialise un objet CURL avec l'URL à atteindre
<code>curl_setopt(\$connexion, option, valeur)</code>	fixe la valeur de certaines options de la connexion. Citons les deux utilisées dans le programme : CURLOPT_HEADER=1 : permet d'avoir les entêtes HTTP envoyées par le serveur CURLOPT_NOBODY=1 : permet d'ignorer le document envoyé par le serveur derrière les entêtes HTTP
<code>curl_exec(\$connexion)</code>	réalise la connexion à \$url avec les options demandées. Affiche sur l'écran tout ce que le serveur envoie
<code>curl_close(\$connexion)</code>	ferme la connexion

Le programme précédent est plutôt simple. Cependant la bibliothèque CURL ne permet pas de manipuler finement la réponse du serveur, par exemple l'analyser ligne par ligne. Le programme suivant fait la même chose que le précédent mais avec les fonctions réseau de base de PHP. Il sera le point de départ pour l'écriture d'un client à notre application *cycledevie.php*.

```
<?php
// syntaxe : $0 URL GET/HEAD
// il faut trois arguments
if($argc != 3){
```



```

// msg d'erreur
fputs(STDERR, "Syntaxe : $argv[0] URL GET/HEAD");
// arrêt
exit(1);
} //if

// connexion et affichage du résultat
$résultats=getURL($argv[1], $argv[2]);
if(isset($résultats->erreur)) {
    // erreur
    echo "L'erreur suivante s'est produite : $résultats->erreur\n";
} else {
    // affichage réponse du serveur echo
    $résultats->réponse;
} //if

// fin
exit(0);

//-----
function getURL ($URL, $header) {
    // se connecte à $URL
    // fait un GET ou un HEAD selon la valeur de header
    // la réponse du serveur forme le résultat de la fonction

    // analyse de l'URL
    $url=parse_url($URL);
    // le protocole if(strtolower($url["scheme"])!="http") {
    $résultats->erreur="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $résultats;
    } //if
    // la machine
    $hote=$url["host"];
    if(! isset($hote)) {
    $résultats->erreur="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $résultats;
    } //if
    // le port
    $port=$url["port"];
    if(! isset($port)) $port=80;
    // le chemin
    $chemin=$url["path"];
    // la requête if(isset($url["query"])) {
    $résultats->erreur="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $résultats;
    } //if

    // analyse de $header
    $header=strtoupper($header); if($header!="GET" &&
    $header!="HEAD") {
        // msg d'erreur
        $résultats->erreur="méthode [$header] doit être GET ou HEAD";
        // arrêt
        return $résultats;
    } //if

```

```

// ouverture d'une connexion sur le port $port de $hote
$connexion=fsockopen($hote,$port,&$errno,&$erreur);

```

```

// retour si erreur
if(! $connexion) {
    $résultats->erreur="Echec de la connexion au site ($hote,$port) :$erreur"; return
    $résultats;
} //if

```

```

// $connexion représente un flux de communication bidirectionnel
// entre le client (ce programme) et le serveur web contacté
// ce canal est utilisé pour les échanges de commandes et d'informations
// le protocole de dialogue est HTTP

```

```

// le client envoie la commande get pour demander l'URL /
// syntaxe get URL HTTP/1.0
// les entêtes (headers) du protocole HTTP doivent se terminer par une lignevide
fputs($connexion, "$header $chemin HTTP/1.0\n\n");

```

```

// le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
// ces données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
// jusqu'à la fermeture du canal
$résultats->réponse="";
while($ligne=fgets($connexion,10000))
    $résultats->réponse.=$ligne;

```

```
// le client ferme la connexion à son tour
fclose($connexion);
// retour
return $résultats;
} //getURL
?>
```

Commentons quelques points de ce programme :

- le programme admet deux paramètres :
 - une URL http dont on veut afficher le contenu à l'écran.
 - une méthode GET ou HEAD à utiliser selon que l'on veut seulement les entêtes HTTP (HEAD) ou également le corps du document associé à l'URL (GET).
- les deux paramètres sont passés à la fonction `getURL`. Celle-ci rend un objet `$résultats`. Celui-ci a un champ `erreurs` s'il y a une erreur, un champ `réponse` sinon. Le champ `erreur` sert à stocker un éventuel message d'erreur. Le champ `réponse` est la réponse du serveur web contacté.
- la fonction `getURL` analyse l'URL `$URL` avec la fonction `parse_url`. L'instruction `$url=parse_url($URL)` va créer le tableau associatif `$url` avec les clés éventuelles suivantes :
 - `scheme` : le protocole de l'URL (http, ftp, ..)
 - `host` : la machine de l'URL
 - `port` : le port de l'URL
 - `path` : le chemin de l'URL
 - `querystring` : les paramètres de l'URL
 Une url sera correcte si elle est de la forme `http://machine[:port][/chemin]`.
- le paramètre `$header` est également vérifié
- une fois les paramètres vérifiés et corrects, on crée une connexion TCP sur la machine (`$hote,$port`), puis on envoie la commande HTTP GET ou HEAD selon le paramètre `$header`.
- on lit alors la réponse du serveur qu'on met dans `$résultats->réponse`.

L'exécution du programme donne les résultats suivants :

```
dos>"e:\php43\php.exe" geturl.php http://localhost/poly/sessions/2/cycledevie.php get
HTTP/1.1 200 OK
Date: Wed, 09 Oct 2002 14:56:55 GMT
Server: Apache/1.3.24 (Win32)
Set-Cookie: PHPSESSID=ea0d2673811ed069e7289d86933a4c0a;path=/poly/sessions/2
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html

<html>
  <head>
    <title>Gestion de sessions</title>
  </head>
  <body>
    <h3>Cycle de vie d'une session PHP</h3>
    <hr>
    <br>ID session : ea0d2673811ed069e7289d86933a4c0a      <br>compteur : 0
    <br><a href="cycledevie.php?action=invalid&PHPSESSID=ea0d2673811ed069e7289d86933a4c0a">Invalid
er la session</a>
    <br><a href="cycledevie.php?PHPSESSID=ea0d2673811ed069e7289d86933a4c0a">Recharger la page</a>
  </body>
</html>
```

Le lecteur attentif aura noté que la réponse du serveur n'est pas la même selon le programme client utilisé. Dans le premier cas, le **serveur** avait envoyé un entête HTTP : **Transfer-Encoding: chunked**, entête qui n'a pas été envoyé dans le second cas. Cela vient du fait que le second **client** a envoyé l'entête HTTP : **get URL HTTP/1.0** qui demande une URL et indique qu'il travaille avec le protocole HTTP version 1.0 ce qui oblige le **serveur** à lui répondre avec ce même protocole. Or l'entête HTTP **Transfer-Encoding: chunked** appartient au protocole HTTP version 1.1. Aussi le serveur ne l'a-t-il pas utilisé dans sa réponse. Ceci nous montre que le premier **client** a lui fait sa requête en indiquant qu'il travaillait avec le protocole HTTP version 1.1.

Nous créons maintenant le programme **clientCompteur** appelé de la façon suivante :

clientCompteur URL N [JSESSIONID]

- URL : url de l'application *cycledevie*
- N : nombre d'appels à faire à cette application
- PHPSESSID : paramètre facultatif - jeton d'une session

Le but du programme est d'appeler N fois l'application *cycledevie.php* en gérant le cookie de session et en affichant à chaque fois la valeur du compteur renvoyée par le serveur. A la fin des N appels, la valeur de celui-ci doit être N-1. Voici un premier exemple d'exécution :

```
dos>"e:\php43\php.exe" clientCompteur2.php http://localhost/poly/sessions/2/cycledevie.php 3
--> GET /poly/sessions/2/cycledevie.php HTTP/1.1
--> Host: localhost:80
--> Connection: close
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 06:27:48 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Set-Cookie: PHPSESSID=2425e00d1d65c2bdcbafe1ce6244f7ea; path=/poly/sessions/2
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--

[Le compteur est égal à 0]

--> GET /poly/sessions/2/cycledevie.php HTTP/1.1
--> Host: localhost:80
--> Connection: close
--> Cookie: PHPSESSID=2425e00d1d65c2bdcbafe1ce6244f7ea
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 06:27:48 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--

[Le compteur est égal à 1]

--> GET /poly/sessions/2/cycledevie.php HTTP/1.1
--> Host: localhost:80
--> Connection: close
--> Cookie: PHPSESSID=2425e00d1d65c2bdcbafe1ce6244f7ea
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 06:27:48 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--

[Le compteur est égal à 2]
```

Le programme affiche :

- les entêtes HTTP qu'il envoie au serveur sous la forme --> entêteEnvoyé
- les entêtes HTTP qu'il reçoit sous la forme <-- entêteReçu
- la valeur du compteur après chaque appel

On voit que lors du premier appel :

- le client n'envoie pas de cookie
- le serveur envoie un (Set-Cookie:)

Pour les appels suivants :

- le client renvoie systématiquement le cookie qu'il a reçu du serveur lors du 1er appel. C'est ce qui va permettre au serveur de le reconnaître et d'incrémenter son compteur.
- le serveur lui, ne renvoie plus de cookie

Nous relançons le programme précédent en passant le jeton ci-dessus comme troisième paramètre :

```
dos>"e:\php43\php.exe" clientCompteur2.php http://localhost/poly/sessions/2/cycledevie.php 12425e00d1d65c2bdcba1ce6244f7ea
```

```
--> GET /poly/sessions/2/cycledevie.php HTTP/1.1
--> Host: localhost:80
--> Connection: close
--> Cookie: PHPSESSID=2425e00d1d65c2bdcba1ce6244f7ea
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 06:32:03 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--
```

```
[Le compteur est égal à 3]
```

On voit ici que dès le 1er appel du client, le serveur reçoit un cookie de session valide. On pointe peut-être là un trou de sécurité potentiel. Si je suis capable d'intercepter sur le réseau un jeton de session, je suis alors capable de me faire passer pour celui qui avait initié celle-ci. Dans notre exemple, le premier appel (sans jeton de session) représente celui qui initie la session (peut-être avec un login et mot de passe qui vont lui donner le droit d'avoir un jeton) et le second appel (avec jeton de session) représente celui qui a "piraté" le jeton de session du premier appel. Si l'opération en cours est une opération bancaire cela peut devenir ennuyeux...

Le code du client est le suivant :

```
<?php
```

```
// syntaxe : $0 URL N [PHPSESSID]
// il faut trois arguments
if($argc!=3 && $argc!=4) {
    // msg d'erreur
    fputs(STDERR,"Syntaxe : $argv[0] URL N [PHPSESSID]");
    // arrêt
    exit(1);
} //if
```

```
// récupération des paramètres
$URL=$argv[1];
$N=$argv[2];
$PHPSESSID=$argv[3];
```

```
// connexion et affichage du résultat
$résultats=getURL($URL,$N,$PHPSESSID);
if(isset($résultats->erreur)) {
    // erreur
    echo "L'erreur suivante s'est produite : $résultats->erreur\n";
}
```

```
// fin
exit(0);
```

```
//-----
function getURL($URL,$N,$PHPSESSID) {
    // se connecte à URL
    // fait un GET ou un HEAD selon la valeur de header
    // la réponse du serveur forme le résultat de la fonction

    // analyse de l'URL
    $url=parse_url($URL);
    // le protocole if(strtolower($url["scheme"])!="http") {
        $résultats->erreur="l'URL [$URL] n'est pas au format http://machine[:port]/chemin"; return $résultats;
    }
}
```

```

} // if
// la machine
$hote=$url["host"];
if(! isset($hote)){
    $résultats->erreur="l' URL [$URL] n'est pas au format http://machine[:port]/chemin"; return $résultats;
} // if
// le port
$port=$url["port"];
if(! isset($port)) $port=80;
// le chemin
$schemin=$url["path"];
// la requête if(isset($url["query"])){
    $résultats->erreur="l' URL [$URL] n'est pas au format http://machine[:port]/chemin"; return $résultats;
} // if

// vérification de $N
if (! preg_match("/^%d+$/", $N)) {
    // erreur
    $résultats->erreur="nombre [$N] erroné";
    // fin
    return $résultats;
} // if

// on fait les $N appels à $URL
for($i=0;$i<$N;$i++){

```

```

    // ouverture d'une connexion sur le port $port de $hote
    $connexion=fsockopen($hote,$port,&$errno,&$erreur);

```

```

// retour si erreur
if(! $connexion){
    $résultats->erreur="Echec de la connexion au site ($hote,$port) :$erreur"; return
    $résultats;
} // if

```

```

// $connexion représente un flux de communication bidirectionnel
// entre le client (ce programme) et le serveur web contacté
// ce canal est utilisé pour les échanges de commandes et d'informations
// le protocole de dialogue est HTTP

```

```

// le client envoie les entêtes HTTP
// get URL HTTP/1.1
envoie($connexion, "GET $schemin HTTP/1.1\n");
// host: hote:port
envoie($connexion, "Host: $hote:$port\n");
// Connection: close
envoie($connexion, "Connection: close\n");
// Cookie: $PHPSESSID
if($PHPSESSID) envoie($connexion, "Cookie: PHPSESSID=$PHPSESSID\n");
// ligne vide
envoie($connexion, "\n");

```

```

// le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
// ses données puis fermer le canal.

```

```

// Le client commence par lire les entêtes HTTP terminés par une ligne vide
$CHUNKED=0;

```

```

while(($ligne=fgets($connexion,10000)) && (($ligne=rtrim($ligne))!="")){
    // écho ligne
    echo "<-- $ligne\n";
    // recherche du jeton si on ne l'a pas encore trouvé
    if(! $PHPSESSID){
        // recherche de la ligne set-cookie
        if(preg_match("/^Set-Cookie: PHPSESSID=(.?.?);/i",$ligne,$champs)){
            // on a trouvé le jeton - on le mémorise
            $PHPSESSID=$champs[1];
        } // if
    } // if
    // recherche du mode de transfert du document
    if(! $CHUNKED){
        // recherche de la ligne Transfer-Encoding: chunked
        if(preg_match("/^Transfer-Encoding: chunked/i",$ligne,$champs)){
            // transfert par morceaux
            $CHUNKED=1;
        } // if
    } // if
} // ligne suivante

```

```

// écho ligne
echo "<-- $ligne\n";

```

```
// la lecture du document dépend de la façon dont il a été envoyé
if($CHUNKED) $document=getChunkedDoc($connexion);
else $document=getDoc($connexion);
```

```
// recherche du compteur dans le document
if(preg_match("/<br>compteur : (\d+)/i",$document,$champs)){
    // on a trouvé le compteur - on l'affiche
    echo "\n[Le compteur est égal à $champs[1]]\n\n";
}//if
```

```
// le client ferme la connexion
fclose($connexion);
}//for i
}//getUrl
//-----
```

```
function getDoc($connexion){
    // lecture du document sur $connexion
    $doc="";
    while($ligne=fread($connexion,10000))
        $doc.=$ligne;
    // fin
    return $doc;
}//getDoc
```

```
//-----
function getChunkedDoc($connexion){
    // lecture du document sur $connexion
    // ce document est envoyé par morceaux sous la forme
    // nombre de caractères du morceau en hexadécimal
    // suite du morceau
    // ligne vide

    // on lit la taille du morceau dans la 1ère ligne
    $taille=hexdec(rtrim(fgets($connexion,10000)));
    // on lit le document qui suit
    $doc="";
    while($taille!=0){
        // lecture morceau de $taille caractères
        $doc.=fread($connexion,$taille);
        // ligne vide
        fgets($connexion,10000);
        // morceau suivant
        // on lit la taille du morceau
        $taille=hexdec(rtrim(fgets($connexion,10000)));
    }//while
    // c'est fini
    return $doc;
}// getChunkedDoc
```

```
//-----
function envoie($flux,$msg){
    // envoie $msg sur $flux
    fwrite($flux,$msg);
    // echo écran echo
    "--> $msg";
}//envoie
?>
```

Décortiquons les points importants de ce programme :

- on doit faire N échanges client-serveur. C'est pourquoi ceux-ci sont dans une boucle

```
for($i=0;$i<$N;$i++){
```

- à chaque échange, le client ouvre une connexion TCP-IP avec le serveur. Une fois celle-ci obtenue, il envoie au serveur les entêtes HTTP de sa requête :

```
// le client envoie les entêtes HTTP
// get URL HTTP/1.1
envoie($connexion, "GET $chemin HTTP/1.1\r\n");
// host: hote:port
envoie($connexion, "Host: $hote:$port\r\n");
// Connection: close
envoie($connexion, "Connection: close\r\n");
// Cookie: $PHPSESSID
if($PHPSESSID) envoie($connexion, "Cookie: PHPSESSID=$PHPSESSID\r\n");
```

```
// ligne vide
envoi($connexion, "¥n");
```

Si le jeton PHPSESSID est disponible, il est envoyé sous la forme d'un cookie, sinon il ne l'est pas. On notera que le client a indiqué qu'il travaillait avec le protocole HTTP/1.1. Ce qui explique qu'ultérieurement, le serveur va lui envoyer l'entête HTTP : **Transfer-Encoding: chunked** qui appartient au protocole HTTP/1.1 mais pas au protocole HTTP/1.0.

- Une fois sa requête envoyée, le client attend la réponse du serveur. Il commence par exploiter les entêtes HTTP de cette réponse. Il recherche dans celle-ci deux lignes :

Cookie :

Transfer-Encoding: chunked

La ligne **Cookie** : est l'entête HTTP qui contient le jeton de session PHPSESSID. Le client doit le récupérer pour le renvoyer au serveur lors de l'échange suivant. La ligne **Transfer-Encoding: chunked**, si elle est présente, indique que le serveur va envoyer un document par morceaux. Chaque morceau est alors envoyé au client sous la forme suivante :

[nombre de caractères du document en hexadécimal]

[document]

[ligne vide]

Si la ligne *Transfer-Encoding: chunked* n'est pas présente, le document est envoyé en une seule fois derrière la ligne vide des entêtes HTTP. Donc, selon la présence ou non de cette ligne, le mode de réception du document va différer. Le code d'exploitation des entêtes HTTP est le suivant :

```
// Le client commence par lire les entêtes HTTP terminés par une ligne vide
$CHUNKED=0;
while(($ligne=fgets($connexion, 10000)) && (($ligne=rtrim($ligne))!="")) {
    // écho ligne
    echo "<— $ligne¥n";
    // recherche du jeton si on ne l'a pas encore trouvé if(!
    $PHPSESSID) {
        // recherche de la ligne set-cookie
        if(preg_match("/^Set-Cookie: PHPSESSID=(. ?):/i", $ligne, $champs)) {
            // on a trouvé le jeton - on le mémorise
            $PHPSESSID=$champs[1];
        }//if
    }//if
    // recherche du mode de transfert du document if(!
    $CHUNKED) {
        // recherche de la ligne Transfer-Encoding: chunked if(preg_match("/^Transfer-
        Encoding: chunked/i", $ligne, $champs)) {
            // transfert par morceaux
            $CHUNKED=1;
        }//if
    }//if
} //ligne suivante
```

- lorsque le jeton aura été trouvé une première fois, il ne sera plus cherché lors des appels suivants au serveur. Lorsque les entêtes HTTP de la réponse ont été traités, on passe au document qui suit les entêtes HTTP. Celui-ci est lu différemment selon son mode de transfert :

```
// la lecture du document dépend de la façon dont il a été envoyé if($CHUNKED)
$document=getChunkedDoc($connexion);
else $document=getDoc($connexion);
```

- Dans le document *\$document* reçu on cherche la ligne qui donne la valeur du compteur. Cette recherche est faite avec une expression régulière :

```
// recherche du compteur dans le document if(preg_match("<br>compteur :
(¥d+)/i", $document, $champs)) {
    // on a trouvé le compteur - on l'affiche
    echo "¥n[Le compteur est égal à $champs[1]]¥n¥n";
} //if
```

- Dans le cas où le serveur envoie le document en une fois, la réception de celui-ci est simple :

```
//-----
function getDoc($connexion) {
    // lecture du document sur $connexion
    $doc=""; while($ligne=fgets($connexion, 10000))
        $doc.=$ligne;
    // fin
```

```
return $doc;
} //getDoc
```

- Dans le cas où le serveur envoie le document en plusieurs morceaux, la lecture de celui-ci est plus complexe :

```
function getChunkedDoc($connexion) {
    // lecture du document sur $connexion
    // ce document est envoyé par morceaux sous la forme
    // nombre de caractères du morceau en hexadécimal
    // suite du morceau

    // on lit la taille du morceau dans la 1ère ligne
    $taille=hexdec(rtrim(fgets($connexion,10000)));
    // on lit le document qui suit
    $doc="";
    while($taille!=0) {
        // lecture morceau de $taille caractères
        $doc.=fread($connexion,$taille);
        // ligne vide
        fgets($connexion,10000);
        // morceau suivant
        // on lit la taille du morceau
        $taille=hexdec(rtrim(fgets($connexion,10000)));
    } //while
    // c'est fini
    return $doc;
} // getChunkedDoc
```

Rappelons qu'un morceau de document est envoyé sous la forme

```
[taille]
[partie de document]
[ligne vide]
```

On commence donc par lire la taille du document. Celle-ci connue, on demande à la fonction *fread* de lire *\$taille* caractères dans le flux *\$connexion* puis la ligne vide qui suit. On répète cela jusqu'à ce que le serveur envoie l'information qu'il va envoyer un document de taille 0.

Exemple 4

Dans l'exemple précédent, le client web renvoie le jeton sous la forme d'un cookie. Nous avons vu qu'il pouvait aussi le renvoyer au sein même de l'URL demandée sous la forme *URL;PHPSESSID=xxx*. Vérifions-le. Le programme *clientCompteur.php* est transformé en *clientCompteur2.php* et modifié de la façon suivante :

```
....
// le client envoie les entêtes HTTP
// get URL HTTP/1.1

if($PHPSESSID)
    envoie($connexion, "GET $chemin?PHPSESSID=$PHPSESSID HTTP/1.1\n");
else envoie($connexion, "GET $chemin HTTP/1.1\n");

// host: hote:port
envoie($connexion, "Host: $hote:$port\n");
// Connection: close
envoie($connexion, "Connection: close\n");
// ligne vide
envoie($connexion, "\n");
....
```

Le client demande donc l'URL du compteur par **GET URL;PHPSESSID=xx HTTP/1.1** et n'envoie plus de cookie. C'est la seule modification. Voici les résultats d'un premier appel :

```
dos>"e:\php43\php.exe" clientCompteur2.php http://localhost/poly/sessions/2/cycledevie.php 2

--> GET /poly/sessions/2/cycledevie.php HTTP/1.1
--> Host: localhost:80
--> Connection: close
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 07:21:19 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Set-Cookie: PHPSESSID=573212ba82303d7903caf8944ee7a86f; path=/poly/sessions/2
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
```

```

<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--

[Le compteur est égal à 0]

--> GET /poly/sessions/2/cycledevie.php?PHPSESSID=573212ba82303d7903caf8944ee7a86f HTTP/1.1
--> Host: localhost:80
--> Connection: close
-->
<-- HTTP/1.1 200 OK
<-- Date: Thu, 10 Oct 2002 07:21:19 GMT
<-- Server: Apache/1.3.24 (Win32)
<-- Expires: Thu, 19 Nov 1981 08:52:00 GMT
<-- Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
<-- Pragma: no-cache
<-- Connection: close
<-- Transfer-Encoding: chunked
<-- Content-Type: text/html
<--

[Le compteur est égal à 1]

```

Lors du premier appel le client demande l'URL sans jeton de session. Le serveur lui répond en lui envoyant le jeton. Le client réinterroge alors la même URL en adjoignant le jeton reçu à celle-ci. On voit que le compteur est bien incrémenté preuve que le serveur a bien reconnu qu'il s'agissait de la même session.

Exemple 5

Cet exemple montre une application composée de trois pages qu'on appellera *page1*, *page2* et *page3*. L'utilisateur doit les obtenir dans cet ordre :

- **page1** est un formulaire demandant une information : un nom
- **page2** est un formulaire obtenu en réponse à l'envoi du formulaire de **page1**. Il demande une seconde information : un âge
- **page3** est un document HTML qui affiche le nom obtenu par **page1** et l'âge obtenu par **page2**.

Il y a trois échanges client-serveur :

- au premier échange le formulaire **page1** est demandé par le client et envoyé par le serveur
- au second échange le client envoie le formulaire **page1** (nom) au serveur. Il reçoit en retour le formulaire **page2** ou de nouveau le formulaire **page1** s'il était erroné.
- au troisième échange le client envoie le formulaire **page2** (âge) au serveur. Il reçoit en retour le formulaire **page3** ou de nouveau le formulaire **page2** s'il était erroné. Le document **page3** affiche le nom et l'âge. Le nom a été obtenu par le serveur au second échange et "oublié" depuis. On utilise une session pour enregistrer le nom à l'échange 2 afin qu'il soit disponible lors de l'échange 3.

La page **page1** obtenue au premier échange est la suivante :



Page 1/3

Votre nom

On remplit le champ du nom :



Page 1/3

Votre nom

On utilise le bouton [Suite] et on obtient alors la page **page2** suivante :



Page 2/3

Nom martin

Votre âge

On remplit le champ de l'âge :



Page 2/3

Nom martin

Votre âge

On utilise le bouton [Suite] et on obtient alors la page **page3** suivante :



Page 3/3

Nom martin

Votre âge 27

Lorsqu'on soumet la page **page1** au serveur, celui-ci peut la renvoyer avec un code d'erreur si le nom est vide :



Page 1/3

Votre nom

Les erreurs suivantes se sont produites

- ♦ Vous n'avez pas indiqué de nom

Lorsqu'on soumet la page **page2** au serveur, celui-ci peut la renvoyer avec un code d'erreur si l'âge est invalide :



Page 2/3

Nom

Votre âge

Les erreurs suivantes se sont produites

- ♦ âge incorrect

L'application est composée de six programmes :

etape1. php	fait appel à <i>page1.php</i>
page1. php	affiche page1. Le formulaire de page1 est traité par <i>etape2.php</i> .
etape2. php	traite les valeurs du formulaire de page1. S'il y a des erreurs, page1 est réaffichée par <i>page1.php</i> sinon c'est page2 qui est affichée par <i>page2.php</i> .
page2. php	affiche page2. Le formulaire de page2 est traité par <i>etape3.php</i> .
etape3. php	traite les valeurs du formulaire de page2. S'il y a des erreurs, page2 est réaffichée par <i>page2.php</i> sinon c'est page3 qui est affichée par <i>page3.php</i> .
page3. php	affiche page3.

L'étape 1 de l'application est traitée par le programme *etape1.php* suivant :

```
<?php
// etape1. php
```



```
// configuration ini_set("register_globals","off");
ini_set("display_errors","off");

// démarrage session
session_start();
$_SESSION["session"]=""; //raz variable session

// préparation page1
$requête->nom="";
$requête->erreurs=array();
// affichage page1
include "page1.php";
// fin
exit(0);
?>
```

Notons les points suivants :

- l'application a besoin d'un suivi de session. Aussi chaque étape de celle-ci démarre-t-elle une session.
- les informations de session à conserver le seront dans un objet *\$session*.
- les informations nécessaires à l'affichage des différentes (trois) pages de l'application seront mises dans un objet *\$requête*.

Le programme *page1.php* affiche les informations contenues dans *\$requête* :

```
<? // page1.php ?>
<html>
<head>
<title>page 1</title>
</head>
<body>
<h3>Page 1/3</h3>
<form name="frmNom" method="POST" action="etape2.php">
<table>
<tr>
<td>Votre nom</td>
<td><input type="text" name="nom" value="<? echo $requête->nom ?>"></td>
</tr>
</table>
<input type="submit" value="Suite">
</form>
<? // erreurs ?
if (count($requête->erreurs) != 0) {
?>
<hr>
<font color="red">
Les erreurs suivantes se sont produites
<ul>
<? for ($i=0;$i<count($requête->erreurs);$i++) { ?>
<li><? echo $requête->erreurs[$i] ?>
<? } //for ?>
</ul>
<? } //if ?>
</body>
</html>
```

- la page reçoit un objet *\$requête* contenant deux champs : *nom* et *erreurs*. Elle affiche la valeur de ces deux champs.
- elle présente de plus un formulaire. Les valeurs de celui-ci (nom) sont envoyées par la méthode POST au programme *etape2.php* :

```
<form name="frmNom" method="POST" action="etape2.php">
```

L'application *etape2.php* est chargée de traiter les valeurs du formulaire de page1 et de réafficher page1 s'il y a des erreurs (nom erroné) sinon d'afficher page2 pour avoir l'âge.

```
<?php
// etape2.php

// configuration
ini_set("register_globals","off");
ini_set("display_errors","off");

// démarrage session
session_start();

// normalement, on doit avoir un paramètre nom
// enregistré dans la requête
$requête->nom=$_POST["nom"];
```

```
// si pas de paramètres, on envoie page1 sans erreurs if (!
isset($requête->nom)) {
    $requête->nom="";
    $requête->erreurs=array();
    include "page1.php"; exit(0);
} //if
```

```
// si le paramètre nom est bien là - on vérifie sa validité
$page=calculerPage($requête);
```

```
// y-a-t-il eu des erreurs ?
if(count($page->erreurs)!=0){
    // page 1 avec erreurs
    $requête->erreurs=$page->erreurs;
    include "page1.php";
    exit(0);
} //if
```

```
// pas d'erreur - on mémorise le nom dans la session
unset($session);
$session->nom=$requête->nom;
$_SESSION["session"]=$session;
```

```
// affichage page 2
$requête->age="";
$requête->erreurs=array();
include "page2.php";
```

```
// fin
exit(0);
```

```
// -----calculerPage
function calculerPage($requête){
    // vérifie la validité de la requête $requête
    // rend un tableau d'erreurs dans $page->erreurs

    // au départ, pas d'erreurs
    $page->erreurs=array();
    // le nom ne doit pas être vide
    if (preg_match("/^\s*$/",$requête->nom)){
        $page->erreurs[]="Vous n'avez pas indiqué de nom";
    }
    // retour page
    return $page;
} //calculerPage
```

```
?>
```

- *etape2* commence par vérifier qu'il a bien le paramètre *nom* attendu. Si ce n'est pas le cas, il fait réafficher une page1 vide. Ce cas est possible si *etape2* est appelée directement par un client qui ne lui passe pas de paramètres.
- si le paramètre *nom* est présent, on vérifie sa validité. Cela se fait par l'intermédiaire d'une procédure appelée *calculerPage* dont le rôle est de produire un objet *\$page* avec un champ *erreurs* qui est un tableau d'erreurs. Une seule erreur est possible mais on a voulu montrer qu'on pouvait gérer une liste d'erreurs.
- s'il y a des erreurs, la page *page1* est réaffichée accompagnée de la liste des erreurs.
- s'il n'y a pas d'erreurs, le nom est mémorisé dans l'objet *\$session* qui mémorise les données liées à la session courante. Puis la page *page2* est affichée.

Le programme *page2.php* affiche page2 :

```
<? // page2. php ?>
<html>
<head>
<title>page 2</title>
</head>
<body>
<h3>Page 2/3</h3>

<form name="frmAge" method="POST" action="etape3.php">
<table>
<tr>
<td>Nom</td>

<td><font color="green"><? echo $requête->nom ?></font></td>
</tr>
<tr>
```

```

<td>Votre âge</td>
<td><input type="text" name="age" size="3" value="<? echo $requête->age ?>"></td>
</tr>
</table>
<input type="submit" value="Suite">
</form>
<? // erreurs ?
if (count($requête->erreurs) != 0) {
?>
<hr>
<font color="red">
  Les erreurs suivantes se sont produites
  <ul>
    <? for ($i=0;$i<count($requête->erreurs);$i++) { echo
      "<li>". $requête->erreurs[$i];
    } //for
  ?>
  </ul>
</font>
<? } ?>
</body>
</html>

```

Le principe de cette page est très analogue à celui de *page2.php*. Elle affiche le contenu d'un objet *\$requête* contenant les champs *nom*, *age* et *erreurs*. Elle affiche un formulaire dont les valeurs seront traitées par *etape3.php*.

```
<form name="frmAge" method="POST" action="etape3.php">
```

Le programme *etape3.php* traite donc les valeurs du formulaire de **page2** réduites ici à l'âge :

```

<?php
// etape3.php

// configuration
ini_set("register_globals", "off");
ini_set("display_errors", "off");

// démarrage session
session_start();

// on récupère les paramètres nom et age
$requête->age=$_POST["age"];
$session=$_SESSION["session"];
$requête->nom=$session->nom;

// normalement, on doit avoir un nom et un âge
if (!isset($requête->age) || !isset($requête->nom)) {
  // si appel incorrect, on envoie page1
  $_SESSION["session"]=""; // par précaution
  $requête->nom="";
  $requête->erreurs=array();
  include "page1.php"; exit(0);
} //if

// le paramètre age est bien là - on vérifie sa validité
$page=calculerPage($requête);

// y-a-t-il eu des erreurs ?
if(count($page->erreurs) != 0) {
  // page 2 avec erreurs
  $requête->erreurs=$page->erreurs;
  include "page2.php";
  exit(0);
} //if

// pas d'erreur - mémorisation de l'âge dans la session
$session->age=$requête->age;
$_SESSION["session"]=$session;

// affichage page 3
include "page3.php";
// fin
exit(0);

// -----calculerPage
function calculerPage($requête){
  // vérifie la validité de la requête $requête
  // rend un tableau d'erreurs dans $page->erreurs

  // au départ, pas d'erreurs

```

```

$page->erreurs=array();
// l'âge doit avoir un format valide
if (! preg_match("/^\s*\d{1,3}\s*$/",$requête->age)){
    $page->erreurs[]="âge incorrect";
}
// retour page
return $page;
} //calculerPage
?>

```

- le programme commence par récupérer le nom dans la session (provenant de page1) et l'âge dans le formulaire de page2. S'il manque l'une de ces informations, la page **page1** est affichée.
- la validité de l'âge est ensuite vérifiée. Si l'âge est incorrect, la page **page2** est réaffichée avec une liste d'erreurs. Si l'âge est correct, c'est la page **page3** qui est affichée. Celle-ci se contente d'afficher les deux valeurs (nom,age) obtenues par les deux formulaires (**page1,page2**).

```

<? // page3. php ?>
<html>
<head>
<title>page 3</title>
</head>
<body>
<h3>Page 3/3</h3>
<table>
<tr>
<td>Nom</td>
<td><font color="green"><? echo $requête->nom ?></font></td>
</tr>
<tr>
<td>Votre âge</td>
<td><font color="green"><? echo $requête->age ?></font></td>
</tr>
</table>
</body>
</html>

```

3.Exemples

Application Impôts : Introduction

Nous introduisons ici l'application IMPOTS qui sera utilisée à de nombreuses reprises par la suite. Il s'agit d'une application permettant de calculer l'impôt d'un contribuable. On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer :

- on calcule le nombre de parts du salarié $\text{nbParts} = \text{nbEnfants} / 2 + 1$ s'il n'est pas marié, $\text{nbEnfants} / 2 + 2$ s'il est marié, où nbEnfants est son nombre d'enfants.
- s'il a au moins trois enfants, il a une demi-part de plus
- on calcule son revenu imposable $R = 0.72 * S$ où S est son salaire annuel
- on calcule son coefficient familial $QF = R / \text{nbParts}$
- on calcule son impôt I . Considérons le tableau suivant :

12620.0	0	0
13190	0.05	631
15640	0.1	1290.5
24740	0.15	2072.5
31810	0.2	3309.5
39970	0.25	4900
48360	0.3	6898.5
55790	0.35	9316.5
92970	0.4	12106
127860	0.45	16754.5
151250	0.50	23147.5
172040	0.55	30710
195000	0.60	39312
0	0.65	49062

Chaque ligne a 3 champs. Pour calculer l'impôt I , on recherche la première ligne où $QF \leq \text{champ1}$. Par exemple, si $QF = 23000$ on trouvera la ligne

24740 0.15 2072.5

L'impôt I est alors égal à $0.15 * R - 2072.5 * \text{nbParts}$. Si QF est tel que la relation $QF \leq \text{champ1}$ n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

0 0.65 49062

ce qui donne l'impôt $I = 0.65 * R - 49062 * \text{nbParts}$.

Les données définissant les différentes tranches d'impôt sont dans une base de données ODBC-MySQL. MySQL est un SGBD du domaine public utilisable sur différentes plate-formes dont Windows et Linux. Avec ce SGBD, une base de données **dbimpots** a été créée avec dedans une unique table appelée **impots**. L'accès à la base est contrôlé par un login/motdepasse ici **admimpots/mdpimpots**. La copie d'écran suivante montre comment utiliser la base **dbimpots** avec MySQL :

```
dos>mysql -u admimpots -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 18 to server version:3.23.49-max-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use dbimpots;
Database changed

mysql> show tables;
+-----+
| Tables_in_dbimpots |
+-----+
| impots              |
+-----+
1 row in set (0.00 sec)

mysql> describe impots;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id     | int  |      |     |          |       |
| salaire | float |      |     |          |       |
| nbEnfants | int  |      |     |          |       |
| nbParts | int  |      |     |          |       |
| R       | float |      |     |          |       |
| QF      | float |      |     |          |       |
| impot   | float |      |     |          |       |
+-----+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| limites | double | YES | | NULL | |
| coeffR  | double | YES | | NULL | |
| coeffN  | double | YES | | NULL | |
+-----+-----+-----+-----+
```

3 rows in set (0.02 sec)

```
mysql> select * from impots;
```

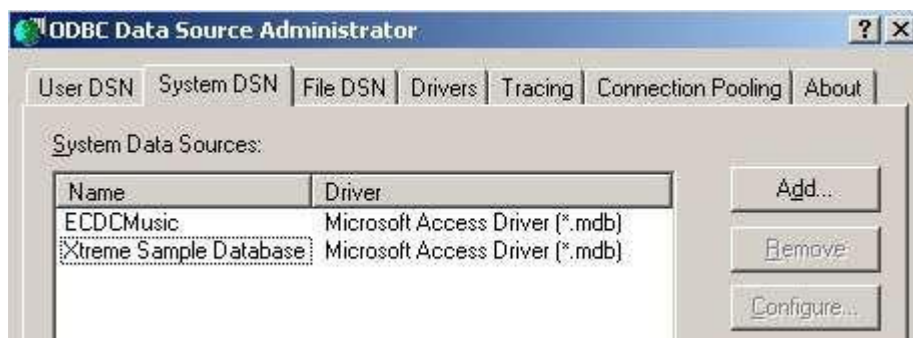
```
+-----+-----+-----+
| limites | coeffR | coeffN |
+-----+-----+-----+
| 12620 | 0 | 0 |
| 13190 | 0.05 | 631 |
| 15640 | 0.1 | 1290.5 |
| 24740 | 0.15 | 2072.5 |
| 31810 | 0.2 | 3309.5 |
| 39970 | 0.25 | 4900 |
| 48360 | 0.3 | 6898 |
| 55790 | 0.35 | 9316.5 |
| 92970 | 0.4 | 12106 |
| 127860 | 0.45 | 16754 |
| 151250 | 0.5 | 23147.5 |
| 172040 | 0.55 | 30710 |
| 195000 | 0.6 | 39312 |
| 0 | 0.65 | 49062 |
+-----+-----+-----+
```

14 rows in set (0.00 sec)

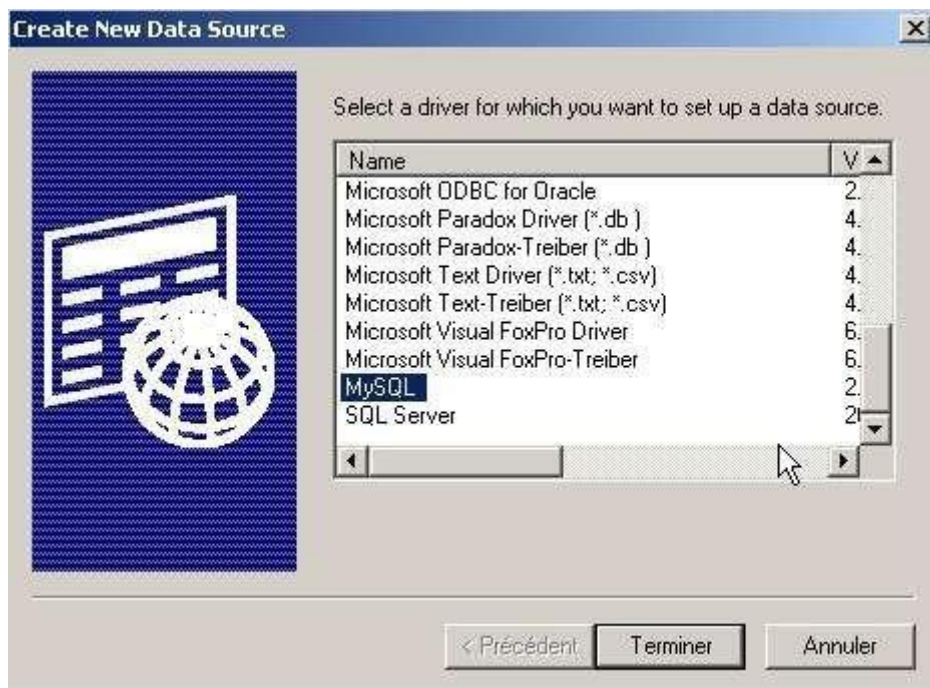
```
mysql>quit
```

La base de données *dbimpots* est transformée en source de données ODBC de la façon suivante :

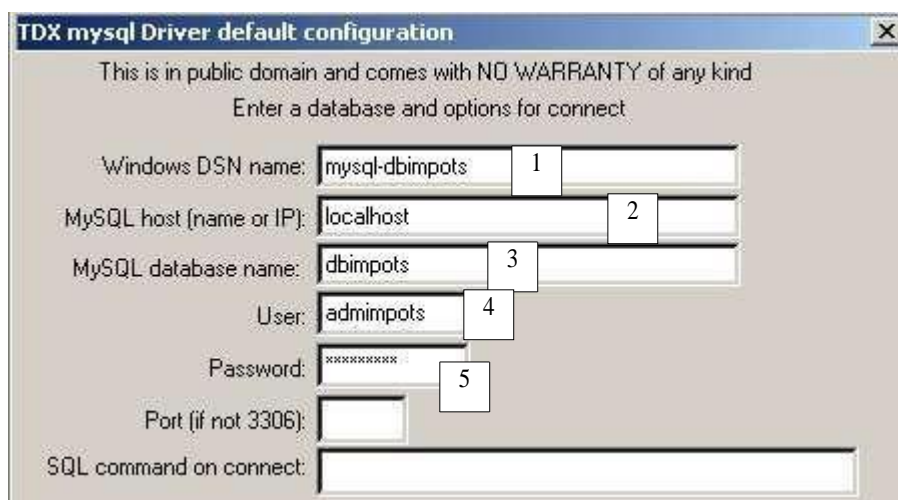
- on lance le gestionnaire des sources de données ODBC 32 bits



- on utilise le bouton [Add] pour ajouter une nouvelle source de données ODBC



- on désigne le pilote MySQL et on fait [Terminer]



- le pilote MySQL demande un certain nombre de renseignements :
 - 1 le nom DSN à donner à la source de données ODBC - peut-être quelconque
 - 2 la machine sur laquelle s'exécute le SGBD MySQL - ici *localhost*. Il est intéressant de noter que la base de données pourrait être une base de données distante. Les applications locales utilisant la source de données ODBC ne s'en apercevraient pas. Ce serait le cas notamment de notre application PHP.
 - 3 la base de données MySQL à utiliser. MySQL est un SGBD qui gère des bases de données relationnelles qui sont des ensembles de tables reliées entre-elles par des relations. Ici, on donne le nom de la base gérée.
 - 4 le nom d'un utilisateur ayant un droit d'accès à cette base
 - 5 son mot de passe

Application Impôts : la classe ImpotsDSN

Notre application s'appuiera sur la classe PHP ImpotsDSN qui aura les attributs, constructeur et méthode suivante :

```
// attributs
var $limites; // tableau des limites
```

```
var $coeffR; // tableau des coeffR
var $coeffN; // tableau des coeffN var
$erreurs; // tableau des erreurs
```

```
// constructeur
function ImpotsDSN($impots) {...}
```

```
// méthode
function calculer($personne) {...}
```

- nous avons vu dans l'exposé du problème que nous avons besoin de trois séries de données

12620.0	0	0
13190	0.05	631
15640	0.1	1290.5
24740	0.15	2072.5
31810	0.2	3309.5
39970	0.25	4900
48360	0.3	6898.5
55790	0.35	9316.5
92970	0.4	12106
127860	0.45	16754.5
151250	0.50	23147.5
172040	0.55	30710
195000	0.60	39312
0	0.65	49062

La 1ère colonne sera placée dans l'attribut **\$limites** de la classe **ImotsDSN**, la seconde dans l'attribut **\$coeffR**, la troisième dans **\$coeffN**. Ces trois attributs sont initialisés par le constructeur de la classe. Celui-ci va chercher les données dans une source de données ODBC. Diverses erreurs peuvent se produire lors de la construction de l'objet. Ces erreurs sont rapportées dans l'attribut **\$erreurs** sous la forme d'un tableau de messages d'erreurs.

- le constructeur reçoit en paramètre un dictionnaire **\$impots** avec les champs suivants :

```
// dsn : nom DSN de la source ODBC de données contenant les valeurs des tableaux limites, coeffR, coeffN
// user : nom d'un utilisateur ayant un droit de lecture sur la source ODBC
// pwd : son mot de passe
// table : nom de la table contenant les valeurs (limites, coeffR, coeffN)
// limites : nom de la colonne contenant les valeurs limites
// coeffR : nom de la colonne contenant les valeurscoeffR
// coeffN : nom de la colonne contenant les valeurscoeffN
```

Le paramètre du constructeur apporte toutes les informations nécessaires pour aller lire les données dans la source de données ODBC

- une fois l'objet **ImpotsDSN** construit, les utilisateur de la classe pourront appeler la méthode *calculerImpots* de l'objet. Celui-ci reçoit en paramètre un dictionnaire *\$personne* :

```
// $personne["marié"] : oui, non
// $personne["enfants"] : nombre d'enfants
// $personne["salaire"] : salaire annuel
```

La classe complète est la suivante :

```
<?php
```

```
// définition d'une classe objImpots
class ImpotsDSN{
```

```
    // attributs : les 3 tableaux de données
    var $limites; // tableau des limites
    var $coeffR; // tableau des coeffR
    var $coeffN; // tableau des coeffN
    var $erreurs; // tableau des erreurs
```

```
    // constructeur
```

```
    function ImpotsDSN($impots){
```

```
        // $impots : dictionnaire contenant les champs suivants
```

```
        // dsn : nom DSN de la source ODBC de données contenant les valeurs des tableaux limites, coeffR,
coeffN
```

```
        // user : nom d'un utilistaeur ayant un droit de lecture sur la source ODBC
        // pwd : son mot de passe
        // table : nom de la table contenant les valeurs (limites, coeffR, coeffN)
        // limites : nom de la colonne contenant les valeurs limites
```



```

// coeffR : nom de la colonne contenant les valeurscoeffR
// coeffN : nom de la colonne contenant les valeurscoeffN

// au départ pas d'erreurs
$this->erreurs=array();

// vérification de l'appel
if (! isset($impots[dsn]) || ! isset($impots[user]) || ! isset($impots[pwd]) || ! isset($impots [table]) ||
! isset($impots[limites]) || ! isset($impots[coeffR]) || ! isset($impots[coeffN])){
    // erreur
    $this->erreurs[]="Appel incorrect";
    // fin
    return;
} //if

// ouverture de la base DSN
$connexion=odbc_connect($impots[dsn], $impots[user], $impots[pwd]);
// erreur ?
if(! $connexion){
    // erreur
    $this->erreurs[]="Impossible d'ouvrir la base DSN [$impots[dsn]] (".odbc_error().")";
    // fin
    return;
} //if

// émission d'une requête sur la base
$requête=odbc_prepare($connexion,"select $impots[limites], $impots[coeffR], $impots[coeffN] from
$impots[table]");
if(! odbc_execute($requête)){
    // erreur
    $this->erreurs[]="Impossible d'exploiter la base DSN [$impots[dsn]] (".odbc_error().")";
    // fin
    odbc_close($connexion);
    return;
} //if

// exploitation des résultats de la requête
$this->limites=array();
$this->coeffR=array();
$this->coeffN=array();
while(odbc_fetch_row($requête)){
    // une ligne de plus
    $this->limites[]=odbc_result($requête, $impots[limites]);
    $this->coeffR[]=odbc_result($requête, $impots[coeffR]);
    $this->coeffN[]=odbc_result($requête, $impots[coeffN]);
} //while

// fermeture base
odbc_close($connexion);
} //constructeur

```

```

// -----
function calculer($personne){
    // $personne["marié"] : oui, non
    // $personne["enfants"] : nombre d'enfants
    // $personne["salaire"] : salaire annuel

    // l'objet est-il dans un état correct ?
    if (! is_array($this->erreurs) || count($this->erreurs)!=0) return -1;

    // vérification de l'appel
    if (! isset($personne[marié]) || ! isset($personne[enfants]) || ! isset($personne[salaire])){
        // erreur
        $this->erreurs[]="Appel incorrect";
        // fin return
        -1;
    } //if

    // les paramètres sont-ils corrects ?
    $personne[marié]=strtolower($personne[marié]);
    if($personne[marié]!="oui" && $personne[marié]!="non"){
        // erreur
        $this->erreurs[]="Statut marital [$personne[marié]] incorrect";
    } //if
    if(! preg_match("/^%s %d{1,3}%s $/", $personne[enfants])){
        // erreur
        $this->erreurs[]="Nombre d'enfants [$personne[enfants]] incorrect";
    } //if
    if(! preg_match("/^%s %d+%s $/", $personne[salaire])){
        // erreur
        $this->erreurs[]="salaire [$personne[salaire]] incorrect";
    } //if
    // des erreurs ?
    if(count($this->erreurs)!=0) return -1;
}

```

```

// nombre de parts
if($personne[marié]==oui) $nbParts=$personne[enfants]/2+2; else
    $nbParts=$personne[enfants]/2+1;
// une 1/2 part de plus si au moins 3 enfants
if($personne[enfants]>=3) $nbParts+=0.5;
// revenu imposable
$revenuImposable=0.72 $personne[salaire];
// quotient familial
$quotient=$revenuImposable/$nbParts;
// est mis à la fin du tableau limites pour arrêter la boucle qui suit
$this->limites[$this->nbLimites]=$quotient;
// calcul de l'impôt
$i=0;
while($quotient>$this->limites[$i]) $i++;
// du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
// ne peut déborder du tableau $limites
// maintenant on peut calculer l'impôt
return floor($revenuImposable $this->coeffR[$i]-$nbParts $this->coeffN[$i]);
} //calculImpots
} //classe impôts
?>

```

Un programme de test pourrait être le suivant :

```

<?php
// bibilothèques
include "ImpotsDSN.php";

// création d'un objet impots
$conf=array(dsn=>"mysql-dbimpots",user=>admimpots,pwd=>mdpimpots,
    table=>impots,limites=>limites,coeffR=>coeffR,coeffN=>coeffN);

$objImpots=new ImpotsDSN($conf);

// des erreurs ? if(count($objImpots->erreurs)!=0) {
// msg
echo "Les erreurs suivantes se sont produites :%n";
$erreurs=$objImpots->erreurs;
for($i=0;$i<count($erreurs);$i++) {
    echo "$erreurs[$i]%n";
} //for
// fin
exit(1);
} //if

// des essais
calculerImpots($objImpots,oui,2,200000);
calculerImpots($objImpots,non,2,200000);
calculerImpots($objImpots,oui,3,200000);
calculerImpots($objImpots,non,3,200000);
calculerImpots($objImpots,array(),array(),array());

// fin
exit(0);

// -----
function calculerImpots($objImpots,$marié,$enfants,$salaire) {
    // echo
    echo "impots($marié,$enfants,$salaire)%n";

    // calcul de l'impôt
    $personne=array(marié=>$marié,enfants=>$enfants,salaire=>$salaire);
    $montant=$objImpots->calculer($personne);
    // des erreurs ? if(count($objImpots->erreurs)!=0) {
    // msg
    echo "Les erreurs suivantes se sont produites :%n";
    $erreurs=$objImpots->erreurs; for($i=0;$i<count($erreurs);$i++) {
        echo "$erreurs[$i]%n";
    } //for
    } else echo "montant=$montant%n";
} //calculerImp

```

- le programme de test prend soin d'"inclure" le fichier contenant la classe ImpotsDSN
- puis crée un objet *objImpots* de la classe *ImpotsDSN* en passant au constructeur de l'objet les informations dont il a besoin
- une fois cet objet créé, la méthode *calculer* de celui-ci va être appelée cinq fois

Les résultats obtenus sont les suivants :

```

dos>e:\php43\php.exe test.php
impots(oui,2,200000)
montant=22504
impots(non,2,200000)
montant=33388
impots(oui,3,200000)
montant=16400
impots(non,3,200000)
montant=22504
impots(Array,Array,Array)
Les erreurs suivantes se sont produites :
Statut marital [array] incorrect
Nombre d'enfants [Array] incorrect
salaire [Array] incorrect

```

Nous utiliserons désormais la classe *ImpotsDSN* sans en redonner la définition.

Application Impôts : version 1

Nous présentons maintenant la version 1 de l'application IMPOTS. On se place dans le contexte d'une application web qui présenterait une interface HTML à un utilisateur afin d'obtenir les trois paramètres nécessaires au calcul de l'impôt :

- l'état marital (marié ou non)
- le nombre d'enfants
- le salaire annuel

L'affichage du formulaire est réalisé par la page PHP suivante :

```

<?php //formulaire des impôts ?>
<html>
<head>
<title>impots</title>

<script language="JavaScript" type="text/javascript">
function effacer(){
// raz du formulaire
with(document.frmImpots){
optMarie[0].checked=false;
optMarie[1].checked=true;
txtEnfants.value="";
txtSalaire.value="";
txtImpots.value="";
}
}
//effacer
</script>
</head>
<body background="/poly/impots/images/standard.jpg">
<center>
Calcul d'impôts
<hr>
<form name="frmImpots" method="POST">
<table>
<tr>
<td>Etes-vous marié(e)</td>

```

```

        <td>
            <input type="radio" name="optMarie" value="oui" <?php echo $requête->chkoui ?>>oui
            <input type="radio" name="optMarie" value="non" <?php echo $requête->chknon ?>>non
        </td>
    </tr>
    <tr>
        <td>Nombre d'enfants</td>
        <td><input type="text" size="5" name="txtEnfants" value="<?php echo $requête->enfants ?
    >"></td>
    </tr>
    <tr>
        <td>Salaire annuel</td>
        <td><input type="text" size="10" name="txtSalaire" value="<?php echo $requête->salaire ?
    >"></td>
    </tr>
    <tr>
        <td><font color="green">Impôt</font></td>
        <td><input type="text" size="10" name="txtImpots" value="<?php echo $requête->impots ?>"
        readonly></td>
    </tr>
    <tr></tr>
    <tr>
        <td><input type="submit" value="Calculer"></td>
        <td><input type="button" value="Effacer" onclick="effacer()"></td>
    </tr>
</table>
</form>
</center>
<?php
    // y-a-t-il des erreurs ?
    if(count($requête->erreurs)!=0){
        // affichage des erreurs
        echo "<hr>\n<font color='red'>\n";
        echo "Les erreurs suivantes se sont produites<br>";
        echo "<ul>";
        for($i=0;$i<count($requête->erreurs);$i++){
            echo "<li>". $requête->erreurs[$i]. "</li>\n";
        }
        echo "</ul>\n</font>\n";
    }
    }
}
</body>
</html>

```

La page PHP se contente d'afficher des informations qui lui sont passées par le programme principal de l'application dans la variable **\$requête** qui est un objet avec les champs suivants :

chkoui, chknon	attributs des boutons radio oui, non - ont pour valeurs possibles "checked" ou "" afin d'allumer ou non le bouton radio correspondant
enfants	le nombre d'enfants du contribuable
salaire	son salaire annuel
impots	le montant de l'impôt à payer
erreurs	une liste éventuelle d'erreurs - peut être vide.

La page envoyée au client contient un script javascript contenant une fonction *effacer* associée au bouton "Effacer" dont le rôle est de remettre le formulaire dans son état initial : bouton non coché, champs de saisie vides. On notera que ce résultat ne pouvait pas être obtenu avec un bouton HTML de type "reset". En effet, lorsque ce type de bouton est utilisé, le navigateur remet le formulaire dans l'état où il l'a reçu. Or dans notre application, le navigateur reçoit des formulaires qui peuvent être non vides.

L'application qui traite le formulaire précédent est la suivante :

```

<?php
// traite le formulaire des impôts

// bibilothèques
include "ImpotsDSN.php";

// configuration de l'application
ini_set("register_globals","off");
ini_set("display_errors","off");
$formulaireImpots="impots_form.php";
$erreursImpots="impots_erreurs.php";
$dbdImpots=array(dsn=>"mysql-dbimpots",user=>admimpots,pwd=>mdpimpots,

```

```
table=>impots,limites=>limites,coeffR=>coeffR,coeffN=>coeffN);
```

```
// on récupère les paramètres
$requete->marié=$_POST["optMarie"];
$requete->enfants=$_POST["txtEnfants"];
$requete->salaire=$_POST["txtSalaire"];
```

```
// a-t-on tous les paramètres ?
if(! isset($requete->marié) || ! isset($requete->enfants) || ! isset($requete->salaire)){
// préparation formulaire vide
$requete->chkoui="";
$requete->chknon="checked";
$requete->enfants="";
$requete->salaire="";
$requete->impots="";
$requete->erreurs=array();
// affichage formulaire
include $formulaireImpots;
// fin
exit(0);
} //if
```

```
// vérification paramètres
$requete=vérifier($requete);

// des erreurs ?
if(count($requete->erreurs)!=0){
// affichage formulaire
include "$formulaireImpots";
// fin
exit(0);
} //if
```

```
// calcul de la réponse
$requete=calculerImpots($bdImpots,$requete);
```

```
// des erreurs ?
if(count($requete->erreurs)!=0){
// affichage formulaire erreurs
include "$erreursImpots";
// fin
exit(0);
} //if
```

```
// affichage formulaire
include "$formulaireImpots";
```

```
// fin
exit(0);
```

```
// -----
```

```
function vérifier($requete){
// vérifie la validité des paramètres de la requête

// au départ pas d'erreurs
$requete->erreurs=array();

// statut valide ?
$requete->marié=strtolower($requete->marié);
if($requete->marié!="oui" && $requete->marié!="non"){
// une erreur
$requete->erreurs[]="Statut marital [$requete->marié] incorrect";
}

// nbre d'enfants valide ?
if(! preg_match("/^\s*\d+\s*$/",$requete->enfants)){
// une erreur
$requete->erreurs[]="Nombre d'enfants [$requete->enfants] incorrect";
}

// salaire valide ?
if(! preg_match("/^\s*\d+\s*$/",$requete->salaire)){
// une erreur
$requete->erreurs[]="Salaire [$requete->salaire] incorrect";
}

// état des boutons radio
```

```

    if($requête->marié=="oui") {
        $requête->chkoui="checked";
        $requête->chknon="";
    }else{
        $requête->chknon="checked";
        $requête->chkoui="";
    }

    // on rend la requête
    return $requête;
} //vérifier

// -----

function calculerImpots($bdImpots,$requête) {
    // calcule le montant de l'impôt

    // $bdImpots : dictionnaire contenant les informations nécessaires pour lire la source de données
    ODBC
    // $requête : la requête contenant les informations sur le contribuable

    // on construit un objet ImpotsDSN
    $objImpots=new ImpotsDSN($bdImpots);

    // des erreurs ?
    if(count($objImpots->erreurs)!=0) {
        // on met les erreurs dans la requête
        $requête->erreurs=$objImpots->erreurs;
        // fini
        return $requête;
    } //if

    // calcul de l'impôt
    $personne=array(marié=>"$requête->marié",enfants=>"$requête->enfants",salaire=>"$requête->salaire");
    $requête->impots=$objImpots->calculer($personne);

    // on rend le résultat
    return $requête;
} //calculerImpots

```

Commentaires :

- tout d'abord, la classe *ImpotsDSN* est incluse. C'est elle qui est à la base du calcul d'impôts et qui va nous cacher l'accès à la base de données
- un certain nombre d'initialisations sont faites visant à faciliter la maintenance de l'application. Si des paramètres changent, on changera leurs valeurs dans cette section. En général, ces valeurs de configuration sont plutôt stockées dans un fichier ou une base de données.
- on récupère les trois paramètres du formulaire correspondant aux champs HTML **optMarie**, **txtEnfants**, **txtSalaire**.
- ces paramètres peuvent être totalement ou partiellement absents. Ce sera notamment le cas lors de la demande initiale du formulaire. Dans ce cas-là, on se contente d'envoyer un formulaire vide.
- ensuite la validité des trois paramètres récupérés est vérifiée. S'ils s'avèrent incorrects, le formulaire est renvoyé tel qu'il a été saisi mais avec de plus une liste d'erreurs.
- une fois vérifiée la validité des trois paramètres, on peut faire le calcul de l'impôt. Celui-ci est calculé par la fonction *calculerImpots*. Cette fonction construit un objet de type **ImpotsDSN** et utilise la méthode *calculer* de cet objet.
- La construction de l'objet *ImpotsDSN* peut échouer si la base de données est indisponible. Dans ce cas, l'application affiche une page d'erreur spécifique indiquant les erreurs qui se sont produites.
- Si tout s'est bien passé, le formulaire est renvoyé tel qu'il a été saisi mais avec de plus le montant de l'impôt à payer.

La page d'erreurs est la suivante :

```

<?php // page d'erreur ?>
<html>
<head>
<title>Application impôts indisponible</title>
</head>
<body background="/poly/impots/images/standard.jpg">
<h3>Calcul d'impôts</h3>
<hr>
Application indisponible. Recommencez ultérieurement.<br><br>
<font color="red">
    Les erreurs suivantes se sont produites<br>
    <ul>
    <?php
    // affichage des erreurs

```

```

    for ($i=0;$i<count($requête->erreurs);$i++) { echo
        "<i>". $requête->erreurs[$i]. "</i>%n";
    }
    ?>
</ul>
</font>
</body>
</html>

```

Voici quelques exemples d'erreurs. Tout d'abord on fournit un mot de passe incorrect à la base :



Calcul d'impôts

Application indisponible. Recommencez ultérieurement.

Les erreurs suivantes se sont produites

- ♦ Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)

On fournit des données incorrectes dans le formulaire :

Calcul d'impôts

Etes-vous marié(e) ☐ oui ☒ non

Nombre d'enfants

Salaire annuel

Impôt

Les erreurs suivantes se sont produites

- ♦ Nombre d'enfants [xx] incorrect
- ♦ Salaire [qq] incorrect

Enfin, on fournit des valeurs correctes :

http://localhost/poly/impots/2/impots.php

Calcul d'impôts

Etes-vous marié(e) ☒ oui ☐ non

Nombre d'enfants

Salaire annuel

Impôt

Application Impôts : version 2

Dans l'exemple précédent, la validité des paramètres *txtEnfants*, *txtSalaire* du formulaire est vérifiée par le serveur. On se propose ici de la vérifier par un script Javascript inclus dans la page du formulaire. C'est alors le navigateur qui fait la vérification des paramètres. Le serveur n'est alors sollicité que si ceux-ci sont valides. On économise ainsi de la "bande passante". La page *impots-form.php* d'affichage devient la suivante :

```
<?php //formulaire des impôts ?>
<html>
<head>
<title>impots</title>
<script language="JavaScript" type="text/javascript"> function
effacer () {
.....
} //effacer

//-----
function calculer() {
// vérification des paramètres avant de les envoyer au serveur
with(document.frmImpots) {
//nbre d'enfants
champs=/^\s*(\d+)\s*$/ .exec(txtEnfants.value);
if(champs==null) {
// le modèle n'est pas vérifié
alert("Le nombre d'enfants n'a pas été donné ou est incorrect");
nbEnfants.focus();
return;
} //if
//salaire
champs=/^\s*(\d+)\s*$/ .exec(txtSalaire.value);
if(champs==null) {
// le modèle n'est pas vérifié
alert("Le salaire n'a pas été donné ou est incorrect");
salaire.focus();
return;
} //if
// c'est bon - on envoie le formulaire au serveur
submit();
} //with
} //calculer
</script>
</head>
<body background="/poly/impots/images/standard.jpg">
.....
<td><input type="button" value="Calculer" onclick="calculer()"></td>
<td><input type="button" value="Effacer" onclick="effacer()"></td>
</body>
</html>
```


On notera les changements suivants :

- le bouton *Calculer* n'est plus de type *submit* mais de type *button* associé à une fonction appelée *calculer*. C'est celle-ci qui fera l'analyse des champs *txtEnfants* et *txtSalaire*. Si elles sont correctes, les valeurs du formulaire seront envoyées au serveur (submit) sinon un message d'erreur sera affiché.

Voici un exemple d'affichage en cas d'erreur :

Calcul d'impôts

Etes-vous marié(e) ☐ oui ☒ non

Nombre d'enfants

Salaire annuel

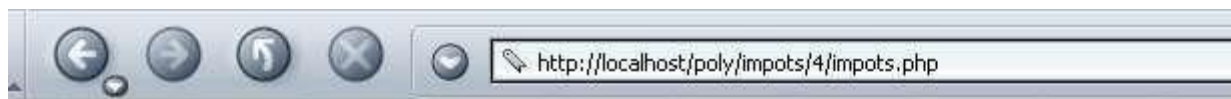
Impôt

Alerte

Le nombre d'enfants n'a pas été donné ou est incorrect

Application Impôts : version 3

Nous modifions légèrement l'application pour y introduire la notion de session. Nous considérons maintenant que l'application est une application de simulation de calcul d'impôts. Un utilisateur peut alors simuler différentes "configurations" de contribuable et voir quelle serait pour chacune d'elles l'impôt à payer. La page web ci-dessous donne un exemple de ce qui pourrait être obtenu :



Calcul d'impôts

Etes-vous marié(e) ☒ oui ☐ non

Nombre d'enfants

Salaire annuel

Impôt

Calculer

Effacer

Résultats des simulations

Marié	Enfants	Salaire annuel (F)	Impôts à payer (F)
oui	2	200000	22504
non	2	200000	33388
non	3	200000	22504
oui	3	200000	16400

Le programme d'affichage de la page ci-dessus est le suivant :

```
<?php //formulaire des impôts ?>
<html>
  <head>
    <title>impots</title>
    <script language="JavaScript" type="text/javascript">
      function effacer() {
        ...
      } //effacer
      //-----
      function calculer() {
        ...
      } //calculer
    </script>
  </head>
  <body background="/poly/impots/images/standard.jpg">
    <center>
      Calcul d'impôts
      <hr>
      <form name="frmImpots" method="POST">
        <table>
          <tr>
            <td>Etes-vous marié(e)</td>
            <td>
              <input type="radio" name="optMarie" value="oui" <?php echo $requête[chkoui] ?>>oui
              <input type="radio" name="optMarie" value="non" <?php echo $requête[chknon] ?>>non
            </td>
          </tr>
          <tr>
            <td>Nombre d'enfants</td>
            <td><input type="text" size="5" name="txtEnfants" value="<?php echo $requête[enfants] ?
            >"></td>
          </tr>
        </table>
      </form>
    </center>
  </body>
</html>
```

```

<td>Salaire annuel</td>
<td><input type="text" size="10" name="txtSalaire" value="<?php echo $requête[salaire] ?
"></td>
</tr>
<tr>
<td><font color="green">Impôt</font></td>
<td><input type="text" size="10" name="txtImpots" value="<?php echo $requête[impots] ?>"
readonly></td>
</tr>
<tr>
<td><input type="button" value="Calculer" onclick="calculer()"></td>
<td><input type="button" value="Effacer" onclick="effacer()"></td>
</tr>
</table>
</form>
</center>
<?php
// y-a-t-il des erreurs ?
if(count($requête[erreurs])!=0){
// affichage des erreurs
echo "<hr>\n<font color='red'>\n";
echo "Les erreurs suivantes se sont produites<br>";
echo "<ul>";
for($i=0;$i<count($requête[erreurs]);$i++){
echo "<li>". $requête[erreurs][$i]. "</li>\n";
}
echo "</ul>\n</font>\n";
}
}
// y-a-t-il des simulations ?
else if(count($requête[simulations])!=0){
// affichage des simulations
echo "<hr>\n<h2>Résultats des simulations</h2>\n";
echo "<table border='1'>\n";
echo "<tr><td>Marié</td><td>Enfants</td><td>Salaire annuel (F)</td><td>Impôts à payer (F)
</td></tr>\n";
for($i=0;$i<count($requête[simulations]);$i++){
echo "<tr>".
"<td>". $requête[simulations][$i][0]. "</td>".
"<td>". $requête[simulations][$i][1]. "</td>".
"<td>". $requête[simulations][$i][2]. "</td>".
"<td>". $requête[simulations][$i][3]. "</td>".
"</tr>\n";
}
}
echo "</table>\n";
}
}
?>
</body>
</html>

```

Le programme d'affichage présente des différences mineures avec sa version précédente :

- le dictionnaire reçoit un dictionnaire *\$requête* au lieu d'un objet *\$requête*. On écrit donc *\$requête[enfants]* et non *\$requête->enfants*.
- ce dictionnaire a un champ *simulations* qui est un tableau a deux dimensions. *\$requête[simulations][i]* est la simulation n° i. Celle-ci est elle-même un tableau de quatre chaînes de caractères : statut marital, nombre d'enfants, salaire annuel, impôt à payer.

Le programme principal a lui évolué plus profondément :

```

<?php
// traite le formulaire des impôts

// bibilothèques
include "ImpotsDSN.php";

// démarrage session
session_start();

// configuration de l'application
ini_set("register_globals","off");
ini_set("display_errors","off");
$formulaireImpots="impots_form.php";
$erreursImpots="impots_erreurs.php";
$dbImpots=array(dsn=>"mysql-dbimpots", user=>admimpots, pwd=>mdpimpots,
table=>impots, limites=>limites, coeffR=>coeffR, coeffN=>coeffN);

```

```

// on récupère les paramètres de la session
$session=$_SESSION["session"];
// session valide ?
if(! isset($session) || ! isset($session[objImpots]) || ! isset($session[simulations])){
// on démarre une nouvelle session
$session=array(objImpots=>new ImpotsDSN($bdImpots),simulations=>array());
// des erreurs ? if(count($session[objImpots]-
>erreurs)!=0){
    $requête=array(erreurs=>$session[objImpots]->erreurs);
    // affichage page d'erreurs
    include $erreursImpots;
    // fin
    $session=array();
    terminerSession($session);
} //if
} //if

// on récupère les paramètres de l'échange en cours
$requête[marié]=$_POST["optMarie"];
$requête[enfants]=$_POST["txtEnfants"];
$requête[salaire]=$_POST["txtSalaire"];

// a-t-on tous les paramètres ?
if(! isset($requête[marié]) || ! isset($requête[enfants]) || ! isset($requête[salaire])){
// affichage formulaire vide
    $requête=array(chkoui=>"" ,chknon=>"checked",enfants=>"" ,salaire=>"" ,impots=>"" , erreurs=>array(),simulations=>array());
    include $formulaireImpots;
    // fin terminerSession($session);
} //if

// vérification paramètres
$requête=vérifier($requête);

// des erreurs ?
if(count($requête[erreurs])!=0){
// affichage formulaire include
    "$formulaireImpots";
// fin
    terminerSession($session);
} //if

// calcul de l'impôt à payer
$requête[impots]=$session[objImpots]->calculer(array(marié=>$requête[marié],
enfants=>$requête[enfants], salaire=>$requête[salaire]));

// une simulation de plus
$session[simulations][]=array($requête[marié], $requête[enfants], $requête[salaire], $requête[impots]);
$requête[simulations]=$session[simulations];

// affichage formulaire include
"$formulaireImpots";

// fin
terminerSession($session);

// -----
function vérifier($requête){
// vérifie la validité des paramètres de la requête

// au départ pas d'erreurs
$requête[erreurs]=array();

// statut valide ?
$requête[marié]=strtolower($requête[marié]); if($requête[marié]!="oui"
&& $requête[marié]!="non"){
// une erreur
    $requête[erreurs][]="Statut marital [$requête[marié]] incorrect";
}

// nbre d'enfants valide ?
if(! preg_match("/^\s \d+\s $/", $requête[enfants])){
// une erreur
    $requête[erreurs][]="Nombre d'enfants [$requête[enfants]] incorrect";
}

// salaire valide ?
if(! preg_match("/^\s \d+\s $/", $requête[salaire])){
// une erreur
    $requête[erreurs][]="Salaire [$requête[salaire]] incorrect";
}

// état des boutons radio
if($requête[marié]=="oui"){

```

```

    $requete[chkoui]="checked";
    $requete[chknon]="";
} else {
    $requete[chknon]="checked";
    $requete[chkoui]="";
}

// on rend la requête
return $requete;
} //vérifier

// -----
function terminerSession($session) {
    // on mémorise la session
    $_SESSION[session]=$session;
    // fin script
    exit(0);
} //terminerSession

```

- tout d'abord, cette application gère une session. On lance donc une session en début de script :

```
session_start();
```

- la session mémorise une seule variable : le dictionnaire **\$session**. Ce dictionnaire a deux champs :
 - **objImpots** : un objet *ImpotsDSN*. On mémorise cet objet dans la session du client afin d'éviter des accès successifs inutiles à la base de données ODBC.
 - **simulations** : le tableau des simulations pour se rappeler d'un échange à l'autre les simulations des échanges précédents.
- une fois la session démarrée, on récupère la variable *\$session*. Si celle-ci n'existe pas encore, c'est que la session démarre. On crée alors un objet *ImpotsDSN* et un tableau de simulations vide. Si besoin est, des erreurs sont signalées.

```

// on récupère les paramètres de la session
$session=$_SESSION["session"];
// session valide ?
if(! isset($session) || ! isset($session[objImpots]) || ! isset($session[simulations])) {
    // on démarre une nouvelle session
    $session=array(objImpots=>new ImpotsDSN($bdImpots), simulations=>array());
    // des erreurs ? if(count($session[objImpots]-
    >erreurs)!=0) {
        $requete=array(erreurs=>$session[objImpots]->erreurs);
        // affichage page d'erreurs
        include $erreursImpots;
        // fin
        $session=array();
        terminerSession($session);
    } //if
} //if

```

- une fois la session correctement initialisée, les paramètres de l'échange sont récupérés. S'il n'y a pas tous les paramètres attendus, un formulaire vide est envoyé.

```

// on récupère les paramètres de l'échange en cours
$requete[marié]=$_POST["optMarie"];
$requete[enfants]=$_POST["txtEnfants"];
$requete[salaire]=$_POST["txtSalaire"];

// a-t-on tous les paramètres ?
if(! isset($requete[marié]) || ! isset($requete[enfants]) || ! isset($requete[salaire])) {
    // affichage formulaire vide
    $requete=array(chkoui=>"", chknon=>"checked", enfants=>"", salaire=>"", impots=>"",
    erreurs=>array(), simulations=>array());
    include $formulaireImpots;
    // fin terminerSession($session);
} //if

```

- si on a bien tous les paramètres, ils sont vérifiés. S'il y a des erreurs, elles sont signalées :

```

// vérification paramètres
$requete=vérifier($requete);

// des erreurs ? if(count($requete[erreurs])!=0) {
    // affichage formulaire include
    "$formulaireImpots";
    // fin
    terminerSession($session);
}

```

```
}//if
```

- si les paramètres sont corrects, l'impôt est calculé :

```
// calcul de l'impôt à payer
$requete[impots]=$session[objImpots]->calculer(array(marié=>$requete[marié],
enfants=>$requete[enfants], salaire=>$requete[salaire]));
```

- la simulation en cours est ajoutée au tableau des simulations et celui-ci envoyé au client avec le formulaire :

```
// une simulation de plus
$session[simulations][]=array($requete[marié], $requete[enfants], $requete[salaire], $requete[impots]);
$requete[simulations]=$session[simulations];

// affichage formulaire include
"$formulaireImpots";

// fin
terminerSession($session);
```

- dans tous les cas, le script se termine en enregistrant la variable **\$session** dans la session. Ceci est fait dans la procédure *terminerSession*.

```
function terminerSession($session) {
    // on mémorise la session
    $_SESSION[session]=$session;
    // fin script
    exit(0);
} //terminerSession
```

Terminons en présentant le programme d'affichage des erreurs d'accès à la base de données (impots-erreurs.php) :

```
<?php // page d'erreur ?>
<html>
<head>
<title>Application impôts indisponible</title>
</head>
<body background="/poly/impots/images/standard.jpg">
<h3>Calcul d'impôts</h3>
<hr>
Application indisponible. Recommencez ultérieurement.<br><br>
<font color="red">
    Les erreurs suivantes se sont produites<br>
    <ul>
    <?php
        // affichage des erreurs
        for ($i=0;$i<count($requete[erreurs]);$i++){ echo
            "<li>". $requete[erreurs][$i]. "</li>";
        }
    }
    </ul>
</font>
</body>
</html>
```

Ici également, l'objet *\$requete* a été remplacé par un dictionnaire *\$requete*.

Application Impôts : version 4

Nous allons maintenant créer une application autonome qui sera un client web de l'application web *impots* précédente. L'application sera une application console lancée à partir d'une fenêtre DOS :

```
dos>e:\php43\php.exe cltImpots.php
Syntaxe cltImpots.php urlImpots marié enfants salaire [jeton]
```

Les paramètres du client web **cltImpots** sont les suivants :

```
// syntaxe $0 urlImpots marié enfants salaire jeton
// client d'un service d'impôts fonctionnant sur urlImpots
// envoie à ce service les trois informations : marié, enfants, salaire
// éventuellement avec le jeton de session si celui-ci a été passé
// affiche le tableau des simulations de la session
```

Voici quelques exemples d'utilisation. Tout d'abord un premier exemple sans jeton de session.

```
dos>e:\php43\php.exe cltImpots.php http://localhost/poly/impots/6/impots.php oui 2 200000

Jeton de session=[a6297317667bc981c462120987b8dd18]

Simulations :
[Marié,Enfants,Salaire annuel (F),Impôts à payer (F)]
[oui,2,200000,22504]
```

Nous avons bien récupéré le montant de l'impôt à payer (22504 f). Nous avons également récupéré le jeton de session. Nous pouvons maintenant l'utiliser pour une seconde interrogation :

```
dos>e:\php43\php.exe cltImpots.php http://localhost/poly/impots/6/impots.php oui 3 200000
a6297317667bc981c462120987b8dd18

Jeton de session=[a6297317667bc981c462120987b8dd18]

Simulations :
[Marié,Enfants,Salaire annuel (F),Impôts à payer (F)]
[oui,2,200000,22504]
[oui,3,200000,16400]
```

Nous obtenons bien le tableau des simulations envoyées par le serveur web. Le jeton récupéré reste bien sûr le même. Si nous interrogeons le service web alors que la base de données n'a pas été lancée, nous obtenons le résultat suivant :

```
dos>e:\php43\php.exe cltImpots2.php http://localhost/poly/impots/6/impots.php oui 3 200000

Jeton de session=[8369014d5053212bc42f64bbdfb152ee]

Les erreurs suivantes se sont produites :
Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)
```

Lorsqu'on écrit un client web programmé, il est nécessaire de savoir exactement ce qu'envoie le serveur en réponse aux différentes demandes possibles d'un client. Le serveur envoie un ensemble de lignes HTML comprenant des informations utiles et d'autres qui ne sont là que pour la mise en page HTML. Les expressions régulières de PHP peuvent nous aider à trouver les informations utiles dans le flot des lignes envoyées par le serveur. Pour cela, nous avons besoin de connaître le format exact des différentes réponses du serveur. Pour cela, nous pouvons interroger le service web avec un navigateur et regarder le code source qui a été envoyé. On notera que cette méthode ne permet pas de connaître les entêtes HTTP de la réponse du serveur web. Il est parfois utile de connaître ceux-ci. On peut alors utiliser l'un des deux clients web génériques vus dans le chapitre précédent.

Si nous répétons les exemples précédents, voici le code HTML reçu par le navigateur pour le tableau des simulations :

```
<h2>Résultats des simulations</h2>
<table border="1">
<tr><td>Marié</td><td>Enfants</td><td>Salaire annuel (F)</td><td>Impôts à payer (F)</td></tr>
<tr><td>oui</td><td>2</td><td>200000</td><td>22504</td></tr>
<tr><td>oui</td><td>3</td><td>200000</td><td>16400</td></tr>
</table>
```

C'est une table HTML, la seule dans le document envoyé. Ainsi l'expression régulière `"<tr>\s*<td>(.*?)</td>\s*<td>(.*?)</td>\s*<td>(.*?)</td>\s*<td>(.*?)</td>\s*</tr>/"` doit-elle permettre de retrouver dans le document les différentes simulations reçues. Lorsque la base de données est indisponible, on reçoit le document suivant :

```
Application indisponible. Recommencez ultérieurement.<br><br>
<font color="red">
  Les erreurs suivantes se sont produites<br>
  <ul>
    <li>Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)</li>
  </ul>
</font>
```

Pour récupérer l'erreur, le client web devra chercher dans la réponse du serveur web les lignes correspondant au modèle : `"(.*?)/"`.

Nous avons maintenant les lignes directrices de ce qu'il faut faire lorsque l'utilisateur demande le calcul de l'impôt à partir du client console précédent :

- vérifier que tous les paramètres sont valides et éventuellement signaler les erreurs.

- se connecter à l'URL donnée comme premier paramètre. Pour cela on suivra le modèle du client web générique déjà présenté et étudié
- dans le flux de la réponse du serveur, utiliser des expressions régulières pour soit :
 - trouver les messages d'erreurs
 - trouver les résultats des simulations

Le code du client *cltImpots.php* est le suivant :

```
<?php
// syntaxe $0 urlImpots marié enfants salaire jeton
// client d'un service d'impôts fonctionnant sur urlImpots
// envoie à ce service les trois informations : marié, enfants, salaire
// éventuellement avec le jeton de session si celui-ci a été passé
// affiche le tableau des simulations de la session

// vérification nbre d'arguments if(count($argv)!=5 &&
count($argv)!=6) {
// erreur
fwrite(STDERR,"Syntaxe $argv[0] urlImpots marié enfants salaire [jeton]¥n");
// fin
exit(1);
} //if

// on note l'URL
$urlImpots=analyseURL($argv[1]);
if(isset($urlImpots[erreur])) {
// erreur
fwrite(STDERR,"$urlImpots[erreur]¥n");
// fin
exit(1);
} //if

// analyse de la demande de l'utilisateur
$demande=analyseDemande("$argv[2], $argv[3], $argv[4]");
// erreurs ?
if($demande[erreur]) {
// msg
echo "$demande[erreur]¥n";
// c'est fini
exit(1);
} //if

// on peut faire la demande - on utilise le jeton de session
$impots=getImpots($urlImpots,$demande,$argv[5]);

// on affiche le jeton de session
echo "Jeton de session=[$impots[jeton]]¥n";

// des erreurs ?
if(count($impots[erreurs])!=0) {
//msg d'erreurs
echo "Les erreurs suivantes se sont produites :¥n";
for($i=0;$i<count($impots[erreurs]);$i++) {
echo $impots[erreurs][$i]. "¥n";
} //for
} else {
// pas d'erreurs - affichage des simulations
echo "Simulations :¥n";
for($i=0;$i<count($impots[simulations]);$i++) {
echo "[". implode(" ", $impots[simulations][$i]). " ]¥n";
} //for
} //if

// fin du programme
exit(0);

// -----
function analyseURL($URL) {
// analyse la validité de l'URL $URL
$url=parse_url($URL);
// le protocole
if(strtolower($url[scheme])!="http") {
$url[erreur]="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $url;
} //if
// la machine
if(!isset($url[host])) {
$url[erreur]="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $url;
} //if
// le port
if(!isset($url[port])) $url[port]=80;
// la requête
```



```

if(isset($url["query"])) {
    $url[erreur]="l'URL [$URL] n'est pas au formathttp://machine[:port]/chemin"; return $url;
} //if
// retour
return $url;
} //analyseURL

// -----
function analyseDemande($demande) {
    // $demande : chaîne à analyser
    // doit être de la forme marié, enfants, salaire
    // format valide
    if(! preg_match("/^%s (oui|non)%s , %s (%d{1,3})%s , %s (%d+)%s $/i", $demande, $champs)) {
        // format invalide
        return(array(erreur=>"Format (marié, enfants, salaire) invalide."));
    }
    // c'est bon
    $marié=strtolower($champs[1]);
    return array(marié=>$marié, enfants=>$champs[2], salaire=>$champs[3]);
} //analyseDemande

// -----
function getImpots($urlImpots, $demande, $jeton) {
    // $urlImpots : URL à interroger
    // $demande : dictionnaire contenant les champs marie, enfants, salaire
    // $jeton : un éventuel jeton de session

    // ouverture d'une connexion sur le port $urlImpots[port] de $urlImpots[host]
    $connexion=fsockopen($urlImpots[host], $urlImpots[port], &$errno, &$erreur);
    // retour si erreur
    if(! $connexion) {
        return array(erreurs=>array("Echec de la connexion au site ($urlImpots[host], $urlImpots[port]): $erreur"));
    } //if

    // on envoie les entetes HTTP au serveur
    POST($connexion, $urlImpots, $jeton,
        array(optMarie=>$demande[marié], txtEnfants=>$demande[enfants], txtSalaire=>$demande[salaire]));

    // on lit la réponse du serveur - d'abord les entêtes HTTP
    // la première ligne
    $ligne=fgets($connexion, 10000);

    // URL trouvée ?
    if(! preg_match("/^(.+?) 200 OK%s $/", $ligne)) {
        // URL inaccessible - retour avec erreur
        return array(erreurs=>array("L'URL $urlImpots[path] n'a pu être trouvée"));
    } //if

    // on lit les autres entêtes HTTP
    while(($ligne=fgets($connexion, 10000)) && (($ligne=rtrim($ligne))!="")) {
        // recherche du jeton si on ne l'a pas encore trouvé if(! $jeton) {
        // recherche de la ligne set-cookie
        if(preg_match("/^Set-Cookie: PHPSESSID=(. ?):/i", $ligne, $champs)) {
            // on a trouvé le jeton - on le mémorise
            $jeton=$champs[1];
        } //if
    } //if
    // ligne suivante

    // on lit le document qui suit
    $document="";
    while($ligne=fread($connexion, 10000)) {
        $document.=$ligne;
    } //while

    // on ferme la connexion
    fclose($connexion);

    // on analyse le document
    $impots=getInfos($document);

    // on rend le résultat
    return array(jeton=>$jeton, erreurs=>$impots[erreurs], simulations=>$impots[simulations]);
} //getImpots

// -----
function POST($connexion, $url, $jeton, $paramètres) {
    // $connexion : la connexion a serveur web
    // $url : l'URL à interroger
    // $paramètres : dictionnaire des paramètres à poster

    // on prépare le POST
    $post="";

```

```

while(list($paramètre,$valeur)=each($paramètres)){
    $post.=$paramètre."&".urlencode($valeur)."&";
}
// on enlève le dernier caractère
$post=substr($post,0,-1);

// on prépare la requête HTTP au format HTTP/1.0
$HTTP="POST $url[path] HTTP/1.0\r\n";
$HTTP.="Content-type: application/x-www-form-urlencoded\r\n";
$HTTP.="Content-length: ".strlen($post)."\r\n";
$HTTP.="Connection: close\r\n";
if($jeton) $HTTP.="Cookie: PHPSESSID=$jeton\r\n";
$HTTP.="\r\n";
$HTTP.=$post;

// on envoie la requête HTTP
fwrite($connexion,$HTTP);
}
// POST

// -----
function getInfos($document) {
// $document : document HTML
// on cherche soit la liste d'erreurs
// soit le tableau des simulations

// préparation du résultat
$impots[erreurs]=array();
$impots[simulations]=array();

// y-a-t-il une erreur ?
// modèle de la ligne d'erreur
$modErreur="|<li>(.*?)</li>|";
// recherche dans le document
if(preg_match_all($modErreur,$document,$champs,PREG_SET_ORDER)) {
// récupération des erreurs
for($i=0;$i<count($champs);$i++){
    $impots[erreurs][$i]=$champs[$i][1];
}
// fini
return $impots;
}

// modèle d'une ligne du tableau des simulations
// <tr><td>oui</td><td>2</td><td>200000</td><td>22504</td></tr>
$modSimulation="|<tr>.*s <td>(.*?)</td>.*s <td>(.*?)</td>.*s <td>(.*?)</td>.*s <td>(.*?)</td>.*s</tr>|";
// recherche dans le document if(preg_match_all($modSimulation,$document,$champs,PREG_SET_ORDER)) {
// récupération des simulations
for($i=0;$i<count($champs);$i++){
    $impots[simulations][$i]=array($champs[$i][1],$champs[$i][2],$champs[$i][3],$champs[$i][4]);
}
// retour résultat
return $impots;
}

// pas normal d'arriver là
return $impots;
}
}
?>

```

Commentaires :

- le programme commence par vérifier la validité des paramètres qu'il a reçus. Il utilise pour ce faire deux fonctions : *analyseURL* qui vérifie la validité de l'URL et *analyseDemande* qui vérifie les autres paramètres.
- le calcul de l'impôt est fait par la fonction *getImpots* :

```
$impots=getImpots($urlImpots,$demande,$argv[5]);
```

- la fonction **getImpots** est déclarée comme suit :

```

// -----
function getImpots($urlImpots,$demande,$jeton) {
// $urlImpots : URL à interroger
// $demande : dictionnaire contenant les champs marie, enfants, salaire
// $jeton : un éventuel jeton de session

```

\$urlImpots est un dictionnaire contenant les champs :

- host** : machine où réside le service web
- port** : son port de service
- path** : le chemin de la ressource demandée

\$demande est un dictionnaire contenant les champs :

marié : oui/non : état marital
enfants : nombre d'enfants
salaire : salaire annuel

\$jeton est le jeton de session.

La fonction rend comme résultat un dictionnaire ayant les champs :

erreurs : tableau de messages d'erreurs
simulations : tableau de simulations, une simulation étant elle-même un tableau de quatre éléments (marié, enfants, salaire, impôt)
jeton : le jeton de session

- une fois le résultat de **getImpots** obtenu, le programme affiche les résultats et se termine :

```
// on affiche le jeton de session
echo "Jeton de session=[\$impots[jeton]]\n";

// des erreurs ? if(count(\$impots[erreurs])!=0) {
//msg d'erreurs
echo "Les erreurs suivantes se sont produites :\n";
for($i=0;$i<count(\$impots[erreurs]);$i++) {
    echo \$impots[erreurs][$i]."\n";
} //for
} else {
// pas d'erreurs - affichage des simulations echo
"Simulations :\n";
for($i=0;$i<count(\$impots[simulations]);$i++) {
    echo "[". implode(" ", \$impots[simulations][$i])."]\n";
} //for
} //if

// fin du programme
exit(0);
```

- analysons maintenant la fonction **getImpots(\$urlImpots,\$demande,\$jeton)** qui doit
 - créer une connexion TCP-IP sur le port *\$urlImpots[port]* de la machine *\$urlImpots[host]*
 - envoyer au serveur web les entêtes HTTP qu'il attend notamment le jeton de session s'il y en a un
 - envoyer au serveur web une requête POST avec les paramètres contenus dans le dictionnaire *\$demande*
 - analyser la réponse du serveur web pour y trouver soit une liste d'erreurs soit un tableau de simulations.
- l'envoi des entêtes HTTP se fait à l'aide d'une fonction POST :

```
// on envoie les entetes HTTP au serveur
POST($connexion, $urlImpots, $jeton,
array(optMarie=>$demande[marié], txtEnfants=>$demande[enfants], txtSalaire=>$demande[salaire]));
```

- une fois les entêtes HTTP envoyés, la fonction *getImpots* lit en totalité de la réponse du serveur et la met dans *\$document*. La réponse est lue sans être analysée sauf pour la première ligne qui nous permet de savoir si le serveur web a trouvé ou non l'URL qu'on lui a demandée. En effet si l'URL a été trouvée, le serveur web répond **HTTP/1.X 200 OK** où X dépend de la version de HTTP utilisée.

```
// URL trouvée ?
if(! preg_match("/^(.+) 200 OK\s $/", $ligne)) {
// URL inaccessible - retour avec erreur
return array(erreurs=>array("L'URL $urlImpots[path] n'a pu être trouvée"));
} //if
```

- le document est analysé à l'aide de la fonction *getInfos*:

```
// on analyse le document
$impots=getInfos($document);
```

Le résultat rendu est un dictionnaire à deux champs :

erreurs : liste des erreurs - peut être vide
simulations : liste des simulations - peut être vide

- ceci fait, la fonction *getImpots* peut rendre son résultat sous la forme d'un dictionnaire.

```
// on rend le résultat
return array(jeton=>$jeton, erreurs=>$impots[erreurs], simulations=>$impots[simulations]);
```

- analysons maintenant la fonction POST :

```
// -----
function POST($connexion, $url, $jeton, $paramètres) {
// $connexion : la connexion a serveur web
// $url : l'URL à interroger
// $paramètres : dictionnaire des paramètres à poster

// on prépare le POST
$post=""; while(list($paramètre, $valeur)=each($paramètres)) {
    $post.=$paramètre."=".urlencode($valeur)."&";
} //while
// on enlève le dernier caractère
$post=substr($post, 0, -1);

// on prépare la requête HTTP au format HTTP/1.0
$HTTP="POST $url[path] HTTP/1.0\r\n";
$HTTP.="Content-type: application/x-www-form-urlencoded\r\n";
$HTTP.="Content-length: ".strlen($post)."\r\n";
$HTTP.="Connection: close\r\n";
if($jeton) $HTTP.="Cookie: PHPSESSID=$jeton\r\n";
$HTTP.="\r\n";
$HTTP.=$post;

// on envoie la requête HTTP
fwrite($connexion, $HTTP);
} //POST
```

Si par programme, nous avons eu l'occasion de demander une ressource web par un GET, nous n'avions pas encore eu l'occasion de le faire avec un POST. Le POST diffère du GET dans la façon d'envoyer des paramètres au serveur. Il doit envoyer les entêtes HTTP suivants :

POST chemin HTTP/1.X

Content-type: application/x-www-form-urlencoded

Content-length: N

avec

- *chemin* : chemin de la ressource web demandée, ici *\$url[path]*
- *HTTP/1.X* : le protocole HTTP désiré. Ici on a choisi HTTP/1.0 pour avoir la réponse en un seul bloc. HTTP/1.1 autorise l'envoi de la réponse en plusieurs blocs (chunked).
- *N* désigne le nombre de caractères que le client s'apprête à envoyer au serveur

Les N caractères constituant les paramètres de la requête sont envoyés immédiatement derrière la ligne vide qui termine les entêtes HTTP envoyés au serveur. Ces paramètres sont sous la forme *param1=val1 ¶m2=val2 &...* où *parami* est le nom du paramètre et *vali* sa valeur. Les valeurs *vali* peuvent contenir des caractères "gênants" tels des espaces, le caractère &, le caractère =, etc.. Ces caractères doivent être remplacés par une chaîne %XX où XX est leur code hexadécimal. La fonction PHP *urlencode* fait ce travail. Le travail inverse est fait par *urldecode*.

Enfin on notera que si on a un jeton, il est envoyé avec un entête HTTP **Cookie: PHPSESSID=jeton**.

- il nous reste à étudier la fonction qui analyse la réponse du serveur :

```
// -----
function getInfos($document) {
// $document : document HTML
// on cherche soit la liste d'erreurs
// soit le tableau des simulations

// préparation du résultat
$impots[erreurs]=array();
$impots[simulations]=array();
```

- rappelons que les erreurs sont envoyées par le serveur sous la forme *message d'erreur*. L'expression régulière *"(.*?)|"* doit permettre de récupérer ces informations. Nous avons ici utilisé le signe | pour délimiter l'expression régulière plutôt que le signe / car celui-ci existe aussi dans l'expression régulière elle-même. La fonction *preg_match_all(\$modèle, \$document, \$champs, PREG_SET_ORDER)* permet de récupérer toutes les occurrences de *\$modèle* trouvées dans *\$document*. Celles-ci sont

misés dans le tableau *\$champs*. Ainsi *\$champs[i]* représente l'occurrence n° i de *\$modèle* dans *\$document*. Dans *\$champs[\$i][0]* est placée la chaîne correspondant au modèle. Si celui-ci avait des parenthèses, la chaîne correspondant à la première parenthèse est placée dans *\$champs[\$i][1]*, la seconde dans *\$champs[\$i][2]* et ainsi de suite. Le code pour récupérer les erreurs sera donc le suivant :

```
// y-a-t-il une erreur ?
// modèle de la ligne d'erreur
$modErreur="|<li>(.*?)</li>|";
// recherche dans le document
if(preg_match_all($modErreur,$document,$champs,PREG_SET_ORDER)){
    // récupération des erreurs for ($i=0;$i<count($champs);$i++){
        $impots[erreurs][]=$champs[$i][1];
    }
}
// fini
return $impots;
}
//if
```

Application Impôts : Conclusion

Nous avons montré différentes versions de notre application client-serveur de calcul d'impôts :

- **version 1** : le service est assuré par un ensemble de programmes PHP, le client est un navigateur. Il fait une seule simulation et n'a pas de mémoire des précédentes.
- **version 2** : on ajoute quelques capacités côté navigateur en mettant des scripts javascript dans le document HTML chargé par celui-ci. Il contrôle la validité des paramètres du formulaire.
- **version 3** : on permet au service de se souvenir des différentes simulations opérées par un client en gérant une session. L'interface HTML est modifiée en conséquence pour afficher celles-ci.
- **version 4** : le client est désormais une application console autonome. Cela nous permet de revenir sur l'écriture de clients web programmés.

A ce point, on peut faire quelques remarques :

- les versions 1 à 3 autorisent des navigateurs sans capacité autre que celle de pouvoir exécuter des scripts javascript. On notera qu'un utilisateur a toujours la possibilité d'inhiber l'exécution de ces derniers. L'application ne fonctionnera alors que partiellement dans sa version 1 (option Effacer ne fonctionnera pas) et pas du tout dans ses versions 2 et 3 (options Effacer et Calculer ne fonctionneront pas). Il pourrait être intéressant de prévoir une version du service n'utilisant pas de scripts javascript.

Lorsqu'on écrit un service Web, il faut se demander quels types de clients on vise. Si on veut viser le plus grand nombre de clients, on écrira une application qui n'envoie aux navigateurs que du HTML (pas de javascript ni d'applet). Si on travaille au sein d'un intranet et qu'on maîtrise la configuration des postes de celui-ci on peut alors se permettre d'être plus exigeant au niveau du client.

La version 4 est un client web qui retrouve l'information dont il a besoin au sein du flux HTML envoyé par le serveur. Très souvent on ne maîtrise pas ce flux. C'est le cas lorsqu'on a écrit un client pour un service web existant sur le réseau et géré par quelqu'un d'autre. Prenons un exemple. Supposons que notre service de simulations de calcul d'impôts ait été écrit par une société X. Actuellement le service envoie les simulations dans un tableau HTML et notre client exploite ce fait pour les récupérer. Il compare ainsi chaque ligne de la réponse du serveur à l'expression régulière :

```
// modèle d'une ligne du tableau des simulations
// <tr><td>oui</td><td>2</td><td>200000</td><td>22504</td></tr>
$modSimulation="|<tr>%s <td>(.*?)</td>%s <td>(.*?)</td>%s <td>(.*?)</td>%s <td>(.*?)</td>%s</tr>|";
```

Supposons maintenant que le concepteur de l'application change l'apparence visuelle de la réponse en mettant les simulations non pas dans un tableau mais dans une liste sous la forme :

```
Résultat des simulations
<ul>
  <li>oui, 2, 200000, 22504
  <li>non, 2, 200000, 33388
</ul>
```

Dans ce cas, notre client web devra être réécrit. C'est là, la menace permanente pesant sur les clients web d'applications qu'on ne maîtrise pas soi-même. XML peut apporter une solution à ce problème :

- au lieu de générer du HTML, le service de simulations va générer du XML. Dans notre exemple, cela pourrait être

```
<simulations>
  <entetes marie="marié" enfants="enfants" salaire="salaire" impot="impôt"/>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

- une feuille de style pourrait être associée à cette réponse indiquant aux navigateurs la forme visuelle à donner à cette réponse XML
- les clients web programmés ignoreront cette feuille de style et récupéreront l'information directement dans le flux XML de la réponse

Si le concepteur du service souhaite modifier la présentation visuelle des résultats qu'il fournit, il modifiera la feuille de style et non le XML. Grâce à la feuille de style, les navigateurs afficheront la nouvelle forme visuelle et les clients web programmés n'auront pas eux à être modifiés.

4.XML et PHP

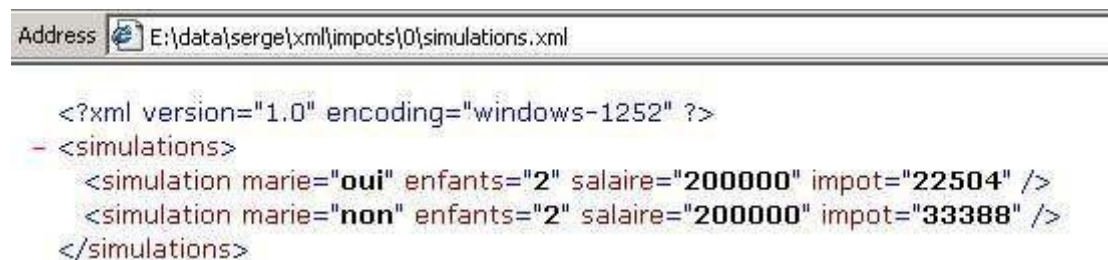
Dans ce chapitre, nous allons faire une introduction à l'utilisation de documents XML (eXtensible Markup Language) avec PHP. Nous le ferons dans le contexte de l'application **impôts** étudiée dans le chapitre précédent.

FichiersXMLetfeuillesdestyle XSL

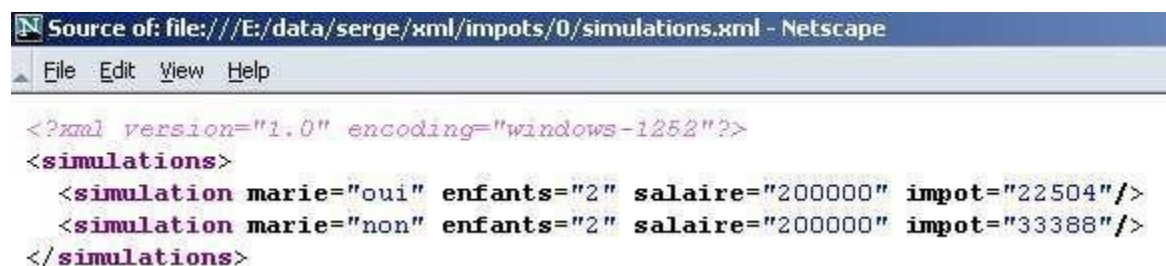
Considérons le fichier XML suivant qui pourrait représenter le résultat de simulations :

```
<?xml version="1.0" encoding="windows-1252"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

Si on le visualise avec IE 6, on obtient le résultat suivant :



IE6 reconnaît qu'il a affaire à un fichier XML (grâce au suffixe .xml du fichier) et le met en page d'une façon qui lui est propre. Avec Netscape on obtient une page vide. Cependant si on regarde le code source (View/Source) on a bien le fichier XML d'origine :



Pourquoi Netscape n'affiche-t-il rien ? Parce qu'il lui faut une feuille de style qui lui dise comment transformer le fichier XML en fichier HTML qu'il pourra alors afficher. Il se trouve que IE 6 a lui, une feuille de style par défaut lorsque le fichier XML n'en propose pas, ce qui était le cas ici.

Il existe un langage appelé **XSL** (eXtensible StyleSheet Language) permettant de décrire les transformations à effectuer pour passer un fichier XML en un fichier texte quelconque. XSL permet l'utilisation de nombreuses instructions et ressemble fort aux langages de programmation. Nous le détaillerons pas ici car il y faudrait plusieurs dizaines de pages. Nous allons simplement décrire deux exemples de feuilles de style XSL. La première est celle qui va transformer le fichier XML **simulations.xml** en code HTML. On modifie ce dernier afin qu'il désigne la feuille de style que pourront utiliser les navigateurs pour le transformer en document HTML qu'ils pourront afficher :

```
<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

La commande XML

```
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
```

désigne le fichier **simulations.xsl** comme un feuille de style (**xml-stylesheet**) de type **text/xsl** c.a.d. un fichier texte contenant du code XSL. Cette feuille de style sera utilisée par les navigateurs pour transformer le texte XML en document HTML. Voici le résultat obtenu avec Netscape 7 lorsqu'on charge le fichier XML **simulations.xml** :



Simulations de calculs d'impôts

marie	enfants	salaire	impôt
oui	2	200000	22504
non	2	200000	33388

Lorsque nous regardons le code source du document (View/Source) nous retrouvons le document XML initial et non le document HTML affiché :

```
Source of: file:///E:/data/serge/xml/impots/0/simulations.xml - Netscape
File Edit View Help

<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

Netscape a utilisé la feuille de style **simulations.xsl** pour transformer le document XML ci-dessus en document HTML affichable. Il est maintenant temps de regarder le contenu de cette feuille de style :

```
<?xml version="1.0" encoding="windows-1252"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Simulations de calculs d'impôts</title>
      </head>
      <body>
        <center>
          <h3>Simulations de calculs d'impôts</h3>
          <hr/>
          <table border="1">
            <th>marie</th><th>enfants</th><th>salaire</th><th>impôt</th>
            <xsl:apply-templates select="/simulations/simulation"/>
          </table>
        </center>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="simulation">
    <tr>
      <td><xsl:value-of select="@marie"/></td>
      <td><xsl:value-of select="@enfants"/></td>
      <td><xsl:value-of select="@salaire"/></td>
      <td><xsl:value-of select="@impot"/></td>
    </tr>
```



```
</xsl:template>
</xsl:stylesheet>
```

- une feuille de style XSL est un fichier XML et en suit donc les règles. Il doit être en autres choses "bien formé" c'est à dire que toute balise ouverte doit être fermée.
- le fichier commence par deux commandes XML qu'on pourra garder dans toute feuille de style XSL sous Windows :

```
<?xml version="1.0" encoding="windows-1252"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

L'attribut *encoding="windows-1252"* permet d'utiliser les caractères accentués dans la feuille de style.

- La balise `<xsl:output method="html" indent="yes"/>` indique à l'interpréteur XSL qu'on veut produire du HTML "indenté".
- La balise `<xsl:template match="élément">` sert à définir l'élément du document XML sur lequel vont s'appliquer les instructions que l'on va trouver entre `<xsl:template ...>` et `</xsl:template>`.

```
<xsl:template match="/">
.....
</xsl:template>
```

Dans l'exemple ci-dessus l'élément `"/"` désigne la racine du document. Cela signifie que dès que le début du document XML va être rencontré, les commandes XSL situées entre les deux balises vont être exécutées.

- Tout ce qui n'est pas balise XSL est mis tel quel dans le flux de sortie. Les balises XSL elles sont exécutées. Certaines d'entre-elles produisent un résultat dans le flux de sortie. Etudions l'exemple suivant :

```
<xsl:template match="/">
<html>
  <head>
    <title>Simulations de calculs d'impôts</title>
  </head>
  <body>
    <center>
      <h3>Simulations de calculs d'impôts</h3>
      <hr/>
      <table border="1">
        <th>marié</th><th>enfants</th><th>salaire</th><th>impôt</th>
        <xsl:apply-templates select="/simulations/simulation"/>
      </table>
    </center>
  </body>
</html>
</xsl:template>
```

Rappelons que le document XML analysé est le suivant :

```
<?xml version="1.0" encoding="windows-1252"?>
<simulations>
  <simulation marié="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marié="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

Dès le début du document XML analysé (`match="/"`), l'interpréteur XSL va produire en sortie le texte

```
<html>
  <head>
    <title>Simulations de calculs d'impôts</title>
  </head>
  <body>
    <center>
      <h3>Simulations de calculs d'impôts</h3>
      <hr>
      <table border="1">
        <th>marié</th><th>enfants</th><th>salaire</th><th>impôt</th>
```

On remarquera que dans le texte initial on avait `<hr/>` et non pas `<hr>`. Dans le texte initial on ne pouvait pas la balise `<hr>` qui, si elle est une balise HTML valide, est néanmoins une balise XML invalide. Or nous avons affaire ici à un texte XML qui doit être "bien formé", c.a.d que toute balise doit être fermée. On écrit donc `<hr/>` et parce qu'on a écrit `<xsl:output text="html ...>` l'interpréteur transformera le texte `<hr/>` en `<hr>`. Derrière ce texte, viendra ensuite le texte produit par la commande XSL :

```
<xsl:apply-templates select="/simulations/simulation"/>
```

Nous verrons ultérieurement quel est ce texte. Enfin l'interpréteur ajoutera le texte :

```

</table>
</center>
</body>
</html>

```

La commande `<xsl:apply-templates select="/simulations/simulation"/>` demande qu'on exécute le "template" (modèle) de l'élément `/simulations/simulation`. Elle sera exécutée à chaque fois que l'interpréteur XSL rencontrera dans le texte XML analysé une balise `<simulation>..</simulations>` ou `<simulation/>` à l'intérieur d'une balise `<simulations>..</simulations>`. A la rencontre d'une telle balise, l'interpréteur exécutera les instructions du modèle suivant :

```

<xsl:template match="simulation">
  <tr>
    <td><xsl:value-of select="@marie"/></td>
    <td><xsl:value-of select="@enfants"/></td>
    <td><xsl:value-of select="@salaire"/></td>
    <td><xsl:value-of select="@impot"/></td>
  </tr>
</xsl:template>

```

Prenons les lignes XML suivantes :

```

<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>

```

La ligne `<simulation ..>` correspond au modèle de l'instruction XSL `<xsl:apply-templates select="/simulations/simulation">`. L'interpréteur XSL va donc chercher à lui appliquer les instructions qui correspondent à ce modèle. Il va trouver le modèle `<xsl:template match="simulation">` et va l'exécuter. Rappelons que ce qui n'est pas une commande XSL est repris tel quel par l'interpréteur XSL et que les commandes XSL sont elles remplacées par le résultat de leur exécution. L'instruction XSL `<xsl:value-of select="@champ"/>` est ainsi remplacé par la valeur de l'attribut "champ" du noeud analysé (ici un noeud `<simulation>`). L'analyse de la ligne XML précédente va produire en sortie le résultat suivant :

XSL	sortie
<code><tr><td></code>	<code><tr><td></code>
<code><xsl:value-of select="@marie"/></code>	oui
<code></td><td></code>	<code></td><td></code>
<code><xsl:value-of select="@enfants"/></code>	2
<code></td><td></code>	<code></td><td></code>
<code><xsl:value-of select="@salaire"/></code>	200000
<code></td><td></code>	<code></td><td></code>
<code><xsl:value-of select="@impot"/></code>	22504
<code></td></tr></code>	<code></td></tr></code>

Au total, la ligne XML

```

<simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>

```

va être transformée en ligne HTML :

```

<tr><td>oui</td><td>2</td><td>200000</td><td>22504</td></tr>

```

Toutes ces explications sont un peu rudimentaires mais il devrait apparaître maintenant au lecteur que le texte XML suivant :

```

<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>

```

accompagné de la feuille de style XSL **simulations.xsl** suivante :

```

<?xml version="1.0" encoding="windows-1252"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Simulations de calculs d'impôts</title>

```

```

</head>
<body>
  <center>
    <h3>Simulations de calculs d'impôts</h3>
    <hr/>
    <table border="1">
      <th>marié</th><th>enfants</th><th>salaire</th><th>impôt</th>
      <xsl:apply-templates select="/simulations/simulation"/>
    </table>
  </center>
</body>
</html>
</xsl:template>

<xsl:template match="simulation">
  <tr>
    <td><xsl:value-of select="@marie"/></td>
    <td><xsl:value-of select="@enfants"/></td>
    <td><xsl:value-of select="@salaire"/></td>
    <td><xsl:value-of select="@impot"/></td>
  </tr>
</xsl:template>
</xsl:stylesheet>

```

produit le texte HTML suivant :

```

<html>
<head>
<title>Simulations de calculs d'impôts</title>
</head>
<body>
<center>
<h3>Simulations de calculs d'impôts</h3>
<hr>
<table border="1">
<th>marié</th><th>enfants</th><th>salaire</th><th>impôt</th>
<tr>
<td>oui</td><td>2</td><td>200000</td><td>22504</td>
</tr>
<tr>
<td>non</td><td>2</td><td>200000</td><td>33388</td>
</tr>
</table>
</center>
</body>
</html>

```

Le fichier XML **simulations.xml** suivant

```

<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>

```

lu par un navigateur récent (ici Netscape 7) est alors affiché comme suit :



Simulations de calculs d'impôts

marié	enfants	salaire	impôt
oui	2	200000	22504
non	2	200000	33388

Application impôts : version 5

4.2.1 Les fichiers XML et feuilles de style XSL de l'application impôts

Revenons à notre point de départ qui était l'application web **impôts** et rappelons que nous voulons la modifier afin que la réponse faite aux clients soit une réponse au format XML plutôt qu'une réponse HTML. Cette réponse HTML sera accompagnée d'une feuille de style XSL afin que les navigateurs puissent l'afficher. Dans le paragraphe précédent, nous avons présenté :

- le fichier **simulations.xml** qui est le prototype d'une réponse XML comportant des simulations de calculs d'impôts
- le fichier **simulations.xsl** qui sera la feuille de style XSL qui accompagnera cette réponse XML

Il nous faut prévoir également le cas de la réponse avec des erreurs. Le prototype de la réponse XML dans ce cas sera le fichier **erreurs.xml** suivant :

```
<?xml version="1.0" encoding="windows-1252"?>
<?xml-stylesheet type="text/xsl" href="erreurs.xsl"?>
<erreurs>
  <erreur>erreur 1</erreur>
  <erreur>erreur 2</erreur>
</erreurs>
```

La feuille de style **erreurs.xsl** permettant d'afficher ce document XML dans un navigateur sera la suivante :

```
<?xml version="1.0" encoding="windows-1252"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes"/>

  <xsl:template match="/">
    <html>
      <head>
        <title>Simulations de calculs d'impôts</title>
      </head>
      <body>
        <center>
          <h3>Simulations de calculs d'impôts</h3>
        </center>
        <hr/>
        Les erreurs suivantes se sont produites :
        <ul>
          <xsl:apply-templates select="/erreurs/erreur"/>
        </ul>
      </body>
    </html>
  </xsl:template>

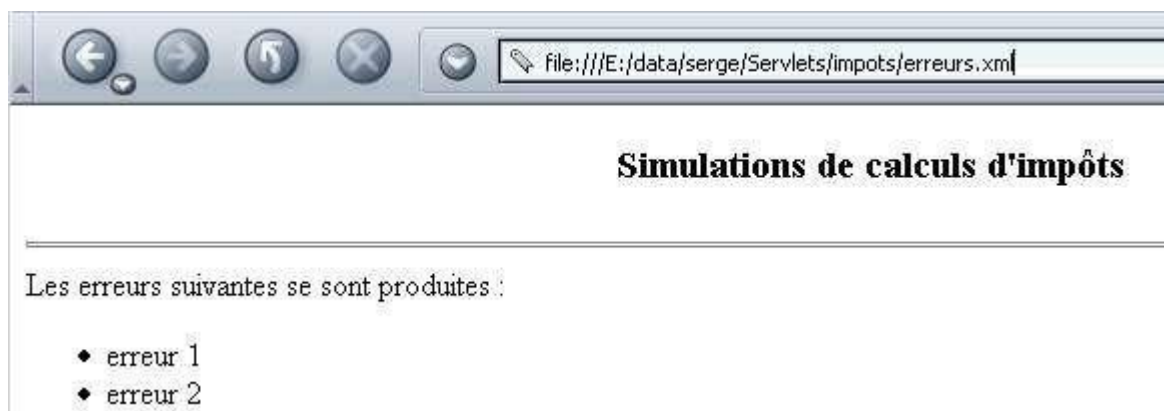
  <xsl:template match="erreur">
    <li><xsl:value-of select="."/></li>
  </xsl:template>
</xsl:stylesheet>
```

Cette feuille de style introduit une commande XSL non encore rencontrée : `<xsl:value-of select="."/>`. Cette commande produit en sortie la valeur du noeud analysé, ici un noeud `<erreur>texte</erreur>`. La valeur de ce noeud est le texte compris entre les deux balises d'ouverture et de fermeture, ici *texte*.

Le code **erreurs.xml** est transformé par la feuille de style **erreurs.xsl** en document HTML suivant :

```
<html>
<head>
<title>Simulations de calculs d'impôts</title>
</head>
<body>
<center>
<h3>Simulations de calculs d'impôts</h3>
</center>
<hr>
    Les erreurs suivantes se sont produites :
    <ul>
<li>erreur 1</li>
<li>erreur 2</li>
</ul>
</body>
</html>
```

Le fichier **erreurs.xml** accompagné de sa feuille de style est affiché par un navigateur de la façon suivante :



4.2.2 L'application xmlsimulations

Nous créons un fichier **xmlsimulations.html** que nous mettons dans le répertoire de l'application **impots**. La page visualisée est la suivante :

http://localhost/poly/impots/7/xmlsimulations.html

Calcul d'impôts

Etes-vous marié(e) ☐ oui ☒ non

Nombre d'enfants

Salaire annuel

Ce document HTML est un document **statique**. Son code est le suivant :

```
<html>
```

* *

* *

```

        alert("Le salaire n'a pas été donné ou est incorrect");
        salaire.focus();
        return;
    } //if
    // c'est bon - on envoie
    submit();
} //with
} //calculer
</script>
</head>

<body background="/poly/impots/7/images/standard.jpg">
<center>
    Calcul d'impôts
    <hr>

    <form name="frmImpots" action="/poly/impots/7/xmlsimulations.php" method="POST">
    <table>
    <tr>
    <td>Etes-vous marié(e)</td>
    <td>
        <input type="radio" name="optMarie" value="oui">oui
        <input type="radio" name="optMarie" value="non" checked>non
    </td>
    </tr>
    <tr>
    <td>Nombre d'enfants</td>
    <td><input type="text" size="3" name="txtEnfants" value=""></td>
    </tr>
    <tr>
    <td>Salaire annuel</td>
    <td><input type="text" size="10" name="txtSalaire" value=""></td>
    </tr>
    <tr>
    <td><input type="button" value="Calculer" onclick="calculer()"></td>
    <td><input type="button" value="Effacer" onclick="effacer()"></td>
    </tr>
    </table>
    </form>
</center>
</body>
</html>

```

On notera que les données du formulaire sont postées à l'URL **/poly/impots/7/xmlsimulations.php**. Le code de l'application *xmlsimulations.php* est très analogue à celui de l'application *impots.php*. Le lecteur est invité à revoir celui-ci. Rappelons le code d'entrée :

```

<?php
// traite le formulaire des impôts

// bibilothèques
include "ImpotsDSN.php";

// démarrage session
session_start();

// configuration de l'application
ini_set("register_globals", "off");
ini_set("display_errors", "off");

$formulaireImpots="impots_form.php";
$erreursImpots="impots_erreurs.php";
$dbImpots=array(dsn=>"mysql-dbimpots", user=>admimpots, pwd=>mdpimpots,
    table=>impots, limites=>limites, coeffR=>coeffR, coeffN=>coeffN);

// on récupère les paramètres de la session
$session=$_SESSION["session"];
// session valide ?
if(! isset($session) || ! isset($session[objImpots]) || ! isset($session[simulations])){
    // on démarre une nouvelle session
    $session=array(objImpots=>new ImpotsDSN($dbImpots), simulations=>array());
    // des erreurs ? if(count($session[objImpots]-
    >erreurs)!=0){
        $requête=array(erreurs=>$session[objImpots]->erreurs);
        // affichage page d'erreurs
        include $erreursImpots;
    } //fin
    $session=array();
    terminerSession($session);
} //if
} //if

// on récupère les paramètres de l'échange en cours
$requête[marié]=$_POST["optMarie"];

```

```

$requete[enfants]=$_POST["txtEnfants"];
$requete[salaire]=$_POST["txtSalaire"];

// a-t-on tous les paramètres ?
if(! isset($requete[marié]) || ! isset($requete[enfants]) || ! isset($requete[salaire])) {

    // affichage formulaire vide
    $requete=array(chkoui=>"",chknon=>"checked",enfants=>"",salaire=>"",impots=>"",
    erreurs=>array(),simulations=>array());
    include $formulaireImpots;

    // fin
    terminerSession($session);
} //if

// vérification paramètres
$requete=vérifier($requete);

// des erreurs ?
if(count($requete[erreurs])!=0) {

    // affichage formulaire
    include "$formulaireImpots";

    // fin
    terminerSession($session);
} //if

// calcul de l'impôt à payer
$requete[impots]=$session[objImpots]->calculer(array(marié=>$requete[marié],
enfants=>$requete[enfants], salaire=>$requete[salaire]));

// une simulation de plus
$session[simulations][]=array($requete[marié],$requete[enfants],$requete[salaire],$requete[impots]);
$requete[simulations]=$session[simulations];

// affichage formulaire
include "$formulaireImpots";

// fin
terminerSession($session);
...

```

Les pages HTML étaient affichées par les lignes *include "..."*. On veut ici générer du XML et non du HTML. Il suffit d'écrire deux nouvelles applications *impots_erreurs.php* et *impots_simulations.php* pour qu'elle génèrent du XML au lieu du HTML. Le reste de l'application ne change pas. Le code devient alors le suivant :

```

<?php
// traite le formulaire des impôts

// bibilothèques
include "ImpotsDSN.php";

// démarrage session
session_start();

// configuration de l'application
ini_set("register_globals","off");
ini_set("display_errors","off");

$formulaireImpots="xmlsimulations.html";
$erreursImpots="impots_erreurs.php";
$simulationsImpots="impots_simulations.php";

$dbImpots=array(dsn=>"mysql-dbimpots",user=>adimpots,pwd=>mdpimpots,
table=>impots,limites=>limites,coeffR=>coeffR,coeffN=>coeffN);

// on récupère les paramètres de la session
$session=$_SESSION["session"];
// session valide ?
if(! isset($session) || ! isset($session[objImpots]) || ! isset($session[simulations])) {
    // on démarre une nouvelle session
    $session=array(objImpots=>new ImpotsDSN($dbImpots),simulations=>array());
    // des erreurs ? if(count($session[objImpots]-
    >erreurs)!=0) {
        $requete=array(erreurs=>$session[objImpots]->erreurs);

        // affichage page d'erreurs au format XML
        header("Content-type: text/xml");
        include $erreursImpots;

        // fin
        $session=array();
        terminerSession($session);
    } //if
} //if

// on récupère les paramètres de l'échange en cours
$requete[marié]=$_POST["optMarie"];

```

```

$requete[enfants]=$_POST["txtEnfants"];
$requete[salaire]=$_POST["txtSalaire"];
// a-t-on tous les paramètres ?
if(! isset($requete[marié]) || ! isset($requete[enfants]) || ! isset($requete[salaire])) {
    // affichage formulaire vide
    include $formulaireImpots;
    // fin
    terminerSession($session);
} //if

// vérification paramètres
$requete=vérifier($requete);

// des erreurs ?
if(count($requete[erreurs])!=0) {
    // affichage erreurs au format XML
    header("Content-type: text/xml");
    include "$erreursImpots";
    // fin
    terminerSession($session);
} //if

// calcul de l'impôt à payer
$requete[impots]=$session[objImpots]->calculer(array(marié=>$requete[marié],
    enfants=>$requete[enfants], salaire=>$requete[salaire]));

// une simulation de plus
$session[simulations][]=array($requete[marié], $requete[enfants], $requete[salaire], $requete[impots]);
$requete[simulations]=$session[simulations];

// affichage simulations au format XML
header("Content-type: text/xml");
include "$simulationsImpots";

// fin
terminerSession($session);

```

Nous avons précédemment présenté et étudié les deux types de réponse XML à fournir ainsi que les feuilles de style qui doivent les accompagner. Le code de l'application **impots_simulations.php** est le suivant :

```

<?php
// génère le code XML de la page de simulations de l'application impots

// qqs constantes
$xmlSimulations="simulations.xml";

// entêtes XML
echo "<?xml version='1.0' encoding='ISO-8859-1' ?>\n";
echo "<?xml-stylesheet type='text/xsl' href='\"$xmlSimulations\"' ?>\n";
// les simulations echo
"<simulations>\n";
for ($i=0;$i<count($requete[simulations]);$i++) {
    // simulation $i
    echo "<simulation marie='\"$requete[simulations][$i][0].\"' ".
        "enfants='\"$requete[simulations][$i][1].\"' ".
        "salaire='\"$requete[simulations][$i][2].\"' ".
        "impot='\"$requete[simulations][$i][3].\"' />\n";
} //for
echo "</simulations>\n";
?>

```

Ce code permet à partir du dictionnaire *\$requete* de générer du code XML analogue au suivant :

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="simulations.xml" ?>
<simulations>
<simulation marie="non" enfants="3" salaire="200000" impot="22504" />
<simulation marie="oui" enfants="3" salaire="200000" impot="16400" />
<simulation marie="oui" enfants="2" salaire="200000" impot="22504" />
</simulations>

```

La feuille de style *simulations.xsl* transformera ce code XML en code HTML.

Le code de l'application **impots_erreurs.php** est le suivant :

```

<?php
// génère le code XML de la page d'erreurs de l'application impots

// qqs constantes

```



```

$xmlErreurs="erreurs.xml";
// entêtes XML
echo "<?xml version='1.0' encoding='ISO-8859-1' ?>\n";
echo "<?xml-stylesheet type='text/xsl' href='\"$xmlErreurs\"' ?>\n";
// les erreurs
echo "<erreurs>\n";
for ($i=0;$i<count($requete[erreurs]);$i++) {
    // erreur $i
    echo "<erreur>". $requete[erreurs][$i]. "</erreur>";
} //for
echo "</erreurs>\n";
?>

```

Ce code permet à partir du dictionnaire *\$requete* de générer du code XML analogue au suivant :

```

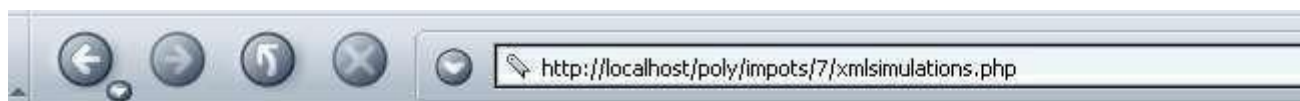
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="erreurs.xml" ?>
<erreurs>
<erreur>Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)</erreur>
</erreurs>

```

La feuille de style *erreurs.xml* transformera ce code XML en code HTML.

Voyons un premier exemple :

Le SGBD MySQL n'est pas lancé. On reçoit alors la réponse suivante :



Simulations de calculs d'impôts

Les erreurs suivantes se sont produites :

- ◆ Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)

Si on regarde le code source reçu par le navigateur, on a la chose suivante :

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="erreurs.xml" ?>
<erreurs>
<erreur>Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)</erreur>
</erreurs>

```

Maintenant, nous lançons le SGBD MySQL et faisons différentes simulations successives. Nous obtenons la réponse suivante :

http://localhost/poly/impots/7/xmlsimulations.php

Simulations de calculs d'impôts

marié	enfants	salaire	impôt
oui	2	200000	22504
non	2	200000	33388
non	3	200000	22504
oui	3	200000	16400

Le code reçu par le navigateur est le suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl" ?>
<simulations>
<simulation marie="oui" enfants="2" salaire="200000" impot="22504" />
<simulation marie="non" enfants="2" salaire="200000" impot="33388" />
<simulation marie="non" enfants="3" salaire="200000" impot="22504" />
<simulation marie="oui" enfants="3" salaire="200000" impot="16400" />
</simulations>
```

On remarquera que notre nouvelle application est plus simple à maintenir que la précédente. Une partie du travail a été transférée sur les feuilles de style XSL. L'intérêt de cette nouvelle répartition des tâches est qu'une fois le format XML des réponses a été fixé, le développement des feuilles de style est indépendant de celui de l'application.

Analyse d'un document XML en PHP

La prochaine version de notre application **impots** va être un client programmé de l'application précédente *xmlsimulations*. Notre client va donc recevoir du code XML qu'il devra analyser pour en retirer les informations qui l'intéresse. Nous allons faire ici une pause dans nos différentes versions et apprendre comment on peut analyser un document XML en PHP. Nous le ferons à partir de l'exemple suivant :

```
dos>e:\php43\php.exe xmlParser.php
syntaxe : xmlParser.php fichierXML
```

L'application *xmlParser.php* admet un paramètre : l'URI (Uniform Resource Identifier) du document XML à analyser. Dans notre exemple, cette URI sera simplement le nom d'un fichier XML placé dans le répertoire de l'application *xmlParser.php*. Considérons deux exemples d'exécution. Dans le premier exemple, le fichier XML analysé est le fichier **erreurs.xml** suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="erreurs.xsl"?>
<erreurs>
  <erreur>erreur 1</erreur>
  <erreur>erreur 2</erreur>
</erreurs>
```

L'analyse donne les résultats suivants :

```
dos>e:\php43\php.exe xmlParser.php erreurs.xml
ERREURS
ERREUR
[erreur 1]
/ERREUR
ERREUR
[erreur 2]
/ERREUR
/ERREURS
```

Nous n'avions pas encore écrit ce que faisait l'application *xmlParser.php* mais l'on voit ici qu'elle affiche la structure du document XML analysé. Le second exemple analyse le fichier XML **simulations.xml** suivant:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="simulations.xsl"?>
<simulations>
  <simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
  <simulation marie="non" enfants="2" salaire="200000" impot="33388"/>
</simulations>
```

L'analyse donne les résultats suivants :

```
dos>e:\php43\php.exe xmlParser.php simulations.xml
SIMULATIONS
SIMULATION, (MARIE,oui) (ENFANTS,2) (SALAIRE,200000) (IMPOT,22504)
/SIMULATION
SIMULATION, (MARIE,non) (ENFANTS,2) (SALAIRE,200000) (IMPOT,33388)
/SIMULATION
/SIMULATIONS
```

L'application *xmlParser.php* contient tout ce dont nous avons besoin dans notre application **impots** puisqu'elle a été capable de retrouver soit les erreurs, soit les simulations que pourrait envoyer le serveur web. Examinons son code :

```
<?
// syntaxe $0 fichierXML
// affiche la structure et le contenu du fichier fichierXML

// vérification appel
if(count($argv)!=2){
  // msg d'erreur
  fwrite(STDERR,"syntaxe : $argv[0] fichierXML\n");
  // fin
  exit(1);
}

// initialisations
$file=$argv[1];           // le fichier xml
$depth=0;                 // niveau d'indentation=profondeur dans l'arborescence

// le programme

// on crée un objet d'analyse de texte xml
$xml_parser=xml_parser_create();

// on indique quelles fonctions exécuter en début et fin de balise
xml_set_element_handler($xml_parser,"startElement","endElement");

// on indique quelle fonction exécuter lors de la rencontre de données
xml_set_character_data_handler($xml_parser,"afficheData");

// ouverture du fichier xml en lecture if (!
($fp=@fopen($file,"r"))){
  fwrite(STDERR,"impossible d'ouvrir le fichier xml $file\n"); exit(2);
}

// exploitation du fichier xml par blocs de 4096 octets
while($data=fread($fp,4096)){
  // analyse des données lues
  if (! xml_parse($xml_parser,$data,feof($fp))){
    // il s'est produit une erreur
    fprintf(STDERR,"erreur XML : %s à la ligne %d\n",
      xml_error_string(xml_get_error_code($xml_parser)),
      xml_get_current_line_number($xml_parser));
    // fin
    exit(3);
  }
}

// le fichier a été exploré

// on libère les ressources occupées par l'analyseur xml
xml_parser_free($xml_parser);

// fin
exit(0);

// -----
// fonction appelée lors de la rencontre d'une balise de début
function startElement($parser,$name,$attributs){
  global $depth;
```

```
// une suite d'espaces (indentation)
for($i=0;$i<$depth;$i++){
    print " ";
}
// attributs
$precisions="";
while(list($attrib,$valeur)=each($attributs)){
    $precisions.="($attrib,$valeur) ";
}
// on affiche le nom de la balise et les éventuels attributs
if($precisions)
    print "$name,$precisions\n";
else print "$name\n";
// un niveau d'arborescence en plus
$depth++;
}
//startElement
```

```
// -----
// la fonction appelée lors de la rencontre d'une balise de fin
```

```
function endElement($parser,$name){
    // fin de balise
    // niveau d'indentation
    global $depth;
    $depth--;
    // une suite d'espaces (indentation)
    for($i=0;$i<$depth;$i++){
        echo " ";
    }
    // le nom de la balise
    echo "/$name\n";
}
//endElement
```

```
// -----
// la fonction d'affichage des données
```

```
function afficheData($parser,$data){
    // niveau d'indentation
    global $depth;
    // les données sont affichées
    $data=trim($data);
    if($data!=""){
        // une suite d'espaces (indentation)
        for($i=0;$i<$depth;$i++){
            echo " ";
        }
        echo "[$data]\n";
    }
}
//afficheData
```

```
?>
```

Etudions le code se rapportant à XML. Pour analyser un document XML, notre application a besoin d'un analyseur de code XML appelé habituellement un "parseur".

```
// on crée un objet d'analyse de texte xml
$xml_parser=xml_parser_create();
```

Lorsque le parseur va analyser le document XML, il va émettre des événements tels que : j'ai rencontré le début du document, le début d'une balise, un attribut de balise, le contenu d'une balise, la fin d'une balise, la fin du document, ... Il transmet ces événements à des méthodes qu'on doit lui désigner :

```
// on indique quelles fonctions exécuter en début et fin de balise
xml_set_element_handler($xml_parser,"startElement","endElement");
// on indique quelle fonction exécuter lors de la rencontre de données
xml_set_character_data_handler($xml_parser,"afficheData");
```

événement émis par le parseur	méthode de traitement
début d'un élément : <balise>	function startElement(\$parser,\$name,\$attributs) \$parser : le parseur du document \$name : nom de l'élément analysé. Si l'élément rencontré est <simulations>, on aura name="simulations". \$attributs : liste des attributs de la balise sous la forme (ATTRIBUT,valeur) où ATTRIBUT est le nom en majuscules de la balise.

valeur d'un élément :
<balise>valeur</balise>

function afficheData(\$parser,\$data)

\$parser : le parseur du document

\$data : les données de la balise

fin d'un élément :
</balise> ou <balise .../
>

function endElement(\$parser,\$name)

les paramètres sont ceux de la méthode *startElement*.

La fonction *startElement* permet de récupérer les attributs de l'élément grâce au paramètre **\$attributs**. Celui-ci est un dictionnaire des attributs de la balise. Ainsi si on a la balise suivante :

```
<simulation marie="oui" enfants="2" salaire="200000" impot="22504"/>
```

le dictionnaire **\$attributs** sera le suivant : **array(marie=>oui,enfants=>2,salaire=>200000,impot=>22504)**

Une fois définis le parseur, et les méthodes précédentes, l'analyse d'un document se fait la fonction **xml_parse** :

```
// exploitation du fichier xml par blocs de 4096 octets
while($data=fread($fp, 4096)) {
    // analyse des données lues
    if (! xml_parse($xml_parser, $data, feof($fp))) {
        ...
    } //if
} //while
```

analyse d'un
document \$doc

function xml_parse(\$parser,\$doc,\$fin)

le parseur **\$parser** analyse le document **\$doc**. **\$doc** peut être un morceau d'un document plus important. Le paramètre **\$fin** indique si c'est le dernier morceau (true) ou non (false). Lors de l'analyse du document **\$doc**, les fonctions définies par **xml_set_element_handler** sont appelées à chaque début et fin de balise. La fonction définie par **xml_set_character_data_handler** est elle appelée à chaque fois que le contenu d'une balise a été obtenu.

Pendant l'analyse du document XML, il peut se produire des erreurs notamment si le document XML est "mal formé" par exemple lors de l'oubli de balises de fermeture. Dans ce cas, la fonction **xml_parse** rend une valeur évaluée à "faux" :

```
if (! xml_parse($xml_parser, $data, feof($fp))) {
    // il s'est produit une erreur fprintf(STDERR, "erreur XML :
    %s à la ligne %d\n",
        xml_error_string(xml_get_error_code($xml_parser)),
        xml_get_current_line_number($xml_parser));
    // fin
    exit(3);
} //if
```

gestion
d'erreurs

function xml_get_error_code(\$parser)

rend le code de la dernière erreur survenue

function xml_error_string(\$code)

rend le message d'erreur associé au code passé en paramètre

function xml_get_current_line(\$parser)

rend le n° de la ligne du document XML en cours d'analyse par le parseur

Lorsque le document a été analysé, on libère les ressources attribuées au parseur :

```
// on libère les ressources occupées par l'analyseur xml
xml_parser_free($xml_parser);
```

libération
des
ressources attribuées
au parseur **\$parser**

function xml_free(\$parser)

Une fois ceci expliqué, le programme précédent accompagné des exemples d'exécution se comprend de lui-même.

Application impôts : version 6

Nous avons maintenant tous les éléments pour écrire un client programmé pour notre service d'impôts qui délivre du XML. Nous reprenons la version 4 de notre application pour faire le client et nous gardons la version 5 pour ce qui est du serveur. Dans cette application client-serveur :

- le service de simulations du calcul d'impôts est fait par l'application **xmlsimulations.php**. La réponse du serveur est donc au format XML comme nous l'avons vu dans la version 5.
- le client n'est plus un navigateur mais un client php autonome. Son interface console est celle de la version 4.

Le lecteur est invité à relire le code de l'application **cltImpots.php** qui était le client programmé de la version 4. Celui-ci recevait un document **\$document** du serveur. Celui-ci était alors un document HTML. C'est maintenant un document XML. Le document HTML *\$document* était analysé par la fonction suivante :

```
function getInfos($document) {
// $document : document HTML
// on cherche soit la liste d'erreurs
// soit le tableau des simulations

// préparation du résultat
$impots[erreurs]=array();
$impots[simulations]=array();

.....
return $impots;
} //getInfos
```

La fonction recevait le document HTML *\$document*, l'analysait et rendait un dictionnaire **\$impots** à deux attributs :

1. **erreurs** : un tableau d'erreurs
2. **simulations** : un tableau de simulations, chaque simulation étant elle-même un tableau à quatre éléments (marie,enfants,salaire,impot).

L'application **cltImpots.php** devient maintenant **cltXmlSimulations.php**. Seule la partie qui traite le document reçu du serveur doit changer pour tenir compte du fait que c'est maintenant un document XML. La fonction **getInfos** devient alors la suivante :

```
// -----
function getInfos($document) {
// $document : document XML
// on cherche soit la liste d'erreurs
// soit le tableau des simulations global

$impots,$balises;

// préparation du résultat
$impots[erreurs]=array();
$impots[simulations]=array();
// balises en cours
$balises=array();

// on crée un objet d'analyse de texte xml
$xml_parser=xml_parser_create();
// on indique quelles fonctions exécuter en début et fin de balise
xml_set_element_handler($xml_parser,"startElement","endElement");
// on indique quelle fonction exécuter lors de la rencontre de données
xml_set_character_data_handler($xml_parser,"getData");
// on fait l'analyse de $document
xml_parse($xml_parser,$document);
// on libère les ressources occupées par l'analyseur xml
xml_parser_free($xml_parser);

// fin
return $impots;
} //getInfos

// -----
// fonction appelée lors de la rencontre d'une balise de début fonction
startElement($parser,$name,$attributs) {
global $impots,$balise,$balises,$contenu;

// on note le nom de la balise et son contenu
$balise=strtolower($name);
$contenu="";
// on l'ajoute à la pile des balises
array_push($balises,$balise);
// s'agit-il d'une simulation ?
if($balise=="simulation") {
// on note les attributs de la simulation
$impots[simulations][]=array($attributs[MARIE],$attributs[ENFANTS],$attributs[SALAIRE],$attributs [IMPOT]);
}
```

```

    }//if
  }//startElement

  // -----
  // la fonction appelée lors de la rencontre d'une balise defin fonction
  endElement($parser,$name){
    // on récupère la balise courante global
    $impots,$balises,$contenu;
    $balise=array_pop($balises);
    // s'agit-il d'une balise d'erreur ?
    if($balise=="erreur"){
      // une erreur de plus
      $impots[erreurs][]=trim($contenu);
    }//if
  }//endElement

  // -----
  // la fonction de traitement du contenu d'une balise fonction
  getData($parser,$data){
    // données globales global
    $balise,$contenu;
    // s'agit-il d'une balise d'erreur ?
    if($balise=="erreur"){
      // on cumule au contenu de la balise courante
      $contenu.=$data;
    }//if
  }//getData
}

```

Commentaires :

- la fonction **getInfos(\$document)** commence par créer un parseur, puis le configure et enfin lance l'analyse du document **\$document** :

```

// on crée un objet d'analyse de texte xml
$xml_parser=xml_parser_create();
// on indique quelles fonctions exécuter en début et fin de balise
xml_set_element_handler($xml_parser,"startElement","endElement");
// on indique quelle fonction exécuter lors de la rencontre de données
xml_set_character_data_handler($xml_parser,"getData");
// on fait l'analyse de $document
xml_parse($xml_parser,$document);

```

- au retour de l'analyse, on libère les ressources allouées au parseur et on rend le dictionnaire **\$impots**.

```

// on libère les ressources occupées par l'analyseur xml
xml_parser_free($xml_parser);
// fin
return $impots;

```

- la fonction **startElement(\$parser,\$name,\$attributs)** est appelée à chaque début de balise. Elle
 - note la balise **\$name** dans un tableau de balises **\$balises**. Ce tableau est géré comme une pile : à la rencontre du symbole de fin de balise, la dernière balise empilée dans **\$balises** sera dépilée. La balise courante est également notée dans **\$balise**. Dans le dictionnaire **\$attributs**, on trouve les attributs de la balise rencontrée, ces attributs étant en **majuscules**.
 - enregistre les attributs dans le dictionnaire global **\$impots[simulations]**, s'il s'agit d'une balise de simulation.
- la fonction **getData(\$parser,\$data)** lorsque le contenu **\$data** d'une balise a été récupéré. Ici, on a pris une précaution. Dans certaines API, notamment Java, de traitement de documents XML, il est indiqué que cette fonction peut être appelée de façon répétée, c.a.d. qu'on n'a pas forcément le contenu **\$data** d'une balise en une seule fois. Ici, la documentation PHP n'indique pas cette restriction. Par précaution, on cumule la valeur **\$data** obtenue à une variable globale **\$contenu**. Ce n'est qu'à la rencontre du symbole de fin de balise qu'on estimera avoir obtenu l'intégralité du contenu de la balise. La seule balise concernée par ce traitement est la balise **<erreur>**.
- la fonction **endElement(\$parser,\$name)** est appelée à chaque fin de balise. Elle est utilisée ici pour changer le nom de la balise courante en enlevant la dernière balise de la pile de balises et pour ajouter le contenu de la balise **<erreur>** qui se termine au tableau **\$impots[erreurs]**.

Voici quelques exemples d'exécution, tout d'abord avec un SGBD qui n'a pas été lancé :

```

dos>e:\php43\php.exe c:\XmlSimulations.php http://localhost/poly/impots/8/xmlsimulations.php oui 2
200000
Jeton de session=[e8c29ea12f79e4771960068d161229fd]
Les erreurs suivantes se sont produites :

```

Impossible d'ouvrir la base DSN [mysql-dbimpots] (S1000)

Puis avec le SGBD lancé :

```
dos>e:\php43\php.exe cltXmlSimulations.php http://localhost/poly/impots/8/xmlsimulations.php oui 3
200000
Jeton de session=[69a54d79db10b70ed0a2d55d5026ac8b]
Simulations :
[oui,3,200000,16400]

dos >e:\php43\php.exe cltXmlSimulations.php http://localhost/poly/impots/8/xmlsimulations.php oui 2
200000 69a54d79db10b70ed0a2d55d5026ac8b
Jeton de session=[69a54d79db10b70ed0a2d55d5026ac8b]
Simulations :
[oui,3,200000,16400]
[oui,2,200000,22504]

dos >e:\php43\php.exe cltXmlSimulations.php http://localhost/poly/impots/8/xmlsimulations.php non 2
200000 69a54d79db10b70ed0a2d55d5026ac8b
Jeton de session=[69a54d79db10b70ed0a2d55d5026ac8b]
Simulations :
[oui,3,200000,16400]
[oui,2,200000,22504]
[non,2,200000,33388]
```

Conclusion

Grâce à sa réponse XML l'application **impots** est devenue plus facile à gérer, à la fois pour son concepteur que pour les concepteurs des application clientes.

- la conception de l'application serveur peut maintenant être confiée à deux types de personnes : le développeur PHP de la servlet et l'infographiste qui va gérer l'apparence de la réponse du navigateur dans les navigateurs. Il suffit à ce dernier de connaître la structure de la réponse XML du serveur pour construire les feuilles de style qui vont l'accompagner. Rappelons que celles-ci font l'objet de fichiers XSL à part et indépendants de l'application PHP. L'infographiste peut donc travailler indépendamment du développeur.
- les concepteurs des applications clientes ont eux aussi simplement besoin de connaître la structure de la réponse XML du serveur. Les modifications que pourrait apporter l'infographiste aux feuilles de style n'ont aucune répercussion sur cette réponse XML qui reste toujours la même. C'est un énorme avantage.
- Comment le développeur peut-il faire évoluer son application PHP sans tout casser ? Tout d'abord, tant que sa réponse XML ne change pas, il peut organiser son application comme il veut. Il peut aussi faire évoluer la réponse XML tant qu'il garde les éléments **<erreur>** et **<simulation>** attendus par ses clients. Il peut ainsi ajouter de nouvelles balises à cette réponse. L'infographiste les prendra en compte dans ses feuilles de style et les navigateurs pourront avoir les nouvelles versions de la réponse. Les clients programmés eux continueront à fonctionner avec l'ancien modèle, les nouvelles balises étant tout simplement ignorées. Pour que cela soit possible, il faut que dans l'analyse XML de la réponse du serveur, les balises cherchées soient bien identifiées. C'est ce qui a été fait dans notre client XML de l'application **impôts** où dans les procédures on disait spécifiquement qu'on traitait les balises **<erreur>** et **<simulation>**. Du coup les autres balises sont ignorées.

suivre...

Nous avons donné dans ce document suffisamment d'informations pour démarrer sérieusement la programmation web en PHP. Certains thèmes n'ont été qu'effleurés et mériteraient des approfondissements :

- **XML (eXtensible Markup Language), Feuilles de style CSS (Cascading Style Sheets) et XSL (eXtensible Stylesheet Language)**
Nous avons abordé ces thème mais beaucoup reste à faire. Ils méritent eux trois un polycopié.
- **Javascript**
Nous avons utilisé ici et là quelques scripts Javascript exécutés par le navigateur sans jamais s'y attarder. Nous ne sommes par exemple jamais entrés dans les détails du langage Javascript. Par manque de place et de temps. Ce langage peut nécessiter un livre à lui tout seul. Nous avons rajouté en annexe, trois exemples significatifs en javascript là encore

sans les expliquer. Néanmoins ils sont compréhensibles par la simple lecture des commentaires et peuvent servir d'exemples.

6. Etude de cas - Gestion d'une base d'articles sur le web

Objectifs :

- écrire une classe de gestion d'une base de données d'articles
- écrire une application web s'appuyant sur cette classe
- introduire les feuilles de style
- proposer un début de méthodologie de développement pour des applications web simples
- introduire javascript dans le navigateur client

Crédits : L'essence de cette étude de cas a été trouvée dans le livre "Les cahiers du programmeur - PHP/MySQL" de Jean-Philippe Leboeuf aux éditions Eyrolles.

Introduction

Un commerçant désire gérer les articles qu'il vend en magasin. Il a déjà chez lui une application ACCESS qui fait ce travail mais il est tenté par l'aventure du web. Il a un compte chez un fournisseur d'accès internet, fournisseur autorisant ses clients à installer des scripts PHP dans leurs dossiers personnels. Ceci leur permet de créer des sites web dynamiques. Par ailleurs, ces mêmes clients disposent d'un compte MySQL leur permettant de créer des tables pouvant servir des données à leurs scripts PHP. Ainsi le commerçant a un compte MySQL ayant pour login **admarticles** et mot de passe **mdparticles**. Il possède une base **dbarticles** sur laquelle il a tous les droits. Notre commerçant a donc les éléments suffisants pour mettre sa gestion d'articles sur le web. Aidé par vous qui avez des compétences dans le développement Web, il se lance dans l'aventure.

La base des données

Notre commerçant fait la maquette suivante de l'interface web d'accueil qu'il souhaiterait :

La maquette d'interface web d'accueil est présentée sur un fond à motif de losanges. Elle est structurée comme suit :

- En-tête :** Une barre bleue horizontale avec le titre "Gestion d'articles" en rouge.
- Menu de gauche :**
 - Un bouton orange "Authentification".
 - Un bouton orange "Utilisation" suivi d'une liste à puces : "Ajouter un article", "Modifier un article", "Supprimer un article", "Lister des articles".
 - Un bouton orange "Administration" suivi d'une liste à puces : "Requête SQL".
- Section de connexion :** Un rectangle orange intitulé "Connexion à la base" contenant :
 - Deux champs de saisie étiquetés "login" et "mot de passe".
 - Un bouton rouge "Connexion" à droite des champs.

Il y aurait deux sortes d'utilisateurs :

- des **administrateurs** qui pourraient tout faire sur la table des articles (ajouter, modifier, supprimer, consulter, ...). Ceux-là pourront utiliser tous les éléments du menu ci-dessus. En particulier, ils pourront émettre toute requête SQL via l'option [Requête SQL].

- les **utilisateurs** normaux (pas administrateurs) qui auraient des droits restreints : droits d'ajouter, de modifier, de supprimer, de consulter. Ils peuvent n'avoir que certains de ces droits, le seul droit de consultation par exemple.

Du fait qu'il y a divers types d'utilisateurs de la base n'ayant pas les mêmes droits, il y a nécessité d'une authentification. C'est pourquoi la page d'accueil commence avec celle-ci. Pour savoir qui est qui, et qui a le droit de faire quoi, deux tables USERS et DROITS seront utilisées. La table USERS aurait la structure suivante :

Field	Type	
<u>login</u>	varchar(10)	login login de l'utilisateur l'identifiant de façon unique. Ce champ est clé primaire de la table.
mdp	varchar(10)	mdp le mot de passe en clair de l'utilisateur
admin	char(1)	admin le caractère 'y' (yes) si l'utilisateur est administrateur, sinon le caractère 'n' (no).

Le contenu de la table pourrait être le suivant :

login	mdp	admin
st	st	y
user1	user1	n
user3	user3	n

La table DROITS précise les droits qu'ont les utilisateurs non administrateurs présents dans la table USERS. Sa structure est la suivante :

Field	Type	
<u>login</u>	varchar(10)	login login de l'utilisateur l'identifiant de façon unique. Ce champ est clé étrangère de la table DROITS et référence la colonne login de la table USERS.
table	varchar(20)	table le nom de la table sur laquelle l'utilisateur a des droits.
ajouter	char(1)	ajouter le caractère 'y' (yes) si l'utilisateur a un droit d'ajout sur la table, sinon le caractère 'n' (no).
modifier	char(1)	modifier droit de modifier : 'y' ou 'n'
supprimer	char(1)	supprimer droit de supprimer : 'y' ou 'n'
consulter	char(1)	consulter droit de consulter : 'y' ou 'n'

Le contenu de la table pourrait être le suivant :

login	table	ajouter	modifier	supprimer	consulter
user1	articles	y	y	y	y
user3	articles	n	n	n	y

Remarques :

- Un utilisateur U se trouvant dans la table USERS et absent de la table DROITS n'a aucun droit.
- Dans notre exemple, les utilisateurs n'auront accès qu'à une seule table, la table ARTICLES. Mais notre commerçant, prévoyant, a néanmoins ajouté le champ **table** à la structure de la table DROITS afin de se donner la possibilité d'ajouter ultérieurement de nouvelles tables à son application.
- Pourquoi gérer des droits dans nos propres tables alors qu'on fait l'hypothèse qu'on utilisera une base MySQL capable elle-même (et mieux que nous) de gérer ces droits dans ses propres tables ? Simplement parce que notre commerçant n'a pas les droits d'administration sur la base MySQL qui lui permettraient de créer des utilisateurs et de leur donner des droits. N'oublions pas en effet que la base MySQL est hébergée chez un fournisseur d'accès et que le commerçant n'est qu'un simple utilisateur de celle-ci sans aucun droit d'administration (heureusement). Il possède cependant tous les droits sur une base appelée **dbarticles** à laquelle il accède pour le moment avec le login **admarticles** et le mot de passe **mdparticles**. C'est dans cette base que prennent place toutes les tables de l'application.

La table ARTICLES rassemble les informations sur les articles vendus par le commerçant. Sa structure est la suivante :

Field	Type	
<u>code</u>	varchar(4)	code de l'article - clé primaire de la table - 4 caractères exactement
nom	varchar(30)	nom de l'article
prix	double	son prix
stockActuel	int(11)	le niveau actuel de son stock
stockMinimum	int(11)	le niveau au-dessous duquel, une commande de réapprovisionnement doit être faite

Son contenu utilisé dans un premier temps comme test pourrait être le suivant :

code	nom	prix	stockActuel	stockMinimum
a100	cartable	20	5	3
b100	trousse	5	20	3
c100	lot de 6 crayons bille	4	30	10
d100	feuilles blanches - lot de 500	10	32	6

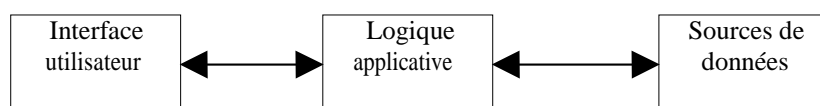
Les contraintes du projet

Le commerçant migre ici une application ACCESS locale en une application Web. Il ne sait ce que deviendra celle-ci et comment elle évoluera. Il voudrait cependant que la nouvelle application soit facile à utiliser et évolutive. C'est pour cette raison que son conseiller informatique a imaginé pour lui, lors de la conception des tables, qu'il pouvait y avoir :

- divers utilisateurs avec différents droits : cela permettra au commerçant de déléguer certaines tâches à d'autres personnes sans pour autant leur donner des droits d'administration
- dans le futur d'autres tables que la table ARTICLES

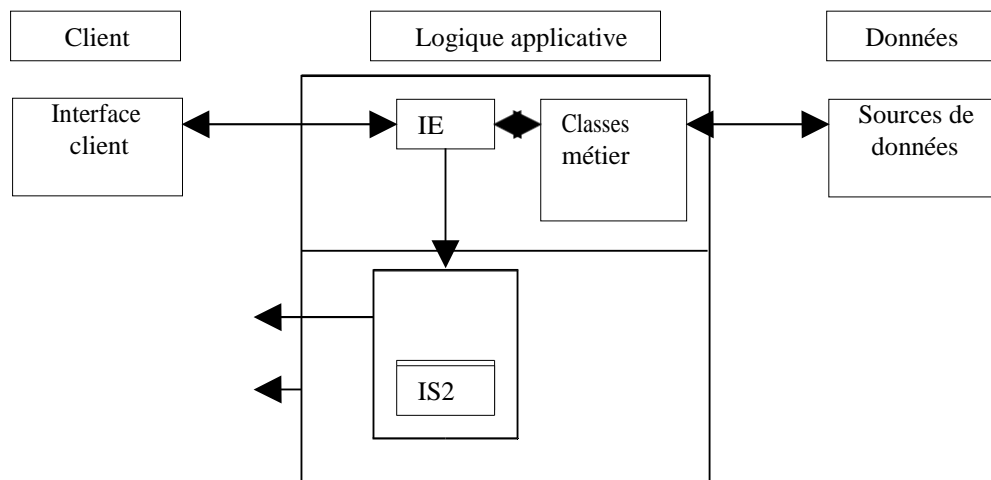
Le même conseiller fait d'autres propositions :

- il sait que dans le développement logiciel, il faut séparer nettement les couches présentation et les couches traitement. L'architecture d'une application web est souvent la suivante :



L'interface utilisateur est ici un navigateur web mais cela pourrait être également une application autonome qui via le réseau enverrait des requêtes HTTP au service web et mettrait en forme les résultats que celui-ci lui envoie. La **logique applicative** est constituée des scripts traitant les demandes de l'utilisateur, ici des scripts PHP. La **source de données** est souvent une base de données mais cela peut être aussi un annuaire LDAP ou un service web distant. Le développeur a intérêt à maintenir une grande indépendance entre ces trois entités afin que si l'une d'elles change, les deux autres n'aient pas à changer ou peu. Le conseiller informatique du commerçant fait alors les propositions suivantes :

- On mettra la logique métier de l'application dans une classe PHP. Ainsi le bloc [Logique applicative] ci-dessus sera constitué des éléments suivants :



Dans le bloc [Logique Applicative], on pourra distinguer

- le bloc [IE=Interface d'Entrée] qui est la porte d'entrée de l'application. Elle est la même quelque soit le type de client.
- le bloc [Classes métier] qui regroupe des classes nécessaires à la logique de l'application. Elles sont indépendantes du client.
- le bloc des générateurs des pages réponse [IS1 IS2 ... IS=Interface de Sortie]. Chaque générateur est chargé de mettre en forme les résultats fournis par la logique applicative pour un type de client donné : code HTML pour un navigateur ou un téléphone WAP, code XML pour une application autonome, ...

Ce modèle assure une bonne indépendance vis à vis des clients. Que le client change ou qu'on veuille faire évoluer sa façon de présenter les résultats, ce sont les générateurs de sortie [IS] qu'il faudra créer ou adapter.

- Dans une application web, l'indépendance entre la couche présentation et la couche traitement peut être améliorée par l'utilisation de feuilles de style. Celles-ci gouvernent la présentation d'une page Web au sein d'un navigateur. Pour changer cette présentation, il suffit de changer la feuille de style associée. Il n'y a pas à toucher à la logique de traitement. On utilisera donc ici une feuille de style.
- Dans le diagramme ci-dessus, c'est la classe métier qui fera l'interface avec la source de données. Par hypothèse, cette source est ici une base MySQL. Afin de permettre une évolution vers une autre base de données, on utilisera la bibliothèque PEAR qui offre des classes d'accès aux bases de données indépendantes du type réel de celles-ci. Ainsi si notre commerçant s'enrichit au point de pouvoir installer un serveur web IIS de Microsoft dans son entreprise, il pourra remplacer la base MySQL par SQL Server sans avoir (ou très peu) à changer la classe métier.

La classe articles

La classe **articles** pourrait être définie de la façon suivante :

```
<?php
// classe articles travaillant sur une base d'articles composée des tables suivantes
// articles : (code, nom, prix, stockActuel, stockMinimum)
// users : (login, mdp, admin)
// droits : (login, table, ajouter, modifier, supprimer, consulter)

// c'est l'utilisateur de la classe qui doit fournir le login/mdp qui permet de faire toute opération sur la base
// il a donc déjà tous les droits sur la base. Ceci implique qu'il n'y a pas à prendre
// de précautions de sécurité particulières ici

// bibliothèques
require_once 'DB.php';

class articles{
    // attributs
    var $sDSN;           // la chaîne de connexion var
    var $sDatabase;      // le nom de la base
    var $oDB;            // connexion à la base
```

```

var $aErreurs:           // liste d'erreurs
var $oResultats:         // résultat d'une requête select
var $connecté:           // booléen qui indique si on est connecté ou non à la base var
$query:                  // la dernière requête exécutée
var $sUser:              // identité de l'utilisateur de la connexion var
$bAdmin:                 // à vrai si l'utilisateur est administrateur
var $dDroits:            // le dictionnaire de ses droits table ->> array (consulter, ajouter, supprimer, modifier)

// constructeur
function articles($dDSN, $sUser, $sMdp) {
    // $dDSN : dictionnaire définissant la liaison à établir
    // $dDSN['sgbd'] : le type du SGBD auquel il faut se connecter
    // $dDSN['host'] : le nom de la machine hôte qui l'héberge
    // $dDSN['database'] : le nom de la base à laquelle il faut se connecter
    // $dDSN['admin'] : le login du propriétaire de la base à laquelle il faut se connecter
    // $dDSN['mdpadmin'] : son mot de passe
    // $sUser : le login de l'utilisateur qui veut exploiter la base des articles
    // $sMdp : son mot de passe

    // crée dans $oDB une connexion à la base définie par $dDSN sous l'identité de $dDSN['admin']
    // si la connexion réussit et que l'utilisateur $sUser est authentifié
    // charge les droits dans $bAdmin et $dDroits les droits de l'utilisateur $sUser
    // met dans $sDSN la chaîne de connexion à la base
    // met dans $sDataBase le nom de la base à laquelle on se connecte
    // met $connecté à vrai
    // si la connexion échoue ou si l'utilisateur $sUser n'est pas identifié correctement
    // met les msg d'erreurs adéquats dans la liste $aErreurs
    // ferme la connexion si besoin est
    // met $connecté à faux

} // constructeur

// -----
function connect() {
    // (re)connexion à la base

} // connect

// -----
function disconnect() {
    // on ferme la connexion à la base $sDSN

} // disconnect

// -----
function execute($sQuery, $bAdmin) {
    // $sQuery : requête à exécuter
    // $bAdmin : vrai si demande d'exécution en tant qu'administrateur

} // execute

// -----
function addArticle($dArticle) {
    // ajoute un article $dArticle (code, nom, prix, stockActuel, stockMinimum) à la table des articles

} // add

// -----
function modifyArticle($dArticle) {
    // modifie un article $dArticle (code, nom, prix, stockActuel, stockMinimum) de la table des articles

} // update

// -----
function deleteArticle($sCode) {
    // supprime un article de la table des articles
    // dont on a le code $sCode

} // delete

// -----
function verifierArticle(&$dArticle) {
    // vérifie la validité d'un article $dArticle (code, nom, prix, stockActuel, stockMinimum)

} // verifier

// -----
function selectArticles($dQuery) {
    // exécute une requête select la table des articles
    // celle-ci a trois composantes
    // liste des colonnes dans $dQuery['colonnes']
    // filtrage dans $dQuery['where']
    // ordre de présentation dans $dQuery['orderby']

} // selectArticles

// -----
function existeArticle($sCode) {
    // rend TRUE si l'article de code $sCode existe dans la table des articles

} // existeArticle

// -----
function existeUser($sUser, $sMdp) {
    // vérification de l'existence de l'utilisateur $sUser ayant le mot de passe $sMdp
    // rend (int $iErreur, string $sAdmin, hashtable $dDroits)
    // $iErreur = -1 pour toute erreur d'exploitation de la base - la liste $aErreurs est alors renseignée
    // $iErreur = 1 si l'utilisateur n'est pas trouvé (absent ou pas bon mot de passe)

```

```

// $iErreur = 2 si l'utilisateur existe mais n'a aucun droit dans la table des droits
// $iErreur = 3 si l'utilisateur existe et est administrateur
// $iErreur = 0 si l'utilisateur existe et n'est pas administrateur
// $sAdmin="y" ssi l'utilisateur existe et est administrateur ($iErreur==3), sinon il est égal à la chaîne vide
// $dDroits est le dictionnaire des droits de l'utilisateur s'il n'est pas administrateur ($iErreur==0)
// sinon c'est un tableau vide
// les clés du dictionnaire sont les tables sur lesquelles l'utilisateur a des droits
// la valeur associée à cette table est à son tour un dictionnaire où les clés sont les droits
// (consulter, ajouter, modifier, supprimer) et les valeurs les chaînes 'y' (yes) ou 'n' (no) selon les cas
...
} //existeUser
// -----
function getCodes() {
    // rend le tableau des codes
...
} //getCodes
} //classe
?>

```

Commentaires

- la classe **articles** utilise la bibliothèque PEAR::DB pour ses accès à la base de données d'où la commande

```
require_once 'DB.php';
```

Cette inclusion suppose que le script DB.php soit dans l'un des répertoires de l'option **include_path** du fichier de configuration de PHP.

- le constructeur a besoin de savoir à quelle base on se connecte et sous quelle identité. Ces informations lui sont données dans le dictionnaire \$dDSN. Rappelons que l'hypothèse de départ était que la base s'appelait **dbarticles** et qu'elle appartenait à un utilisateur appelé **admarticles** ayant le mot de passe **mdparticles**. Rappelons aussi que cette application autorise plusieurs utilisateurs ayant différents droits. Il y a là une ambiguïté à lever. La connexion est bien ouverte sous l'identité de *admarticles* et au final c'est sous cette identité que seront faites toutes les opérations sur la base *dbarticles* puisque c'est le seul nom que connaît le SGBD MySQL qui ait les droits suffisants pour gérer la base *dbarticles*. Pour "simuler" l'existence de différents utilisateurs, on fera travailler l'utilisateur *admarticles* avec les droits de l'utilisateur dont le login (**\$sUser**) et le mot de passe (**\$sMdp**) sont passés en paramètres au constructeur. Ainsi, avant de faire une opération sur la base des articles, on vérifiera que l'utilisateur (*\$sUser*, *\$sMdp*) a bien les droits de la faire. Si oui, c'est l'utilisateur *admarticles* qui la fera pour lui.
- le login et le mot de passe de l'administrateur de la base des articles doivent être passés au constructeur. C'est une saine précaution. Si on inscrivait en "dur" dans le code de la classe ces deux informations, tout utilisateur de la classe pourrait se faire passer aisément pour administrateur de la base des articles. En effet, une classe PHP n'est pas protégée. Aussi l'attribut **\$bAdmin** de la classe qui indique si l'utilisateur (**\$sUser**, **\$sMdp**) pour lequel on travaille est administrateur ou non pourrait très bien être positionné directement de l'extérieur comme dans l'exemple suivant :

```

$oArticles=new articles($dDSN, $sUser, $sMdp)
// ici $sUser a été reconnu comme un utilisateur non administrateur de la base
$oArticle->bAdmin=TRUE;
// maintenant $sUser est devenu administrateur

```

PHP n'est pas JAVA ou C# et une classe PHP n'est qu'une structure de données un peu plus évoluée qu'un dictionnaire mais qui n'offre pas la sécurité d'une vraie classe où l'attribut *bAdmin* aurait été déclaré privé ou protégé rendant impossible sa modification de l'extérieur. Parce que l'utilisateur de la classe doit connaître le login et le mot de passe de l'administrateur de la base des articles, seul ce dernier peut utiliser la classe. L'opération précédente ne présente donc plus aucun intérêt pour lui. La classe est là uniquement pour lui offrir des facilités de développement. Une conséquence importante est qu'il n'y a pas lieu de prendre des précautions de sécurité. Encore une fois, celui qui utilise la classe *articles* est nécessairement administrateur de la base des articles.

- la classe gère les erreurs de connexion à la base ou tout autre erreur de façon unique en remplissant l'attribut **\$aErreurs** avec le ou des messages d'erreurs. Après chaque opération, l'utilisateur de la classe doit donc vérifier cette liste.
- Les méthodes **addArticle**, **updateArticle**, **deleteArticle**, **selectArticles**, **execute** découlent directement de la maquette de l'interface web présentée précédemment. Elles correspondent en effet aux options du menu proposé. Les méthodes **addArticle** et **modifyArticle** s'appuient sur la méthode **vérifierArticle** pour vérifier que l'article qui va être ajouté ou être modifié a des données correctes. Toujours dans le même esprit, la méthode **existeArticle** permet de vérifier qu'on ne s'apprête pas à ajouter un article qui existe déjà. On pourrait se passer de cette méthode si on utilise une table d'articles où le code est clé primaire. C'est alors le SGBD lui-même qui signalera l'échec de l'ajout pour cause de doublon. Il le dira probablement avec un message d'erreur peu lisible et en anglais.

- Un article à modifier ou à supprimer sera désigné par son code qui est unique. La méthode **getCodes** permet d'obtenir tous ces codes.
- La méthode **disconnect** ferme la connexion à la base, connexion ouverte lors de la construction de l'objet. On ne voit pas l'intérêt ici de la méthode **connect** qui va recréer une connexion avec la base. Cela va permettre d'ouvrir et fermer cette connexion à volonté avec un même objet. L'intérêt n'apparaît qu'en conjonction avec l'application web. Celle-ci va créer un objet **articles** qu'elle va mémoriser dans une session. Si celle-ci va être capable de conserver la plupart des attributs de l'objet au fil des échanges client-serveur successifs, elle n'est pas capable cependant de garder l'attribut représentant la connexion ouverte. Celle-ci devra donc être réouverte à chaque nouvel échange client-serveur. On demandera une connexion persistante afin que la connexion ouverte soit stockée dans un pool de connexions et reste ouverte de façon permanente. Ainsi lorsque le script demandera une nouvelle connexion, celle-ci sera récupérée dans le pool de connexions. On arrive donc au même résultat que si la session avait pu mémoriser la connexion ouverte.
- la méthode **existeUser** permet au constructeur de savoir si l'utilisateur **\$sUser** identifié par le mot de passe **\$sMdp** existe bien. Si oui, la méthode permet de savoir s'il est administrateur ou non (indiqué dans la table USERS) et mémorise cette information dans l'attribut **\$bAdmin**. S'il n'est pas administrateur, la méthode va récupérer ses droits dans la table DROITS et les met dans l'attribut **\$dDroits** qui est un dictionnaire à double indexation : **\$dDroits[\$table][\$droit]** vaut 'y' si l'utilisateur **\$sUser** a le droit **\$droit** sur la table **\$table** et vaut 'n' sinon.

Écrire la classe articles. Les accès à la base de données seront faits à l'aide de la bibliothèque PEAR::DB qui permet de s'affranchir du type exact de la base. On trouvera en annexe les éléments de base de cette bibliothèque.

La structure de l'application WEB

Maintenant que nous avons la classe "métier" de gestion de la base d'articles, nous pouvons utiliser celle-ci dans différents environnements. Il est proposé ici de l'utiliser dans une application web. Découvrons celle-ci au-travers de ces différentes pages :

La page type de l'application

Revenons sur la page d'accueil déjà présentée :

The diagram illustrates the layout of the application's main page. At the top is a blue header bar with the title "Gestion d'articles". Below this, the page is organized into several sections. On the left, there is a vertical menu with three main categories: "Authentification", "Administration", and "Requête SQL". Under "Authentification", there is a sub-section "Utilisation" which lists four options: "Ajouter un article", "Modifier un article", "Supprimer un article", and "Lister des articles". To the right of the "Authentification" section is a box titled "Connexion à la base". This box contains a login form with two input fields: "login" and "mot de passe", and a red button labeled "Connexion".

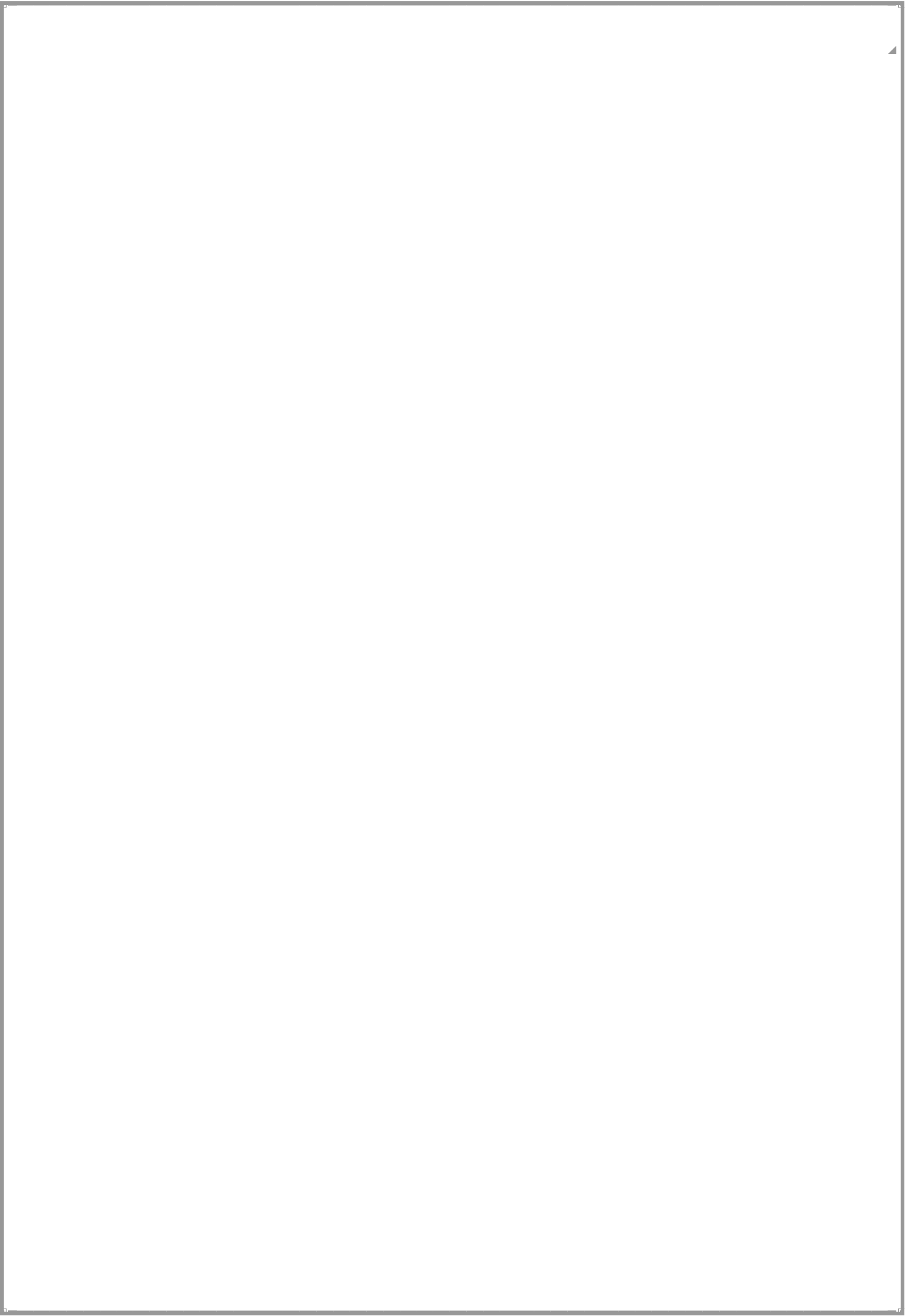
Toutes les pages de l'application auront la structure ci-dessus, celle d'un tableau à deux lignes et trois colonnes comprenant quatre zones :

- la zone 1 forme la première ligne du tableau. Elle est réservée au titre accompagné éventuellement d'une image. Les trois colonnes de la ligne sont ici fusionnées.
- la seconde ligne a trois zones, une zone par colonne :
 - la zone 2 contient les options du menu. Elle contient à son tour un tableau à une colonne et plusieurs lignes. Les options du menu sont placées dans les lignes du tableau.

- o la zone 3 est vide et ne sert qu'à séparer les zones 2 et 4. On aurait pu procéder différemment pour réaliser cette séparation.
- o la zone 4 est celle qui contient la partie dynamique de la page. C'est cette partie qui change d'une action à l'autre, les autres restant identiques.

Le script PHP générant cette page type s'appellera **main.php** et pourrait être le suivant :

```
<html>
<head>
  <title>Gestion d'articles</title>
  <link type="text/css" href="<?php echo $dConfig['urlPageStyle'] ?>" rel="stylesheet" />
</head>
<body background="<?php echo $dConfig['urlBackGround'] ?>">
  <table>
    <tr height="60">
      <td colspan="3" align="left" valign="top" >
        <h1> </h1>
      </td>
    </tr>
    <tr>
      <td>
        <table>
          <tr>
            <td class="menutitle" >
              <a href="<?php echo $main["liens"]["login"] ?>" ?>Authentification</a>
            </td>
          </tr>
          <tr>
            <td><br /></td>
          </tr>
          <tr>
            <td class="menutitle" >
              Utilisation
            </td>
          </tr>
        </table>
      </td>
      <td height="10"></td>
      <td>
        <table>
          <tr>
            <td class="menublock" >
              
              <a href="<?php echo $main["liens"]["addArticle"] ?>" ?> Ajouter un
                article
              </a>
            </td>
          </tr>
          <tr>
            <td class="menublock" >
              
              <a href="<?php echo $main["liens"]["updateArticle"] ?>" ?> Modifier
                un article
              </a>
            </td>
          </tr>
          <tr>
            <td class="menublock" >
              
              <a href="<?php echo $main["liens"]["deleteArticle"] ?>" ?> Supprimer
                un article
              </a>
            </td>
          </tr>
          <tr>
            <td class="menublock" >
              
              <a href="<?php echo $main["liens"]["selectArticle"] ?>" ?> Lister des
                articles
              </a>
            </td>
          </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td><br /></td>
    </tr>
    <tr>
      <td class="menutitle" >
        Administration
      </td>
    </tr>
    <tr height="10">
      <td>
        <table>
          <tr>
            <td class="menublock" >
              
              <a href="<?php echo $main["liens"]["sql"] ?>" ?> Requête
                SQL
              </a>
            </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</body>
</html>
```



```

        </tr>
      </table>
    </td>
    <td>
      
    </td>
    <td>
      <fieldset>
        <legend><?php echo $main["légende"] ?></legend>
        <?php
          include $main["contenu"];
        ?>
      </fieldset>
    </td>
  </tr>
</table>
</body>
</html>

```

Les zones paramétrées de la page ont été mises en relief dans le listing ci-dessus. La page type est paramétrée de plusieurs façons :

- par un dictionnaire **\$main** ayant les clés suivantes :
 - **title** : titre à mettre en zone 1 de la page
 - **liens** : dictionnaires des liens à générer dans la colonne du menu. Ces liens sont associés aux options du menu de la zone 2
 - **contenu** : url de la page à afficher dans la zone 4
- par un dictionnaire **\$dConfig** rassemblant des informations tirées d'un fichier de configuration de l'application appelé **config.php**
- par des classes faisant partie de la feuille de style utilisée par la page :

```
<link type="text/css" href="<?php echo $dConfig['urlPageStyle'] ?>" rel="stylesheet" />
```

La page utilise ici les classes de style suivantes :

- **menutitle** : pour une option principale du menu
- **menublock** : pour une option secondaire du menu

Changer l'un des paramètres change l'aspect de la page. Ainsi changer **\$main['title']** changera le titre de la zone 1.

Le traitement type d'une demande d'un client

Le client interagit avec l'application grâce aux liens de la zone 2 de la page type. Ces liens seront du type suivant :

```
apparticles.php?action=xx&phase=y&PHPSESSID=zzzzzzzzzzzz
```

action désigne l'action en cours parmi les suivantes :

authentifier	authentification du client
selectArticles	sélection d'articles (consultation)
updateArticle	modification d'un article
deleteArticle	suppression d'un article
sql	émission d'une requête SQL quelconque (administrateur)

phase une action peut se faire en plusieurs étapes - désigne l'étape en cours

PHPSESSID jeton de session lorsque celle-ci a démarré - permet au serveur de récupérer des informations stockées dans la session lors des précédents échanges

De même l'attribut **action** dans les formulaires aura la même forme. Par exemple, dans la page d'accueil il y a un formulaire de login dans la zone 4. La balise HTML de ce formulaire est définie comme suit :

```
<form name="frmLogin" method="post" action="apparticles.php?action=authentifier&phase=1">
```

Le traitement de la demande du client est accomplie par le script principal de l'application appelé **apparticles.php**. Son travail est de construire la réponse au client. Il procèdera toujours de la même façon :

1. grâce au nom de l'action et à la phase en cours, il **délèguera** la demande à une fonction spécialisée. Celle-ci traitera la demande et générera la page réponse adéquate. Pour chaque demande du client, il peut y avoir plusieurs pages réponse possibles : **page1**,

page2, ..., pagen. Ces pages contiennent des informations qui doivent être calculées par la fonction. Ce sont donc des pages paramétrées. Elles seront générées par des scripts **page1.php, page2.php, ..., pagen.php**.

2. par souci d'homogénéité, les parties variables des pages à afficher dans la zone 4 de la page type seront elles-aussi placées dans le dictionnaire **\$main**.

Supposons qu'en réponse à une demande, le serveur doit envoyer la page **pagex.php** au client. Il procèdera de la façon suivante :

- il placera dans le dictionnaire **\$main** les valeurs nécessaires à la page **pagex.php**
- il mettra dans **\$main['contenu']** qui désigne l'URL de la page à afficher en zone 4 de la page type, l'URL de **pagex.php**
- il demandera l'affichage de la page type avec l'instruction

```
include "main.php";
```

La page type sera alors affichée avec dans la zone 4 le code du script **pagex.php** qui sera évalué pour générer le contenu de la zone 4. On se rappellera que celle-ci est une simple cellule d'un tableau. Il ne faut donc pas que le code HTML généré par **pagex.php** commence par les balises <HTML>, <HEAD>, <BODY>, Celles-ci ont déjà été émises au début de la page type. Voici par exemple à quoi pourrait ressembler le script **login.php** qui génère la zone 4 de la page d'accueil :

```
<form name="frmLogin" method="post" action="<?php echo $main["post"] ?>">
  <table>
    <tr>
      <td>login</td>
      <td><input type="text" value="<?php echo $main["login"] ?>" name="txtLogin" class="text"></td>
    </tr>
    <tr>
      <td>mot de passe</td>
      <td><input type="password" value="" name="txtMdp" class="text"></td>
      <td><input type="submit" value="Connexion" class="submit"></td>
    </tr>
  </table>
</form>
```

On voit que la page :

1. est réduite à un formulaire
2. est paramétrée à la fois par le dictionnaire **\$main** et la feuille de style.

Le fichier de configuration

On a toujours intérêt à paramétrer le plus possible les applications afin d'éviter d'aller dans le code simplement parce qu'on a décidé par exemple de changer le chemin d'un script ou d'une image. L'application principale **apparticles.php** chargera donc un fichier de configuration **config.php** à son démarrage :

```
// chargement du fichier de configuration
include "config.php";
```

On mettra dans ce fichier, des directives de configuration destinées à PHP et des initialisations de variables globales :

```
<?php

// configuration de php ini_set("register_globals","off");
ini_set("display_errors","off"); ini_set("expose_php","off");
ini_set("session.use_cookies","0"); // pas de cookies

// configuration base articles
$dConfig["DSN"]=array(
    "sgbd"=>"mysql",
    "admin"=>"admarticles",
    "mdpadmin"=>"mdparticles",
    "host"=>"localhost",
    "database"=>"dbarticles"
);

// url des pages
$dConfig['urlBackGround']='../images/standard.jpg';
$dConfig['urlPageStyle']='mystyle.css';
$dConfig['urlAppArticles']='apparticles.php';
$dConfig['urlPageMain']='main.php';
$dConfig['urlPageLogin']='login.php';
$dConfig['urlPageErreurs']='erreurs.php';
$dConfig['urlPageInfos']='infos.php';
```

```

$DConfig["urlPageAddArticle"]="addarticle.php";
$DConfig["urlPageUpdateArticle1"]="updatearticle1.php";
$DConfig["urlPageUpdateArticle2"]="updatearticle2.php";
$DConfig["urlPageDeleteArticle1"]="deletearticle1.php";
$DConfig["urlPageDeleteArticle2"]="deletearticle2.php";
$DConfig["urlPageSelectArticle1"]="selectarticle1.php";
$DConfig["urlPageSelectArticle2"]="selectarticle2.php";
$DConfig["urlPageSQL1"]="sql1.php";
$DConfig["urlPageSQL2"]="sql2.php";
$DConfig["urlPageSQL3"]="sql3.php";

// liens de la page principale
$main["liens"]["login"]="$UrlAppArticles?action=authentifier&phase=0";
$main["liens"]["addArticle"]="$UrlAppArticles?action=addArticle&phase=0";
$main["liens"]["updateArticle"]="$UrlAppArticles?action=updateArticle&phase=0";
$main["liens"]["deleteArticle"]="$UrlAppArticles?action=deleteArticle&phase=0";
$main["liens"]["selectArticle"]="$UrlAppArticles?action=selectArticle&phase=0";
$main["liens"]["sql"]="$UrlAppArticles?action=sql&phase=0";

// on mémorise $main dans la configuration
$DConfig["main"]=$main;
?>

```

La feuille de style associée à la pagetype

Nous avons vu que la réponse du serveur avait un format unique celui de **main.php**. On aura pu remarquer que ce script produit une page brute dépourvue d'effets de présentation. C'est une bonne chose pour plusieurs raisons :

- le développeur n'a pas à se soucier de la présentation graphique de la page qu'il crée. Il n'a en effet pas nécessairement les compétences pour créer des pages graphiques attractives. Il peut ici se concentrer entièrement sur le code.
- la maintenance des scripts est facilitée. Si ceux-ci comportaient des attributs de présentation, ni la structure du code, ni celle de la présentation n'apparaîtraient clairement. L'aspect graphique des feuilles est souvent délégué à un graphiste. Celui-ci n'aimerait probablement pas à devoir chercher dans un script qu'il ne comprend pas, où sont les attributs de présentation qu'il doit modifier.

Il faut pourtant bien se soucier de l'aspect graphique des pages. En effet, c'est cela qui attire les internautes vers un site. Ici, la présentation est déléguée à une feuille de style. La page **main.php** indique dans son code la feuille de style à utiliser pour l'afficher :

```

<head>
  <title>Gestion d'articles</title>
  <link type="text/css" href="<?php echo $DConfig['urlPageStyle'] ?>" rel="stylesheet" />
</head>

```

La feuille de style utilisée dans ce document est la suivante :

```

BODY {
  background : url(../images/standard.jpg); border : 2px
  none #FFDAB9;
  font-family : Garamond;
  font-size : 16px; margin-
  left : 0px; padding-left :
  20px;
}

INPUT {
  background : #EEE8AA; border :
  1px solid #EE82EE; font-family :
  Garamond; font-size : 18px;
}

INPUT.submit{
  font-family : "Times New Roman"; font-
  size : 16px;
  background : #FA8072; border :
  2px double Green; font-weight :
  bold;
  text-align : center;
  vertical-align : middle;
  cursor : pointer;
}

TD.menuitle{
  background-image : url(../images/menugelgd.gif);

```

```

height : 23px;
text-align : center;
vertical-align : middle;
background : url(../images/menugelgd.gif) no-repeat center;
}

TD.menublock{
background : url(../images/bandeग्रismenugd.gif) repeat-x; text-align
: left;
vertical-align : middle;
}

A {
font-family : "Comic Sans MS"; color
: #FF7F50;
font-size : 15px;
text-decoration : none;
}

A:HOVER {
background : #FFA07A;
color : Red;
}

FIELDSET {
border : 1px solid #A0522D;
background : #FFE4C4;
margin : 10px 10px 10px 10px;
padding-left : 10px;
padding-right : 10px;
padding-bottom : 10px;
}

LEGEND{
background : #FFA500;
}

TH {
background : #228B22; text-
align : center; vertical-
align : middle;
}

TD.libellé{
border : 1px solid #008B8B;
color : #339966;
}

H1 {
font : bold 20px/30px Garamond; color :
#FF7F50;
background : #D1E1F8; background-
attachment : fixed; text-align :
center;
vertical-align : middle;
font-family : Garamond;
}

SELECT.TEXT {
background : #6495ED;
text-align : center;
color : Aqua;
}

```

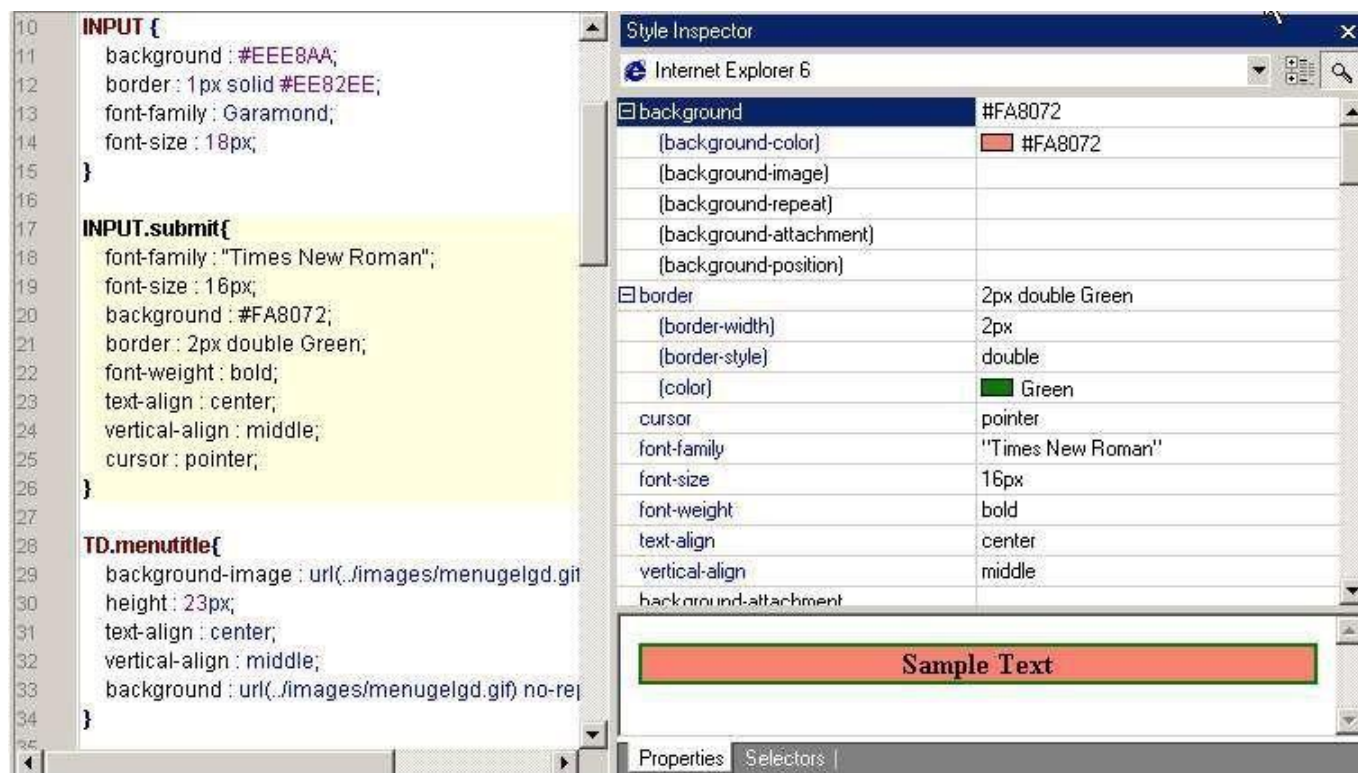
Nous n'entrerons pas dans les détails de cette feuille de style. Nous l'accepterons telle-quelle. Un peu plus loin nous verrons comment la construire et la modifier. Il existe des logiciels pour cela. Néanmoins indiquons le rôle des attributs de présentation utilisés dans la feuille :

Attribut :	régit la présentation de la balise HTML :
BODY	<BODY>
H1	<H1> (Header1)
A	<A> (Anchor)
A:HOVER	fixe les attributs de présentation de l'ancree lorsque l'utilisateur passe la souris dessus
FIELDSET	<FIELDSET> - cette balise n'est pas reconnue par tous les navigateurs
LEGEND	<LEGEND> - cette balise n'est pas reconnue par tous les navigateurs
INPUT	<INPUT>
INPUT.TEXT	<INPUT class="TEXT">
INPUT.SUBMIT	<INPUT class="SUBMIT">
TH	<TH> (Table Header)
TD.menutitle	<TD class="menutitle"> (Table Data)

TD. menublock <TD class="menublock">
 TD. libellé <TD class="libellé">

Voyons sur un exemple comment peuvent être écrites ces règles de présentation. Dans cet exemple, nous utiliserons le logiciel TopStyle Lite disponible gratuitement à l'URL <http://www.bradsoft.com>. Une fois chargée la feuille de style, on a une fenêtre à trois zones :

- une zone d'édition de texte. Les attributs de présentation peuvent être définis à la main à condition de connaître les règles d'écriture des feuilles de style qui suivent une norme appelée CSS (Cascading Style Sheets).
- la zone 2 présente les propriétés éditables de l'attribut en cours de construction. C'est la méthode la plus simple. Elle évite d'avoir à connaître le nom exact des attributs de présentation qui sont très nombreux
- la zone 3 donne l'aspect visuel de l'attribut en cours de construction

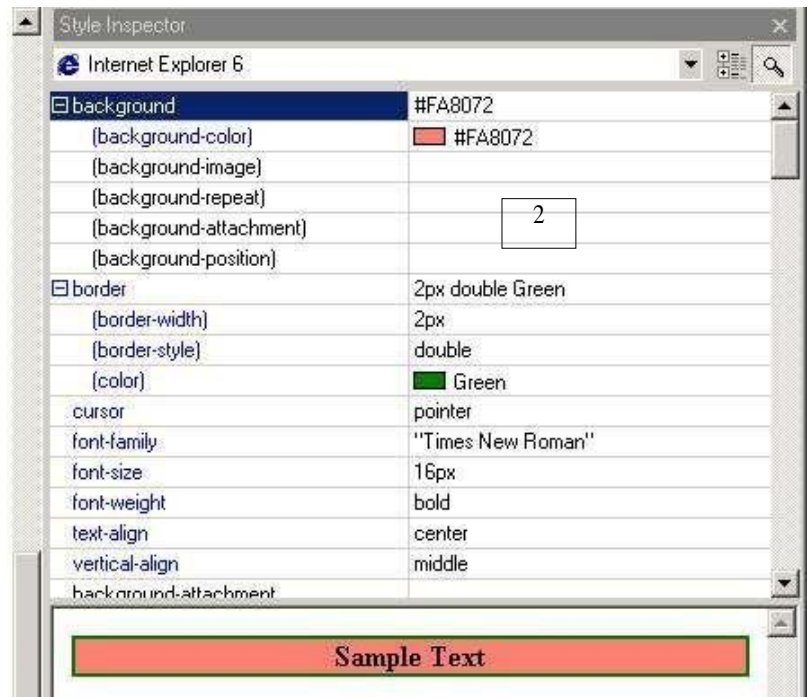


Dans la zone 1 ci-dessus, opérons un copier-coller de l'attribut **INPUT.submit** vers un attribut **INPUT.fantaisie**. Cet attribut fixera la présentation de la balise HTML **<INPUT class="fantaisie">**


```

81 background : #D1E1F8;
82 background-attachment : fixed;
83 text-align : center;
84 vertical-align : middle;
85 font-family : Garamond;
86 }
87
88
89 SELECT.TEXT {
90 background : #6495ED;
91 text-align : center;
92 color : Aqua;
93 }
94
95 INPUT.fantaisie{
96 font-family : "Times New Roman";
97 font-size : 16px;
98 background : #FA8072;
99 border : 2px double Green;
100 font-weight : bold;
101 text-align : center;
102 vertical-align : middle;
103 cursor : pointer;
104 }

```

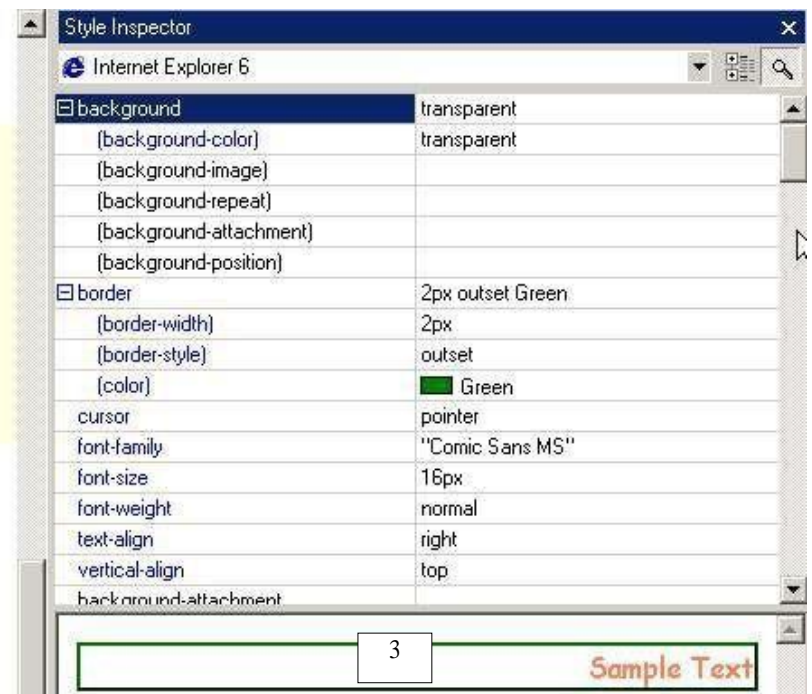


Utilisons la zone 2 pour modifier certaines des propriétés de l'attribut **INPUT.fantaisie** :

```

91 text-align : center;
92 color : Aqua;
93 }
94
95 INPUT.fantaisie{
96 border : 2px outset Green;
97 font-family : "Comic Sans MS";
98 font-size : 16px;
99 background : transparent;
100 font-weight : normal;
101 text-align : right;
102 vertical-align : top;
103 cursor : pointer;
104 color : #FF7F50;
105 }

```



Désormais, toute balise `<INPUT ... class="fantaisie">` trouvée dans une page HTML associée à la feuille de style précédente sera présentée comme l'exemple de la zone 3 ci-dessus.

L'intérêt des feuilles de style est grand. Leur utilisation permet de changer le "look" d'une application web en ne modifiant celle-ci qu'en un seul point : sa feuille de style. Les feuilles de style ne sont pas reconnues par les navigateurs anciens. La directive `<link ...>` ci-dessous sera ignorée par certains d'entre-eux :

```

<head>
<title>Gestion d'articles</title>
<link type="text/css" href="<?php echo $dConfig['urlPageStyle'] ?>" rel="stylesheet" />
</head>

```

Dans notre application, cela donnera la page d'accueil suivante :

Gestion d'articles

Authentification

Utilisation

- [Ajouter un article](#)
- [Modifier un article](#)
- [Supprimer un article](#)
- [Lister des articles](#)

Connexion à la base

login	<input type="text"/>	<input type="button" value="Connexion"/>
mot de passe	<input type="password"/>	

Administration

- [Requête SQL](#)

On a là une page minimale sans recherche graphique. On peut avoir pire. Certaines versions de navigateurs reconnaissent les feuilles de style mais les interprètent mal. On peut alors avoir une page défigurée et inutilisable. Se pose donc la question du type du navigateur client. Il existe des techniques qui aident à déterminer le type du navigateur client. Elles ne sont pas totalement fiables. On peut alors écrire différentes feuilles de style pour différents navigateurs voire écrire une version sans feuille de style pour les navigateurs qui les ignorent. Cela alourdit bien sûr la tâche de développement. Ce problème important a été ici ignoré.

Avec les feuilles de style, on peut imaginer offrir un environnement personnalisé aux utilisateurs de notre application. Nous pourrions leur présenter une page leur offrant plusieurs styles de présentation possibles. Ils pourraient choisir celui qui leur convient le mieux. Ce choix pourrait être enregistré dans une base de données. Lorsque l'utilisateur se connecte de nouveau, on pourrait alors démarrer l'application avec la feuille de style qui a reçu sa préférence.

Le module d'entrée de l'application

Les clients ne connaîtront de l'application que son module d'entrée : **apparticles.php**. Les grandes lignes de son fonctionnement sont les suivantes :

- La demande du client est récupérée et analysée. Celle-ci est paramétrée ou non. Lorsqu'elle est paramétrée, les paramètres attendus sont les suivants : **action=[action]&phase=[phase]&PHPSESSID=[PHPSESSID]**
- Si la demande n'est pas paramétrée ou si les paramètres récupérés ne sont pas ceux attendus, le serveur envoie comme réponse la page d'authentification (login, mot de passe). Dès que l'utilisateur se sera identifié correctement, une session est créée. Elle servira à stocker des informations tout au long des échanges client-serveur.
- Si une demande est correctement reconnue, elle est traitée par un module qui dépend de l'action et de la phase en cours.
- Tous les accès à la base de données se font par l'intermédiaire de la classe métier **articles.php**.
- Le traitement d'une demande se termine toujours par l'envoi au client de la page **main.php** dans laquelle on a précisé dans **\$main['contenu']** l'URL de la page à mettre dans la zone 4 de la page type.

Le squelette du script **apparticles.php** pourrait être le suivant :

```
<?php
// gestion d'une table d'articles include
"config.php";
include "articles.php";

// action à entreprendre
$action=$_POST["action"] ? $_POST["action"] : $_GET["action"] ? $_GET["action"] : "authentifier";
$action=strtolower($action);
// phase éventuelle
$phase=$_POST["phase"] ? $_POST["phase"] : $_GET["phase"] ? $_GET["phase"] : "0";

// session
```

```

session_start();
$dSession=$_SESSION["session"];

// y-a-t-il une session en cours ? if(!
isset($dSession)){
    // authentification de l'utilisateur
    if($sAction=="authentifier" && $sPhase=="0") authentifieur_0($dConfig);
    if($sAction=="authentifier" && $sPhase=="1") authentifieur_1($dConfig);
    if($sAction=="authentifier" && $sPhase=="2") authentifieur_2($dConfig);
    // demande incorrecte
    authentifieur_0($dConfig);
} //if - pas de session

// on récupère la session
$dSession=unserialize($dSession);

// traitement de la demande
// ----- authentification
if($sAction=="authentifier" && $sPhase=="0") authentifieur_0($dConfig);
if($sAction=="authentifier" && $sPhase=="1") authentifieur_1($dConfig);
if($sAction=="authentifier" && $sPhase=="2") authentifieur_2($dConfig);
// ----- ajout d'article
if($sAction=="addarticle" && $sPhase=="0") addArticle_0($dConfig, $dSession);
if($sAction=="addarticle" && $sPhase=="1") addArticle_1($dConfig, $dSession);
if($sAction=="addarticle" && $sPhase=="2") addArticle_2($dConfig, $dSession);
// ----- mise à jour d'article
if($sAction=="updatearticle" && $sPhase=="0") updateArticle_0($dConfig, $dSession);
if($sAction=="updatearticle" && $sPhase=="1") updateArticle_1($dConfig, $dSession);
if($sAction=="updatearticle" && $sPhase=="2") updateArticle_2($dConfig, $dSession);
if($sAction=="updatearticle" && $sPhase=="3") updateArticle_3($dConfig, $dSession);
// ----- suppression d'article
if($sAction=="deletearticle" && $sPhase=="0") deleteArticle_0($dConfig, $dSession);
if($sAction=="deletearticle" && $sPhase=="1") deleteArticle_1($dConfig, $dSession);
if($sAction=="deletearticle" && $sPhase=="2") deleteArticle_2($dConfig, $dSession);
// ----- consultation d'articles
if($sAction=="selectarticle" && $sPhase=="0") selectArticle_0($dConfig, $dSession);
if($sAction=="selectarticle" && $sPhase=="1") selectArticle_1($dConfig, $dSession);
if($sAction=="selectarticle" && $sPhase=="2") selectArticle_2($dConfig, $dSession);
// ----- émission d'une requête SQL
if($sAction=="sql" && $sPhase=="0") sql_0($dConfig, $dSession);
if($sAction=="sql" && $sPhase=="1") sql_1($dConfig, $dSession);
if($sAction=="sql" && $sPhase=="2") sql_2($dConfig, $dSession);

// action erronée - on présente la page d'authentification session_destroy();
authentifieur_0($dConfig, "0");
...
?>

```

On notera les points suivants :

- les fonctions traitant une demande particulière du client se terminent par la génération de la page réponse et par une instruction **exit** qui termine l'exécution du script **apparticles.php**. Autrement dit, on ne "revient" pas de ces fonctions.
- les fonctions admettent un ou deux paramètres :
 - **\$dConfig** est un dictionnaire contenant des informations issues du fichier de configuration **config.php**. Toutes les fonctions l'utilisent.
 - **\$dSession** est un dictionnaire contenant des informations de session. Il n'existe que lorsque la session a été créée c'est à dire après que l'authentification de l'utilisateur a réussi. C'est pourquoi les fonctions **authentifier** n'ont pas ce paramètre.

La page d'erreurs

Toute application logicielle doit savoir gérer correctement les erreurs qui peuvent survenir. Une application web n'échappe pas à cette règle. Ici, lors d'une erreur, nous placerons la page **erreurs.php** suivante dans la zone 4 de la page type :

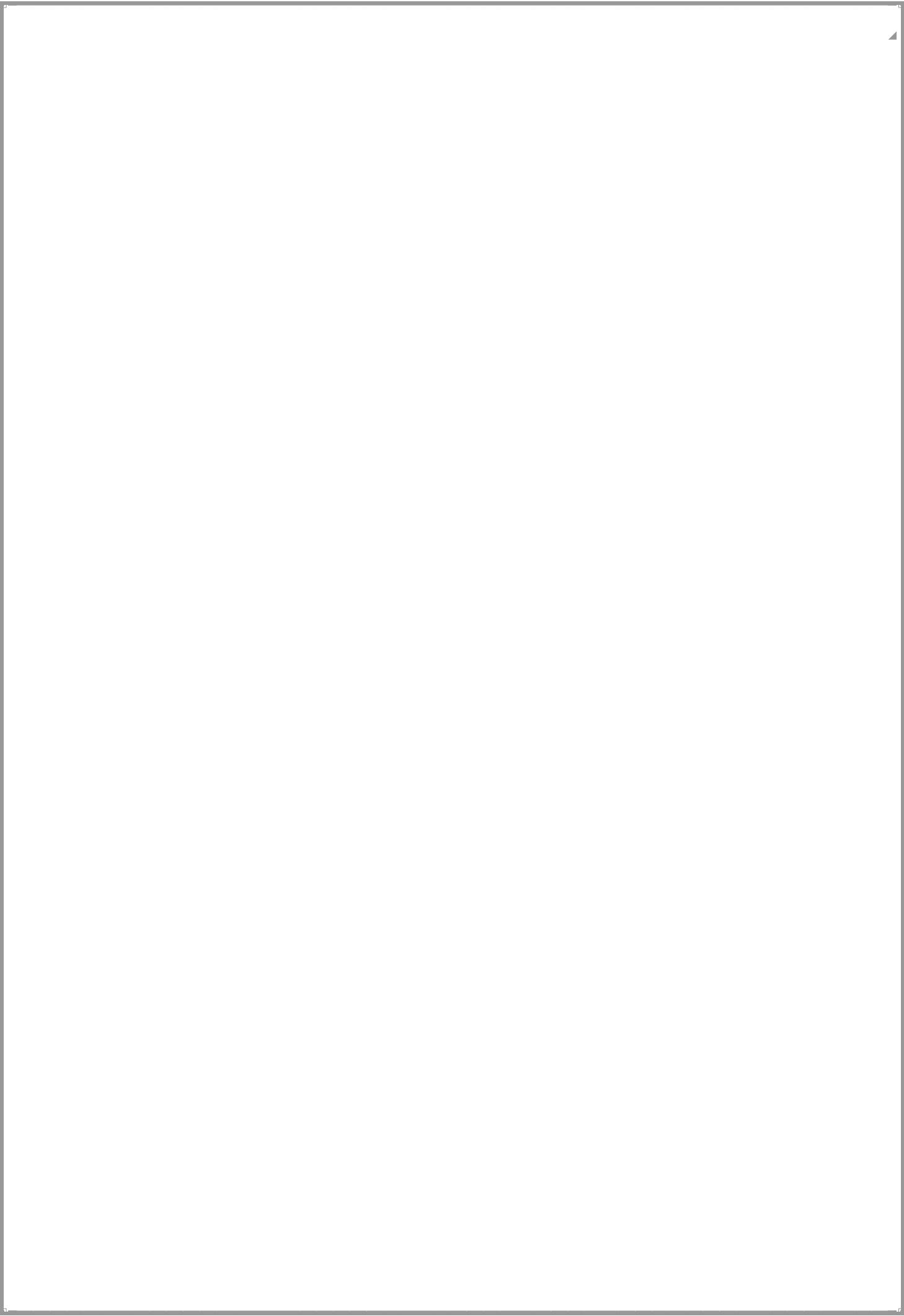
Les erreurs suivantes se sont produites :

```

<ul>
<?php
    for($i=0;$i<count($main["erreurs"]);$i++){ echo
        "<li>". $main["erreurs"][$i]. "</li>¥n";
    } //for
?>
</ul>
<a href="<?php echo $main["href"] ?>"><?php echo $main["lien"] ?></a>

```

Elle présente la liste d'erreurs définie dans **\$main['erreurs']**. Par ailleurs, elle peut proposer un lien de retour, en général vers la page qui a précédé la page d'erreurs. Ce lien sera défini par un libellé **\$main['lien']** et une URL **\$main['href']**. Pour ne pas avoir



ce lien, il suffira de mettre la chaîne vide dans `$main['lien']`. Voici un exemple de page d'erreurs dans le cas où l'utilisateur s'identifie incorrectement :



La page d'informations

Parfois on voudra donner en réponse à l'utilisateur une simple information, par exemple que son identification a réussi. On utilisera pour cela la page **infos.php** suivante :

```
<?php echo $main["infos"] ?>
```

Pour afficher une information en retour à une demande d'un client, on

1. mettra l'information dans `$main['infos']`
2. mettra l'URL de **infos.php** dans `$main['contenu']`

Voici par exemple, l'information retournée lorsque l'utilisateur s'est identifié correctement :



Le fonctionnement de l'application

Nous avons maintenant une bonne idée de la structure générale de l'application à écrire. Il nous reste à présenter les cheminements de l'utilisateur dans l'application, les actions qu'il peut faire et les réponses qu'il reçoit du serveur. Ceci fait, nous pourrions écrire les fonctions qui traitent les différentes demandes d'un client. Dans ce qui suit, nous allons présenter le fonctionnement de l'application au travers des pages présentées à l'utilisateur en réponse à certaines de ces actions. Nous précisons à chaque fois les points suivants :

action utilisateur action initiale de l'utilisateur qui a amené à la réponse affichée

paramètres envoyés les paramètres envoyés par le navigateur client au serveur en réponse à l'action manuelle de l'utilisateur

page réponse le script qui génère la zone 4 de la page type

L'authentification

Avant de pouvoir utiliser l'application, l'utilisateur devra s'identifier à l'aide de la page suivante :

Gestion d'articles

Authentification

Utilisation

- Ajouter un article
- Modifier un article
- Supprimer un article
- Lister des articles

Connexion à la base

login

mot de passe

Connexion

Administration

- Requête SQL

action utilisateur 1 - demande initiale de l'URL apparticles.php
2 - utilisation de l'option Authentification du menu
3 - demande directe de l'URL articles.php avec des paramètres erronés

paramètres envoyés 1 - pas de paramètres
2 - action=authentifier?phase=0
3 - une liste de paramètres erronés

page réponse login.php

Sur la page d'accueil, le lien [Ajouter un article] est de la forme suivante : **action=addarticle?phase=0**. Les autres liens sont de la même forme avec action=(authentifier, updatearticle, deletearticle, selectarticle, sql). L'utilisateur remplit le formulaire et utilise le bouton [Connexion] :

Connexion à la base

login

user1

mot de passe *****

Connexion

La réponse est la suivante :

Gestion d'articles (user1 - utilisateur)

Authentification

Utilisation

- Ajouter un article
- Modifier un article
- Supprimer un article
- Lister des articles

Administration

- Requête SQL

Connexion réussie

Vous êtes connecté à la base des articles.

action utilisateur	bouton [Connexion]
paramètres envoyés	action=authentifier?phase=1
page réponse	infos.php

Le titre de la page a été modifié pour indiquer le login de l'utilisateur et ses droits administrateur/utilisateur. Par ailleurs, tous les liens de la zone 2 ont été modifiés pour refléter le fait qu'une session a démarré. Le paramètre PHPSESSID=[PHPSESSID] leur a été ajouté.

Si le serveur n'a pas pu identifier le client, celui-ci aura une réponse différente :

Gestion d'articles

Authentification

Utilisation

- Ajouter un article
- Modifier un article
- Supprimer un article
- Lister des articles

Administration

- Requête SQL

Authentification : erreurs

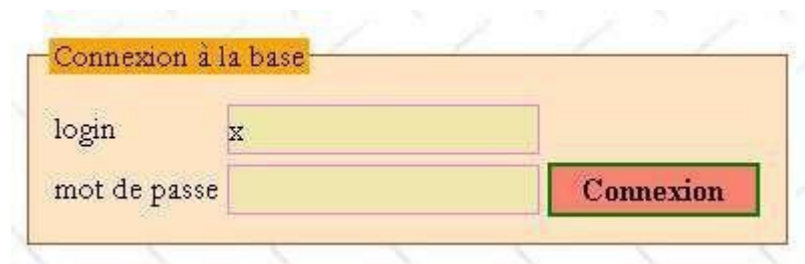
Les erreurs suivantes se sont produites :

- ♦ L'utilisateur [x] n'a pas été reconnu

[Retour à la page de login](#)

action utilisateur	bouton [Connexion]
paramètres envoyés	action=authentifier?phase=1
page réponse	erreurs.php

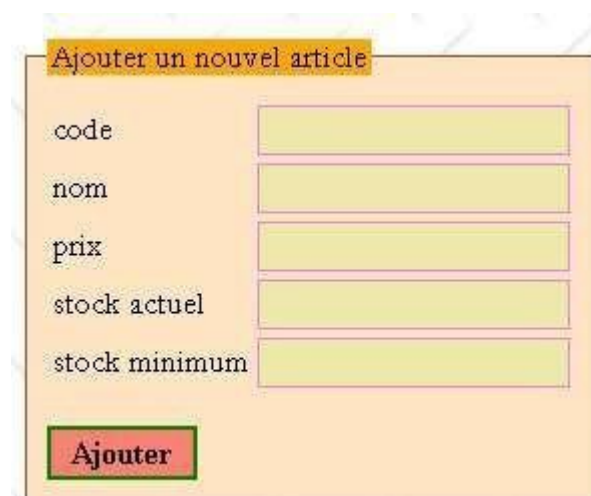
Le lien [Retour à la page de login] est un lien sur l'URL **apparticles.php?action=authentifier&phase=2&txtLogin=x**. Ce lien ramène le client à la page de login où le champ de login est rempli par la valeur du paramètre **txtLogin** :



action utilisateur	lien [Retour à la page de login]
paramètres envoyés	action=authentifier?phase=2&txtLogin=x
page réponse	login.php

Ajout d'un article

Le lien du menu [Ajouter un article] amène la page suivante dans la zone 4 de la page type :



action utilisateur	lien [Ajouter un article]
paramètres envoyés	action=addArticle?phase=0&PHPSESSID=[PHPSESSID]
page réponse	addarticle.php

L'utilisateur remplit les champs et envoie le tout au serveur avec le bouton [Ajouter] qui est de type *submit*. Aucune vérification n'est faite côté client. C'est le serveur qui les fait. Il peut envoyer en réponse une page d'erreurs comme sur l'exemple ci-dessous :

Demande	Réponse
---------	---------

Ajouter un nouvel article

code	xxxxx
nom	x
prix	x
stock actuel	x
stock minimum	x

Ajouter

Ajouter un nouvel article : erreurs

Les erreurs suivantes se sont produites :

- ◆ Prix x invalide
- ◆ Stock actuel x invalide
- ◆ Stock minimum x invalide

[Retour à la page d'ajout d'article](#)

action utilisateur	bouton [Ajouter]
paramètres envoyés	action=addArticle?phase=1&PHPSESSID=[PHPSESSID]
page réponse	erreurs.php

Le lien [Retour à la page d'ajout d'article] permet de revenir à la page de saisie :

Demande

Réponse

Ajouter un nouvel article : erreurs

Les erreurs suivantes se sont produites :

- ◆ Prix x invalide
- ◆ Stock actuel x invalide
- ◆ Stock minimum x invalide

[Retour à la page d'ajout d'article](#)

Ajouter un nouvel article

code	xxxxx
nom	x
prix	x
stock actuel	x
stock minimum	x

Ajouter

action utilisateur	lien [Retour à la page d'ajout d'article]
paramètres envoyés	action=addArticle?phase=2&PHPSESSID=[PHPSESSID]
page réponse	article.php

Si l'ajout se fait sans erreurs, l'utilisateur reçoit un message de confirmation :

Demande

Réponse

Ajouter un nouvel article

code x100
nom calculatrice
prix 100
stock actuel 11
stock minimum 2

Ajouter

Ajouter un nouvel article

Ajout réussi

action utilisateur	bouton [Ajouter]
paramètres envoyés	action=addArticle?phase=1&PHPSESSID=[PHPSESSID]
page réponse	infos.php

Consultation d'articles

Le lien du menu [Lister des articles] amène la page suivante dans la zone 4 de la page type :

Sélectionner des articles

Définissez les caractéristiques de la requête

colonnes (col1, col2, ...)
where
order by

Afficher

action utilisateur	lien du menu [Lister des articles]
paramètres envoyés	action=selectArticle?phase=0&PHPSESSID=[PHPSESSID]
page réponse	select1.php

Une requête **select [colonnes] from articles where [where] orderby [orderby]** sera émise sur la table des articles où [colonnes], [where] et [orderby] sont le contenu des champs ci-dessus. Par exemple :

Demande

Sélectionner des articles

Définissez les caractéristiques de la requête

colonnes (col1, col2, ...) *

where

order by

prix desc

Afficher

Réponse

select * from articles order by prix desc

code	nom	prix	Stockactuel	Stock minimum
x100	calculatrice	100	11	2
a100	cartable	20	5	3
e100	stylo plume	20	40	17
d100	feuilles blanches - lot de 500	10	32	6
b100	trousse	5	20	3
c100	lot de 6 crayons bille	4	30	10

[Retour à la page de sélection d'articles](#)

action utilisateur
paramètres envoyés

bouton [Afficher]
action=selectArticle?phase=1&PHPSESSID=[PHPSESSID]

page réponse

select2.php

La demande peut être erronée auquel cas le client reçoit une page d'erreurs :

Demande

Sélectionner des articles

Définissez les caractéristiques de la requête

colonnes (col1, col2, ...) x

where

order by

prix desc

Afficher

Réponse

Sélectionner des articles : erreurs

Les erreurs suivantes se sont produites :

- ♦ Echec de la requête [select x from articles order by prix desc] : [DB Error: no such field]

[Retour à la page de sélection d'articles](#)

Dans les deux cas (erreurs ou pas), le lien [Retour à la page de sélection d'articles] permet de revenir à la page **select1.php** :

Demande

Sélectionner des articles : erreurs

Les erreurs suivantes se sont produites :

- ♦ Echec de la requête [select x from arti

[Retour à la page de sélection d'articles](#)

Réponse

Sélectionner des articles

Définissez les caractéristiques de la requête :

colonnes (col1, col2, ...) x

where

order by

prix desc

Afficher

action utilisateur
paramètres envoyés
page réponse

lien [Retour à la page de sélection d'articles]
action=selectArticle?phase=2&PHPSESSID=[PHPSESSID]
select1.php

Modification d'articles

Le lien du menu [Modifier un article] amène la page suivante dans la zone 4 de la page type :

Modifier un article existant

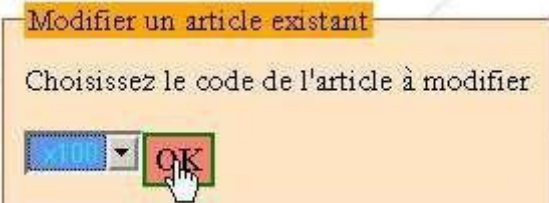
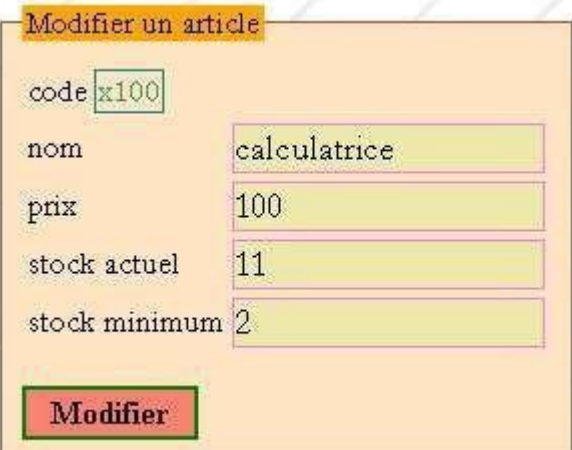
Choisissez le code de l'article à modifier

100

OK

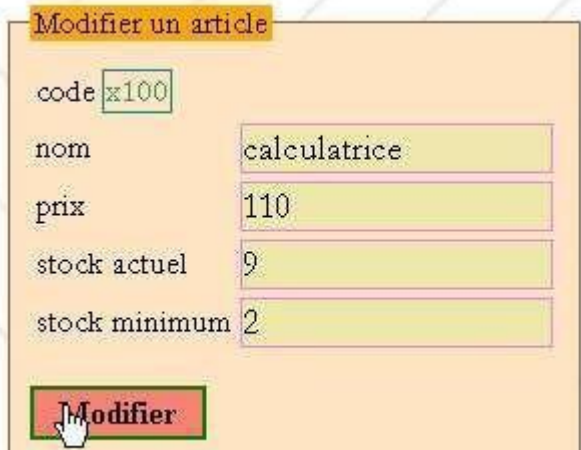
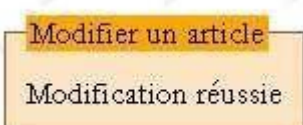
action utilisateur	lien de menu [Modifier un article]
paramètres envoyés	action=updateArticle?phase=0&PHPSESSID=[PHPSESSID]
page réponse	updatearticle1.php

On choisit le code de l'article à modifier dans la liste déroulante et on fait [OK] pour modifier l'article ayant ce code :

Demande	Réponse
	

action utilisateur	bouton [OK]
paramètres envoyés	action=updateArticle?phase=1&PHPSESSID=[PHPSESSID]
page réponse	updatearticle2.php

Une fois obtenue la fiche de l'article à modifier, l'utilisateur peut faire ses modifications :

Demande	Réponse
	

action utilisateur	bouton [Modifier]
paramètres envoyés	action=updateArticle?phase=2&PHPSESSID=[PHPSESSID]
page réponse	infos.php

L'utilisateur peut faire des erreurs lors de la modification :

Demande	Réponse
---------	---------

Modifier un article

code

nom

prix

stock actuel

stock minimum

Modifier un article : erreurs

Les erreurs suivantes se sont produites :

- ◆ Stock actuel 9x invalide

[Retour à la page de modification d'article](#)

Le lien [Retour à la page de modification d'article] permet de revenir à la page de saisie :

Modifier un article

code

nom

prix

stock actuel

stock minimum

action utilisateur
paramètres envoyés
page réponse

lien [Retour à la page de modification d'article]
action=updateArticle?phase=3&PHPSESSID=[PHPSESSID]
updatearticle2.php

Suppression d'un article

Le lien du menu [Supprimer un article] amène la page suivante dans la zone 4 de la page type :

Supprimer un article

Choisissez le code de l'article à supprimer

action utilisateur
paramètres envoyés
page réponse

lien du menu [Supprimer un article]
action=deleteArticle?phase=0&PHPSESSID=[PHPSESSID]
deletearticle1.php

L'utilisateur choisit le code de l'article à supprimer dans une liste déroulante :

Demande	Réponse
<p>Supprimer un article</p> <p>Choisissez le code de l'article à supprimer</p> <p> <input type="text" value="x100"/> <input type="button" value="OK"/> </p>	<p>Supprimer un article</p> <p>code <input type="text" value="x100"/></p> <p>nom <input type="text" value="calculatrice"/></p> <p>prix <input type="text" value="110"/></p> <p>stock actuel <input type="text" value="9"/></p> <p>stock minimum <input type="text" value="2"/></p> <p><input type="button" value="Supprimer"/></p>

action utilisateur	bouton [OK]
paramètres envoyés	action=deleteArticle?phase=1&PHPSESSID=[PHPSESSID]
page réponse	deletearticle2.php

L'utilisateur confirme la suppression de l'article avec le bouton [Supprimer] :

Demande	Réponse
<p>Supprimer un article</p> <p>code <input type="text" value="x100"/></p> <p>nom <input type="text" value="calculatrice"/></p> <p>prix <input type="text" value="110"/></p> <p>stock actuel <input type="text" value="9"/></p> <p>stock minimum <input type="text" value="2"/></p> <p><input type="button" value="Supprimer"/></p>	<p>Supprimer un article</p> <p>Suppression réussie</p>

action utilisateur	bouton [Supprimer]
paramètres envoyés	action=deleteArticle?phase=2&PHPSESSID=[PHPSESSID]
page réponse	infos.php

Émissions de requêtes administrateur

Le lien du menu [Requête SQL] amène la page suivante dans la zone 4 de la page type :

Exécution d'une requête SQL

Texte de la requête SQL

action utilisateur	lien de menu [Requête SQL]
paramètres envoyés	action=sql?phase=0&PHPSESSID=[PHPSESSID]

page réponse sql1.php

On tape le texte de la requête SQL dans le champ de saisie et on utilise le bouton [Exécuter] pour l'exécuter. Seul un administrateur peut émettre ces requêtes comme le montre l'exemple suivant :

Demande	Réponse
<p>Exécution d'une requête SQL</p> <div><div></div><div>select * from users</div><div>Exécuter</div></div>	<p>SQL : erreurs</p> <p>Les erreurs suivantes se sont produites :</p> <ul style="list-style-type: none">♦ L'utilisateur [user1] n'a pas les droits d'administration <p>Retour à la page d'émission de requêtes SQL</p>

action utilisateur	bouton [Exécuter]
paramètres envoyés	action=sql?phase=1&PHPSESSID=[PHPSESSID]
page réponse	erreurs.php

Le lien [Retour à la page d'émission de requêtes SQL] permet de revenir à la page de saisie :

Exécution d'une requête SQL

select * from users

Exécuter

action utilisateur	lien [Retour à la page d'émission de requêtes SQL]
paramètres envoyés	action=sql?phase=2&PHPSESSID=[PHPSESSID]
page réponse	sql1.php

Si on est administrateur et que la requête est syntaxiquement correcte :

Demande

Exécution d'une requête SQL

Texte de la requête SQL

select * from users,droits where users.login=droits.login

Exécuter

on obtient le résultat de la requête :

Réponse

Exécution d'une requête SQL : résultats

login	mdp	admin	login	table	ajouter	modifier	supprimer	consulter
user1	user1	n	user1	articles	y	y	y	y
user3	user3	n	user3	articles	n	n	n	y

[Retour à la page d'émission de requêtes SQL](#)

action utilisateur bouton [Exécuter]
paramètres envoyés action=sql?phase=1&PHPSESSID=[PHPSESSID]
page réponse sql2.php

On peut émettre des requêtes de mise à jour des tables :

Demande

Exécution d'une requête SQL

Texte de la requête SQL

```
insert into users (login,mdp,admin) values('user4','user4','n')
```

Exécuter

Réponse

Exécution d'une requête SQL : résultats

1 ligne(s) ont été traitées par la commande

[Retour à la page d'émission de requêtes SQL](#)

action utilisateur bouton [Exécuter]
paramètres envoyés action=sql?phase=1&PHPSESSID=[PHPSESSID]
page réponse infos.php

Travail à faire

Écrire les scripts et fonctions nécessaires à l'application :

identifiant	type	rôle
apparticles.php	script	le point d'entrée du traitement des demandes des clients
authentifier_0	fonction	traite la demande paramétrée action=authentifier&phase=0
authentifier_1	fonction	traite la demande paramétrée action=authentifier&phase=1
authentifier_2	fonction	traite la demande paramétrée action=authentifier&phase=2
addarticle_0	fonction	traite la demande paramétrée action=addArticle&phase=0
addarticle_1	fonction	traite la demande paramétrée action=addArticle&phase=1
addarticle_2	fonction	traite la demande paramétrée action=addArticle&phase=2
updatearticle_0	fonction	traite la demande paramétrée action=updatearticle&phase=0
updatearticle_1	fonction	traite la demande paramétrée action=updatearticle&phase=1

updatearticle_2	fonction traite la demande paramétrée action=updatearticle&phase=2
updatearticle_3	fonction traite la demande paramétrée action=updatearticle&phase=3
deletearticle_0	fonction traite la demande paramétrée action=deletearticle&phase=0
deletearticle_1	fonction traite la demande paramétrée action=deletearticle&phase=1
deletearticle_2	fonction traite la demande paramétrée action=deletearticle&phase=2
selectarticle_0	fonction traite la demande paramétrée action=selectarticle&phase=0
selectarticle_1	fonction traite la demande paramétrée action=selectarticle&phase=1
selectarticle_2	fonction traite la demande paramétrée action=selectarticle&phase=2
sql_0	fonction traite la demande paramétrée action=sql&phase=0
sql_1	fonction traite la demande paramétrée action=sql&phase=1
sql_2	fonction traite la demande paramétrée action=sql&phase=2
main.php	script génère la page type
login.php	script génère la page de login
erreurs.php	script génère la page d'erreurs
infos.php	script génère la page d'informations
addarticle.php	script génère la page d'ajout d'un article
updatearticle1.php	script génère la page 1 de la modification d'un article
updatearticle2.php	script génère la page 2 de la modification d'un article
deletearticle1.php	script génère la page 1 de la suppression d'un article
deletearticle2.php	script génère la page 2 de la suppression d'un article
select1.php	script génère la page 1 de la sélection d'articles
select2.php	script génère la page 2 de la sélection d'articles
sql1.php	script génère la page 1 de l'émission de requêtes
	script génère la page 2 de l'émission de requêtes

Faire évoluer l'application

Nous avons à ce point une application qui fait ce qu'elle doit faire avec une ergonomie acceptable. Nous allons la faire évoluer sur différents points :

- le SGBD
- sa sécurité
- son look
- ses performances

Changer le type de la base de données

Notre étude supposait que le SGBD utilisé était MySQL. Changez de SGBD et montrez que la seule modification à faire est dans la définition de la variable `$dDSN` dans le fichier de configuration **config.php**.

Améliorer la sécurité

Lors du développement d'une application web, il ne faut jamais faire l'hypothèse que le client est un navigateur et que la demande qu'il nous envoie est contrôlée par le formulaire qu'on lui a envoyé avant cette demande. N'importe quel programme peut être client d'une application web et donc envoyer n'importe quelle demande paramétrée ou non à l'application. Celle-ci doit donc tout vérifier.

Si on se reporte au code du script **apparticles.php**, on constate

- qu'aucune action autre que l'authentification ne peut avoir lieu sans session. Celle-ci n'existe que si l'utilisateur a réussi à s'authentifier. Rappelons qu'une session est identifiée par une chaîne de caractères assez longue appelée le jeton de session et qui a la forme suivante : **176a43609572907333118333edf6d1fb**. Ce jeton peut être envoyé à l'application de diverses façons, par exemple en utilisant une URL paramétrée :

apparticles.php?PHPSESSID=176a43609572907333118333edf6d1fb.

Un programme qui demanderait, de façon répétée, l'URL précédente en faisant varier le jeton de façon aléatoire dans l'espoir de trouver le bon jeton, a toute chance de mettre de nombreux jours avant de générer la bonne combinaison tellement le nombre des combinaisons possibles est grand. D'ici là, la session étant d'une durée limitée, sera très probablement terminée. Un autre risque serait que le jeton, passant en clair sur le réseau, soit intercepté. Le risque est réel. On peut alors utiliser une connexion cryptée entre le serveur et son client.

- qu'une fois la session lancée, seules certaines actions sont autorisées. Une URL paramétrée par **action=tricher&phase=0&PHPSESSID=[PHPSESSID]** serait rejetée car l'action 'tricher' n'est pas une action autorisée. Lorsque les paramètres (action, phase) ne sont pas reconnus, notre application répond par la page d'authentification.

Cependant l'application ne vérifie pas si les actions autorisées s'enchaînent correctement. Par exemple, les deux actions suivantes :

- 1.action=addArticle&phase=0&PHPSESSID=[PHPSESSID]
- 2.action=updateArticle&phase=1&PHPSESSID=[PHPSESSID]

sont deux actions autorisées. Cependant l'action 2 n'est pas autorisée à suivre l'action 1.

Comment suivre l'enchaînement des URL demandées par le navigateur client ?

On peut s'aider de deux variables PHP : **\$_SERVER['REQUEST_URI']** et **\$_SERVER['HTTP_REFERER']** qui sont deux informations envoyées par les navigateurs clients dans leurs entêtes HTTP.

\$_SERVER['REQUEST_URI'] : C'est l'URI demandée par le client. Par exemple

```
/apparticles.php?action=addArticle&phase=0&PHPSESSID=[PHPSESSID]
```

\$_SERVER['HTTP_REFERER'] : C'est l'URL qui était visualisée dans le navigateur avant la nouvelle URL qu'est en train de demander le navigateur (l'URI précédente). Par exemple, si le navigateur qui a visualisé l'URI évoquée précédemment fait une nouvelle demande à un serveur, la variable **\$_SERVER['HTTP_REFERER']** de celui-ci aura pour valeur

```
http://machine:port//apparticles.php?action=addArticle&phase=0&PHPSESSID=[PHPSESSID]
```

Pour vérifier que deux actions de notre application se suivent dans l'ordre, on peut procéder ainsi :

Lors de l'action 1 :

- on note l'URI demandée (URI1) et on la note dans la session

Lors de l'action 2 :

- on récupère le HTTP-REFERER de l'action 2. On en déduit l'URI (URI2) de l'URL qui était précédemment visualisée dans le navigateur qui fait la demande.
- on récupère l'URI URI1 qui était mémorisée dans la session et qui est l'URI de l'action demandée précédemment au serveur
- Si l'action 2 suit l'action 1, alors on doit avoir URI2=URI1. Si ce n'était pas le cas, on refuserait de faire l'action demandée et on présenterait la page d'authentification.
- on note dans la session l'URI URI2 de l'action en cours pour vérification de l'action suivante. Et ainsi de suite.

Voici un exemple. Après authentification, on choisit le lien [Ajouter un article] :

http://localhost:81/st/php/articles/gestion/articles8/apparticles.php?action=addArticle&phase=0&PHPSESSID=...

Gestion d'articles phpinfo()

Gestion d'articles (user1 - utilisateur)

Authentification

Utilisation

- Ajouter un article
- Modifier un article
- Supprimer un article
- Lister des articles

Administration

- Requête SQL

Ajouter un nouvel article

code

nom

prix

stock actuel

stock minimum

Ajouter

L'URL de cette page est :

http://localhost:81/st/php/articles/gestion/articles8/apparticles.php?action=addArticle&phase=0&PHPSESSID=006a63e6027f16c70b63cdae93405eeb

Directement dans le champ [Adresse] du navigateur, nous modifions l'URL de la façon suivante :

http://localhost:81/st/php/articles/gestion/articles8/apparticles.php?action=deleteArticle&phase=1&PHPSESSID=006a63e6027f16c70b63cdae93405eeb

Nous obtenons alors la page d'authentification :

http://localhost:81/st/php/articles/gestion/articles8/apparticles.php?action=deleteArticle&phase=1&PHPSESSID=...

Gestion d'articles phpinfo()

Gestion d'articles

Authentification

Utilisation

- Ajouter un article
- Modifier un article
- Supprimer un article
- Lister des articles

Connexion à la base

login

mot de passe

Connexion

Cela mérite une explication. Lorsqu'on demande une URL en tapant directement son identité dans le champ adresse du navigateur, celui-ci n'envoie pas l'entête HTTP_REFERER. Notre application n'y trouve donc pas l'URI de l'action précédente, URI qu'elle avait mémorisée dans la session. Elle renvoie alors la page d'authentification en réponse.

Ce mécanisme est efficace pour les navigateurs mais nullement pour un client programmé. Celui-ci peut envoyer l'entête HTTP_REFERER qu'il veut. Il peut donc "tricher" en disant qu'il est bien passé par telle étape alors qu'il ne l'a pas fait. Il faut alors

s'assurer que l'enchaînement des étapes est respecté. Ainsi si l'action demandée est `action=addArticle&phase=1` (saisie) alors l'action précédente doit être forcément `action=deleteArticle&phase=0` (demande initiale de la page de saisie) ou `action=addArticle&phase=2` (retour en saisie après ajout erroné). De même si l'action demandée est `action=addArticle&phase=2` (ajout) alors l'action précédente doit être `action=addArticle&phase=1` (saisie). On peut forcer l'utilisateur à respecter ces enchaînements.

Alors que le premier mécanisme est général et peut s'appliquer à toute application, le second nécessite un codage spécifique à chaque application et est plus lourd : il faut passer en revue toutes les actions possibles de l'utilisateur et leurs enchaînements. On peut mémoriser ces derniers dans un dictionnaire comme le montre le code qui suit :

```
// authentification
$dPrec['authentifier']['0']=array();
$dPrec['authentifier']['1']=array(
    array('action'=>'authentifier','phase'=>'0'),
    array('action'=>'authentifier','phase'=>'2')
);
$dPrec['authentifier']['2']=array(
    array('action'=>'authentifier','phase'=>'1'),
);

// ajout d'article
$dPrec['addarticle']['0']=array();
$dPrec['addarticle']['1']=array(
    array('action'=>'addarticle','phase'=>'0'),
    array('action'=>'addarticle','phase'=>'2')
);
$dPrec['addarticle']['2']=array(
    array('action'=>'addarticle','phase'=>'1'),
);

// modification d'article
$dPrec['updatearticle']['0']=array();
$dPrec['updatearticle']['1']=array(
    array('action'=>'updatearticle','phase'=>'0'),
);
$dPrec['updatearticle']['2']=array(
    array('action'=>'updatearticle','phase'=>'1'),
    array('action'=>'updatearticle','phase'=>'3')
);
$dPrec['updatearticle']['3']=array(
    array('action'=>'updatearticle','phase'=>'2'),
);

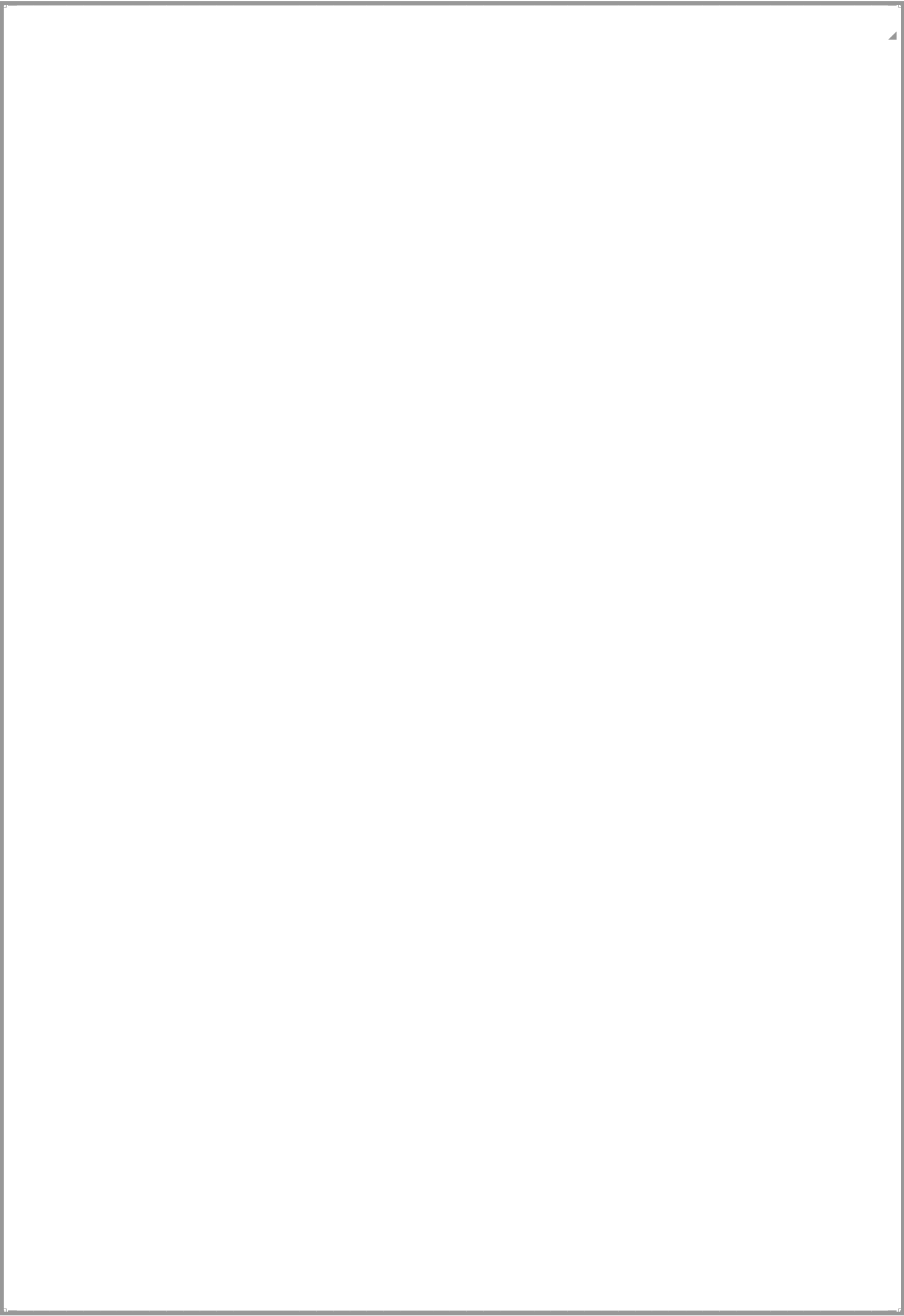
// suppression d'article
$dPrec['deletearticle']['0']=array();
$dPrec['deletearticle']['1']=array(
    array('action'=>'deletearticle','phase'=>'0'),
);
$dPrec['deletearticle']['2']=array(
    array('action'=>'deletearticle','phase'=>'1'),
);

// sélection d'articles
$dPrec['selectarticle']['0']=array();
$dPrec['selectarticle']['1']=array(
    array('action'=>'selectarticle','phase'=>'0'),
    array('action'=>'selectarticle','phase'=>'2')
);
$dPrec['selectarticle']['2']=array(
    array('action'=>'selectarticle','phase'=>'1'),
);

// requête administrateur
$dPrec['sql']['0']=array();
$dPrec['sql']['1']=array(
    array('action'=>'sql','phase'=>'0'),
    array('action'=>'sql','phase'=>'2')
);
$dPrec['sql']['2']=array(
    array('action'=>'sql','phase'=>'1'),
);
```

\$dPrec['action']['phase'] est un tableau qui contient les actions qui peuvent précéder l'action et la phase qui servent d'index au dictionnaire. Ces actions précédentes sont elles aussi représentées par un dictionnaire à deux clés 'action' et 'phase'. Si une action peut être précédée par toute action alors **\$dPrec['action']['phase']** sera un tableau vide. L'absence d'une action dans le dictionnaire signifie qu'elle n'est pas autorisée. Considérons l'action "authentifier" ci-dessus :

```
// authentification
$dPrec['authentifier']['0']=array();
$dPrec['authentifier']['1']=array(
```



```

    array('action'=>'authentifier', 'phase'=>'0'),
    array('action'=>'authentifier', 'phase'=>'2')
);
$dPrec['authentifier'][$'2']=array(
    array('action'=>'authentifier', 'phase'=>'1'),
);

```

Le code ci-dessus signifie que l'action *action=authentifier&phase=0* peut être précédée de toute action, que *action=authentifier&phase=1* peut être précédée de *action=authentifier&phase=0* ou de *action=authentifier&phase=2* et que *action=authentifier&phase=2* peut être précédée de l'action *action=authentifier&phase=1*.

Écrire la fonction suivante :

```

// -----
function enchainementOK(&$dConfig, &$dSession, $sAction, $sPhase) {
    // vérifie si l'action en cours ($sAction, $sPhase) peut suivre l'action précédente
    // mémorisée dans $dSession['précédent']
    // le dictionnaire des enchaînements autorisés est dans $dConfig['précédents']
    // rend TRUE si l'enchaînement est possible, FALSE sinon
}

```

Cette fonction permet à l'application principale de vérifier que l'enchaînement des actions est correct :

```

<?php
// gestion d'une table d'articles include
"config.php";
include "articles.php";

// session
session_start();
$dSession=$_SESSION["session"];

// action à entreprendre
$sAction=$_POST["action"] ? $_POST["action"] : $_GET["action"] ? $_GET["action"] : "authentifier";
$sAction=strtolower($sAction);
// phase éventuelle de l'action
$sPhase=$_POST["phase"] ? $_POST["phase"] : $_GET["phase"] ? $_GET["phase"] : "0";

// y-a-t-il une session en cours ? if(!
isset($dSession)){
    // authentification de l'utilisateur
    if($sAction=="authentifier" && $sPhase=="0") authentifier_0($dConfig);
    if($sAction=="authentifier" && $sPhase=="1") authentifier_1($dConfig);
    if($sAction=="authentifier" && $sPhase=="2") authentifier_2($dConfig);
    // action anormale
    authentifier_0($dConfig);
} //if - pas de session

// on récupère la session
$dSession=unserialize($dSession);

// l'enchaînement des actions est-il normal ?
if( ! enchainementOK($dConfig,$dSession,$sAction,$sPhase) ){
    // enchaînement anormal
    authentifier_0($dConfig);
} //if

// traitement des actions
if($sAction=="authentifier"){
    if($sPhase=="0") authentifier_0($dConfig);
    if($sPhase=="1") authentifier_1($dConfig);
    if($sPhase=="2") authentifier_2($dConfig);
} //if
if($sAction=="addarticle"){

```

Faire évoluer le "look"

Rappelons-nous qu'une des conditions posées lors de l'étude de cette application était qu'elle devait être évolutive. Supposons qu'au bout de quelques semaines, on s'aperçoive que l'ergonomie de l'application doit être améliorée. Modifiez l'application de telle façon que la structure et la présentation de la page type soient changées. Les modifications auront lieu à deux endroits :

1. dans le script **main.php** qui définit la structure de la page type. Faites évoluer celle-ci.
2. dans la feuille de style qui donne le "look" de l'application. Changez celui-ci.

Améliorer les performances

Pour l'instant, nous avons opté pour un navigateur client léger : il ne fait rien d'autre que de la présentation. On peut lui faire faire du traitement en incluant des scripts dans les pages Web qu'on lui envoie. Ceux-ci peuvent être en différents langages, notamment vbscript et javascript. Internet Explorer et Netscape dominent le marché des navigateurs dans une proportion proche de 60/40. Par ailleurs, IE n'existe que dans le domaine Windows et pas sur Unix par exemple où c'est Netscape qui prédomine. Netscape n'exécute pas nativement les scripts vbscript alors que les deux navigateurs exécutent les scripts javascript. Comme Netscape occupe encore une part significative du marché des navigateurs, les scripts vbscript sont à éviter. C'est donc javascript qui est généralement utilisé dans les scripts côté client.

Sont délégués aux scripts côté client, des traitements où le serveur n'a pas à intervenir. Dans notre application, il serait intéressant que le navigateur client n'envoie une demande au serveur qu'après l'avoir vérifiée. Ainsi il est inutile d'envoyer au serveur une demande d'authentification alors que l'utilisateur a laissé le champ [login] vide dans le formulaire d'authentification. Il est préférable de prévenir l'utilisateur que sa demande est erronée :



On remarquera que cela n'empêchera pas le serveur de vérifier que le champ login est non vide car son client n'est pas forcément un navigateur et alors la vérification précédente a pu ne pas être faite. Faire l'hypothèse que le client est un navigateur est un risque majeur pour la sécurité de l'application.

Reprenez les différents moments où le navigateur envoie des informations au serveur et lorsque celles-ci peuvent être vérifiées, écrivez une ou des fonctions javascript qui permettront au navigateur de vérifier la validité des informations avant leur envoi au serveur.

Pour reprendre l'exemple précédent, le script **login.php** qui génère la page d'authentification devient le suivant :

```
<script language="javascript">
function check() {
    // on vérifie qu'il y a bien un login
    with(document.frmLogin) {
        champs=/^%s $/.exec(txtLogin.value);
        if(champs!=null) {
            // pas de login
            alert("Vous n'avez pas indiqué de login");
            txtLogin.focus();
            return;
        } //if
        // les données sont là - on les envoie au serveur submit();
    } //with
} //check
</script>

<form name="frmLogin" method="post" action="<?php echo $main["post"] ?>">
<table>
<tr>
<td>login</td>
<td><input type="text" value="<?php echo $main["login"] ?>" name="txtLogin" class="text"></td>
</tr>
</table>
```



```

<td>mot de passe</td>
<td><input type="password" value="" name="txtMdp" class="text"></td>
<td><input type="button" onclick="check()" value="Connexion" class="submit"></td>
</tr>
</table>
</form>

```

Pour aller plus loin

Pour terminer, signalons quelques pistes pour approfondir cette étude de cas :

- il serait intéressant de voir si la **page type** de cette application ne pourrait pas faire l'objet d'une **classe**. Celle-ci pourrait être alors utilisée dans d'autres applications.
- notre application est bien adaptée à des clients de type navigateur mais moins à des clients de type "Application autonome". Celles-ci doivent :
 - créer une connexion tcp avec le serveur
 - lui "parler" HTTP
 - analyser ses réponses HTML pour y trouver l'information désirée puisque le client autonome ne sera probablement pas intéressé par le code HTML de présentation destiné aux navigateurs.

Il serait intéressant que notre application **génère du XML** plutôt que du HTML. Ses clients pourraient alors être indifféremment des navigateurs (assez récents quand même) ou des applications autonomes. Ces dernières n'auraient aucune difficulté à retrouver l'information qu'elles recherchent puisque la réponse XML du serveur ne contiendrait aucune information de présentation, seulement du contenu.

- il faudrait très certainement s'intéresser aux **accès simultanés** à la base d'articles. Il y a au moins deux points à éclaircir :
 - est-ce que le SGBD utilisé par l'application gère correctement l'accès simultané à un même article ? Par exemple, que se passe-t-il si deux utilisateurs modifient le même article au même moment (ils appuient sur le bouton [Modifier] en même temps) ? Cela dépend probablement du SGBD sous-jacent.
 - actuellement notre application ne gère pas les accès simultanés. Cependant, la base devrait rester dans un état cohérent même si des surprises sont à attendre. Prenons la séquence d'événements suivante :
 - l'utilisateur U1 entre en modification d'un article
 - l'utilisateur U2 entre en suppression du même article un peu après
 - chacune des deux actions nécessite des échanges client-serveur. Selon la façon de travailler de chacun, l'utilisateur U2 peut terminer son travail avant U1. Lorsque celui-ci va terminer ses modifications et les valider par [Modifier], il aura la page d'informations en réponse, le SGBD lui indiquant que **[0 ligne(s) ont été modifiées]**, ceci parce que la page qu'il voulait modifier a été supprimée entre-temps. L'utilisateur sera sans doute surpris. D'un point de vue ergonomie, il serait sans doute préférable d'afficher une page signalant mieux l'erreur. Par ailleurs, on pourrait envisager d'offrir à l'utilisateur un accès exclusif à un article dès qu'il entrerait en mise à jour de celui-ci. Un autre utilisateur voulant entrer en mise à jour du même article se verrait répondre qu'une autre mise à jour est en cours. Cela posera problème si le premier utilisateur tarde à valider sa mise à jour : les autres seront bloqués. Il y a là des solutions à trouver qui dépendront assez largement des capacités du SGBD utilisé. Oracle a, par exemple, davantage de capacités dans ce domaine que MySQL.

ANNEXE

PEAR DB: a unified API for accessing SQL-databases

This chapter describes how to use the PEAR database abstraction layer.

DSN -- The data source name

Connect -- Connecting and disconnecting

Query -- Performing a query against a database.

Fetch -- Fetching rows from the query

DSN

To connect to a database through PEAR::DB, you have to create a valid DSN - data source name. This DSN consists in the following parts:

phptype: Database backend used in PHP (i.e. mysql, odbc etc.)

dbsyntax: Database used with regards to SQL syntax etc.

protocol: Communication protocol to use (i.e. tcp, unix etc.)

hostspec: Host specification (hostname[:port])

database: Database to use on the DBMS server

username: User name for login

password: Password for login

proto_opts: Maybe used with *protocol*

The format of the supplied DSN is in its fullest form:

```
phptype(dbsyntax)://username:password@protocol+hostspec/database
```

Most variations are allowed:

```
phptype://username:password@protocol+hostspec:110//usr/db_file.db
```

```
phptype://username:password@hostspec/database_name
```

```
phptype://username:password@hostspec
```

```
phptype://username@hostspec
```

```
phptype://hostspec/database
```

```
phptype://hostspec
```

```
phptype(dbsyntax)
```

```
phptype
```

The currently supported database backends are:

```
mysql -> MySQL
```

```
pgsql -> PostgreSQL
```

```
ibase -> InterBase
```

```
msql -> Mini SQL
```

```
mssql -> Microsoft SQL Server
```

```
oci8 -> Oracle 7/8/8i
```

```
odbc -> ODBC (Open Database Connectivity)
```

```
sybase -> SyBase
```

```
ifx -> Informix
```

```
fbsql -> FrontBase
```

With an up-to-date version of **DB**, you can use a second DSN format

```
phptype(syntax)://user:pass@protocol(proto_opts)/database
```

example: Connect to database through a socket
`mysql://user@unix(/path/to/socket)/pear`

Connect to database on a non standard port
`pgsql://user:pass@word@tcp(localhost:5555)/pear`

Connect

To connect to a database you have to use the function **DB::connect()**, which requires a valid DSN as parameter and optional a boolean value, which determines whether to use a persistent connection or not. In case of success you get a new instance of the database class. It is strongly recommended to check this return value with **DB::isError()**. To disconnect use the method **disconnect()** from your database class instance.

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

$user = 'foo';
$pass = 'bar';
$host = 'localhost';
$db_name = 'clients_db';

//Data Source Name: This is the universal connection string
$dsn = "mysql://$user:$pass@$host/$db_name";

// DB::connect will return a PEAR DB object on success
// or an PEAR DB Error object on error
$db = DB::connect($dsn, true);

// Alternatively: $db = DB::connect($dsn);

// With DB::isError you can differentiate between an error or
// a valid connection.
if (DB::isError($db)) {
    die ($db->getMessage());
}
....
// You can disconnect from the database with:
$db->disconnect();
?>
```

Query

To perform a query against a database you have to use the function **query()**, that takes the query string as an argument. On failure you get a DB Error object, check it with **DB::isError()**. On success you get *DB_OK* (predefined PEAR::DB constant) or when you set a *SELECT*-statement a DB Result object.

```
<?php
// Once you have a valid DB object...
$sql = "select * from clients";
$result = $db->query($sql);

// Always check that $result is not an error
if (DB::isError($result)) {
    die ($result->getMessage());
}
....
?>
```

Fetch

The **DB_Result** object provides two functions to fetch rows: **fetchRow()** and **fetchInto()**. **fetchRow()** returns the row, null on no more data or a **DB_Error**, when an error occurs. **fetchInto()** requires a variable, which be will directly assigned by reference to the result row. It will return null when result set is empty or a **DB_Error** too.

```
<?php
...
$db = DB::connect($dsn);
$res = $db->query("select * from mytable");

//Get each row of data on each iteration until
// there is no more rows
while ($row = $res->fetchRow()) {
    $id = $row[0];
}
// Or:
//an example using fetchInto()
while ($res->fetchInto($row)) {
    $id = $row[0];
}
?>
```

Select the format of the fetched row

The fetch modes supported are:

1. `DB_FETCHMODE_ORDERED` (default) : returns an ordered array. The order is taken from the select statment.

```
<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_ORDERED);
/*
$row will contain:
array (
    0 => <column "id" data>,
    1 => <column "name" data>,
    2 => <column "email" data>
)
*/
// Access the data with:
$id = $row[0];
$name = $row[1];
$email = $row[2];
?>
```

1. `DB_FETCHMODE_ASSOC` : returns an associative array with the column names as the array keys

```

<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_ASSOC);
/*
$row will contain:
array (
    'id' => <column "id" data>,
    'name' => <column "name" data>,
    'email' => <column "email" data>
)
*/
// Access the data with:
$id = $row['id'];
$name = $row['name'];
$email = $row['email'];
?>

```

1. `DB_FETCHMODE_OBJECT`: returns a `DB_row` object with column names as properties

```

<?php
$res = $db->query('select id, name, email from users');
$row = $res->fetchRow(DB_FETCHMODE_OBJECT);
/*
$row will contain:
db_row Object
(
    [id] => <column "id" data>,
    [name] => <column "name" data>,
    [email] => <column "email" data>
)
*/
// Access the data with:
$id = $row->id;
$name = $row->name;
$email = $row->email;
?>

```

Set the format of the fetched row

You can set the fetch mode for every call or for your whole DB instance.

```

<?php
// 1) Set the mode per call:
while ($row = $result->fetchRow(DB_FETCHMODE_ASSOC)) {
    $id = $row['id'];
}

// 2) Set the mode for all calls:
$db = DB::connect($dsn);
// this will set a default fetchmode for this Pear DB instance
// (for all queries)
$db->setFetchMode(DB_FETCHMODE_ASSOC);

$result = $db->query(...);
while ($row = $result->fetchRow()) {
    $id = $row['id'];
}
?>

```

Fetch rows by number

The PEAR DB fetch system also supports an extra parameter to the fetch statement. So you can fetch rows from a result by number. This is especially helpful if you only want to show sets of an entire result (for example in building paginated HTML lists), fetch rows in an special order, etc.

```
<?php
...
// the row to start fetching
$from = 50;

// how many results per page
$res_per_page = 10;

// the last row to fetch for this page
$to = $from + $res_per_page;

foreach (range($from, $to) as $rownum) {
    if (!$row=$res->fetchrow($fetchmode, $rownum)) {
        break;
    }
    $id = $row[0];
    ....
}
?>
```

Freeing the result set

It is recommended to finish the result set after processing in order to save memory. Use **free()** to do this.

```
<?php
...
$result=$db->query('SELECT *FROM clients');
while ($row = $result->fetchRow()) {
    ...
}
$result->free();
?>
```

Quick data retrieving

PEAR DB provides some special ways to retrieve information from a query without the need of using **fetch*()** and loop throw results.

getOne() retrieves the first result of the first column from a query

```
$numrows = $db->getOne('select count(id) from clients');
```

getRow() returns the first row and return it as an array

```
$sql = 'select name, address, phone from clients where id=1';
if (is_array($row = $db->getRow($sql))) {
    list($name, $address, $phone) = $row;
}
```

getCol() returns an array with the data of the selected column. It accepts the column number to retrieve as the second param.

```
$all_client_names = $db->getCol('select name from clients');
```

The above sentence could return for example: `$all_client_names = array('Stig', 'Jon', 'Colin');`

getAssoc() fetches the entire result set of a query and return it as an associative array using the first column as the key.

```
$data = getAssoc('SELECT name, surname, phone FROM mytable')
/*
Will return:
array(
  'Peter' => array('Smith', '944679408'),
  'Tomas' => array('Cox', '944679408'),
  'Richard' => array('Merz', '944679408')
)
*/
```

```
getAll() fetches all the rows returned from a query
$data = getAll('SELECT id, text, date FROM mytable');
/*
Will return:
array(
  1 => array('4', 'four', '2004'),
  2 => array('5', 'five', '2005'),
  3 => array('6', 'six', '2006')
)
*/
```

The **get*()** family methods will do all the dirty job for you, this is: launch the query, fetch the data and free the result. Please note that as all PEAR DB functions they will return a **PEAR DB_error** object on errors.

Getting more information from query results

With PEAR DB you have many ways to retrieve useful information from query results. These are:

1. **numRows()**: Returns the total number of rows returned from a "SELECT" query.

```
// Number of rows
echo $res->numRows();
```
1. **numCols()**: Returns the total number of columns returned from a "SELECT" query.

```
// Number of cols
echo $res->numCols();
```
1. **affectedRows()**: Returns the number of rows affected by a data manipulation query ("INSERT", "UPDATE" or "DELETE").

```
// remember that this statement won't return a result object
$db->query('DELETE * FROM clients');
echo 'I have deleted ' . $db->affectedRows() . ' clients';
```
1. **tableInfo()**: Returns an associative array with information about the returned fields from a "SELECT" query.

```
// Table Info
print_r($res->tableInfo());
```

Annexes

7. Les outils du développement web

Nous indiquons ici où trouver et comment installer les outils nécessaires au développement web. Certains outils ont vu leurs versions évoluer et il se peut que les explications données ici ne conviennent plus pour les versions les plus récentes. Le lecteur sera alors amené à s'adapter... Dans le cours de programmation web, nous utiliserons essentiellement les outils suivants, tous disponibles gratuitement :

- un **navigateur** récent capable d'afficher du XML. Les exemples du cours ont été testés avec Internet Explorer 6.
- un **JDK** (Java Development Kit) récent. Les exemples du cours ont été testés avec le JDK 1.4. Ce JDK amène avec lui le Plug-in Java 1.4 pour les navigateurs ce qui permet à ces derniers d'afficher des applets Java utilisant le JDK 1.4.
- un **environnement de développement Java** pour écrire des servlets Java. Ici c'est JBuilder 7.
- des **serveurs web** : Apache, PWS (Personal Web Server), Tomcat.
 - Apache sera utilisé pour le développement d'applications web en PERL (Practical Extracting and Reporting Language) ou PHP (Personal Home Page)
 - PWS sera utilisé pour le développement d'applications web en ASP (Active Server Pages) ou PHP
 - Tomcat sera utilisé pour le développement d'applications web à l'aide de servlets Java ou de pages JSP (Java Server pages)
- une **application de gestion de base de données** : MySQL
- **EasyPHP** : un outil qui amène ensemble le serveur Web Apache, le langage PHP et le SGBD MySQL

Serveurs Web, Navigateurs, Langages de scripts

1. **Serveurs Web principaux**
 - Apache (Linux, Windows)
 - Internet Information Server IIS (NT), Personal Web Server PWS (Windows 9x)
2. **Navigateurs principaux**
 - Internet Explorer (Windows)
 - Netscape (Linux, Windows)
3. **Langages de scripts côté serveur**
 - VBScript (IIS, PWS)
 - JavaScript (IIS, PWS)
 - Perl (Apache, IIS, PWS)
 - PHP (Apache, IIS, PWS)
 - Java (Apache, Tomcat)
 - Langages .NET
4. **Langages de scripts côté navigateur**
 - VBScript (IE)
 - Javascript (IE, Netscape)
 - Perlscript (IE)
 - Java (IE, Netscape)

Où trouver les outils

Netscape	http://www.netscape.com/ (lien downloads)
Internet Explorer	http://www.microsoft.com/windows/ie/default.asp
PHP	http://www.php.net http://www.php.net/downloads.php (Windows Binaries)
PERL	http://www.activestate.com http://www.activestate.com/Products/ http://www.activestate.com/Products/ActivePerl/

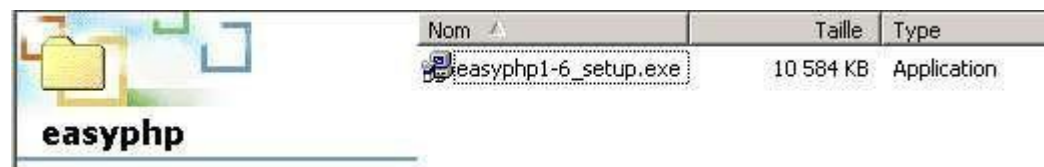
Vbscript, Javascript JAVA	http://msdn.microsoft.com/scripting (suivre le lien windows script) http://java.sun.com/ http://java.sun.com/downloads.html (JSE) http://java.sun.com/j2se/1.4/download.html
Apache	http://www.apache.org/ http://www.apache.org/dist/httpd/binaries/win32/
PWS	inclus dans NT 4.0 Option pack for Windows 95 inclus dans le CD de Windows 98 http://www.microsoft.com/ntserver/nts/downloads/recommended/NT4OptPk/win95.asp http://www.microsoft.com
IIS (windows NT/2000)	
Tomcat	http://jakarta.apache.org/tomcat/
JBuiler	http://www.borland.com/jbuilder/ http://www.borland.com/products/downloads/download_jbuilder.html
EasyPHP	http://www.easyphp.org/ http://www.easyphp.org/telechargements.php3

EasyPHP

Cette application est très pratique en ce qu'elle amène dans un même paquetage :

- le serveur Web Apache (1.3.x)
- le langage PHP (4.x)
- le SGBD MySQL (3.23.x)
- un outil d'administration de MySQL : PhpMyAdmin

L'application d'installation se présente sous la forme suivante :



L'installation d'EasyPHP ne pose pas de problème et une arborescence est créée dans le système de fichiers :

Address: C:\Program Files\EasyPHP			
Nom	Taille	Type	Modifié le
apache		Dossier de fichiers	24/05/2002 08:18
cgi-bin		Dossier de fichiers	24/05/2002 08:18
home		Dossier de fichiers	24/05/2002 08:18
mysql		Dossier de fichiers	24/05/2002 08:18
php		Dossier de fichiers	24/05/2002 08:18
phpmyadmin		Dossier de fichiers	24/05/2002 08:18
safe		Dossier de fichiers	24/05/2002 08:18
tmp		Dossier de fichiers	24/05/2002 08:18
www		Dossier de fichiers	24/05/2002 08:18
easyphp.exe	120 KB	Application	15/04/2002 10:52
easyphp.ini	1 KB	Paramètres de confi...	09/07/2002 18:37
EasyPHP.log	5 KB	Fichier LOG	18/07/2002 07:53
licences.txt	25 KB	Texte seulement	02/04/2002 14:36
phpini.exe	452 KB	Application	10/06/2001 10:36
unins000.dat	74 KB	Fichier DAT	24/05/2002 08:18
unins000.exe	73 KB	Application	24/05/2002 08:18

easyphp.exe	l'exécutable de l'application
apache	l'arborescence du serveur apache

mysql l'arborescence du SGBD mysql
 phpmyadmin l'arborescence de l'application phpmyadmin
 php l'arborescence de php
 www racine de l'arborescence des pages web délivrées par le serveur apache d'EasyPHP
 cgi-bin arborescence où l'on peut palcer des script CGI pour le serveur Apache

L'intérêt principal d'EasyPHP est que l'application arrive préconfigurée. Ainsi Apache, PHP, MySQL sont déjà configurés pour travailler ensemble. Lorsqu'on lance *EasyPhp* par son lien dans le menu des programmes, une icône se met en place en bas à droite de l'écran.



C'est le E avec un point rouge qui doit clignoter si le serveur web *Apache* et la base de données *MySQL* sont opérationnels. Lorsqu'on clique dessus avec le bouton droit de la souris, on accède à des options de menu :



L'option *Administration* permet de faire des réglages et des tests de bon fonctionnement :



Administration PHP

Le bouton **infos php** doit vous permettre de vérifier le bon fonctionnement du couple Apache-PHP : une page d'informations PHP doit apparaître :

■ **Environnement EasyPHP :**
Ces pages vous informeront sur le bon fonctionnement de PHP, sa configuration et sur les éléments installés.

[infos php](#) [extensions](#) [paramètres](#) [retour](#)

PHP Version 4.2.0



System	Windows NT 5.0 build 2195
Build Date	Apr 20 2002 18:36:03
Server API	Apache
Virtual Directory Support	enabled
Configuration File (php.ini) Path	c:\winnt

Le bouton **extensions** donne la liste des extensions installées pour php. Ce sont en fait des bibliothèques de fonctions.

Gestion des extensions : vous avez 12 extensions chargées :

- [apache](#) [listes des fonctions](#)
- [bcmath](#) [listes des fonctions](#)
- [calendar](#) [listes des fonctions](#)
- [com](#) [listes des fonctions](#)
- [ftp](#) [listes des fonctions](#)
- [mysql](#) [listes des fonctions](#)
- [odbc](#) [listes des fonctions](#)
- [pcre](#) [listes des fonctions](#)
- [session](#) [listes des fonctions](#)
- [standard](#) [listes des fonctions](#)
- [wddx](#) [listes des fonctions](#)
- [xml](#) [listes des fonctions](#)

L'écran ci-dessus montre par exemple que les fonctions nécessaires à l'utilisation de la base MySQL sont bien présentes.

Le bouton **paramètres** donne le *login/motdepasse* de l'administrateur de la base de données MySQL.

Paramètres par défaut de la base de données :

```

serveur : "localhost"
username : "root"
mot de passe : ""

```

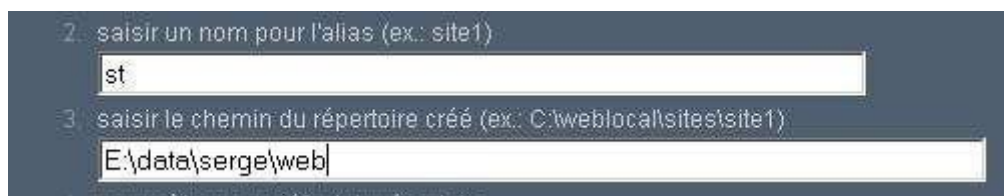
L'utilisation de la base MySQL dépasse le cadre de cette présentation rapide mais il est clair ici qu'il faudrait mettre un mot de passe à l'administrateur de la base.

Administration Apache

Toujours dans la page d'administration d'EasyPHP, le lien **vos alias** permet de définir des alias associés à un répertoire. Cela permet de mettre des pages Web ailleurs que dans le répertoire www de l'arborescence d'easyPhp.



Si dans la page ci-dessus, on met les informations suivantes :



et qu'on utilise le bouton *valider* les lignes suivantes sont ajoutées au fichier `<easyphp>\apache\conf\httpd.conf` :

```
Alias /st/ "e:/data/serge/web/"

<Directory "e:/data/serge/web"> Options
    FollowSymLinks Indexes AllowOverride
    None
    Order deny,allow allow
    from 127.0.0.1 deny from
    all
</Directory>
```

`<easyphp>` désigne le répertoire d'installation d'EasyPHP. `httpd.conf` est le fichier de configuration du serveur Apache. On peut donc faire la même chose en éditant directement ce fichier. Une modification du fichier `httpd.conf` est normalement prise en compte immédiatement par Apache. Si ce n'était pas le cas, il faudrait l'arrêter puis le relancer, toujours avec l'icône d'easyphp :



Pour terminer notre exemple, on peut maintenant placer des pages web dans l'arborescence `e:\data\serge\web` :

```
dos>dir e:\data\serge\web\html\balises.htm
14/07/2002  17:02                3 767 balises.htm
```

et demander cette page en utilisant l'alias `st` :

Address  http://localhost:81/st/html/balises.htm

Les balises HTML

cellule(1,1)	cellule(1,2)	cellule(1,3)
cellule(2,1)	cellule(2,2)	cellule(2,3)

Une image



le site de l'ISTIA [ici](#)

Dans cet exemple, le serveur Apache a été configuré pour travailler sur le port 81. Son port par défaut est 80. Ce point est contrôlé par la ligne suivante du fichier *httpd.conf* déjà rencontré :

Port 81

Le fichier de configuration d'Apache [httpd.conf]

Lorsqu'on veut configurer un peu finement Apache, on est obligé d'aller modifier "à la main" son fichier de configuration *httpd.conf* situé ici dans le dossier *<easyphp>\apache\conf* :



Voici quelques points à retenir de ce fichier de configuration :

ligne(s)	rôle
ServerRoot "D:/Program Files/Apache Group/Apache"	indique le dossier où se trouve l'arborescence de Apache
Port 80	indique sur quel port va travailler le serveur Web. Classiquement c'est 80. En changeant cette ligne, on peut faire travailler le serveur Web sur un autre port
ServerAdmin root@istia.univ-angers.fr	l'adresse email de l'administrateur du serveur Apache

ServerName stahe.istia.uang	le nom de la machine sur laquelle "tourne" le serveur Apache
ServerRoot "E:/Program Files/EasyPHP/apache"	le répertoire d'installation du serveur Apache. Lorsque dans le fichier de configuration, apparaissent des noms relatifs de fichiers, ils sont relatifs par rapport à ce dossier.
DocumentRoot "E:/Program Files/EasyPHP/www"	le dossier racine de l'arborescence des pages Web délivrées par le serveur. Ici, l'url <i>http://machine/rep1/fic1.html</i> correspondra au fichier <i>E:\Program Files\EasyPHP\www\rep1\fic1.html</i>
<Directory "E:/Program Files/EasyPHP/www">	fixe les propriétés du dossier précédent
ErrorLog logs/error.log	dossier des logs, donc en fait <i><ServerRoot>\logs\error.log : E:\Program Files\EasyPHP\apache\logs\error.log</i> . C'est le fichier à consulter si vous constatez que le serveur Apache ne fonctionne pas.
ScriptAlias /cgi-bin/ "E:/Program Files/EasyPHP/cgi-bin/"	<i>E:\Program Files\EasyPHP\cgi-bin</i> sera la racine de l'arborescence où l'on pourra mettre des scripts CGI. Ainsi l'URL <i>http://machine/cgi-bin/rep1/script1.pl</i> sera l'url du script CGI <i>E:\Program Files\EasyPHP\cgi-bin\rep1\script1.pl</i> .
<Directory "E:/Program Files/EasyPHP/cgi-bin/">	fixe les propriétés du dossier ci-dessus
LoadModule php4_module "E:/Program Files/EasyPHP/php/php4apache.dll" AddModule mod_php4.c	lignes de chargement des modules permettant à Apache de travailler avec PHP4.
AddType application/x-httpd-php .phtml .phtml .php3 .php4 .php .php2 .inc	fixe les suffixes des fichiers à considérer comme des fichiers comme devant être traités par PHP

Administration de MySQL avec PhpMyAdmin

Sur la page d'administration d'EasyPhp, on clique sur le bouton **PhpMyAdmin** :

Address  http://127.0.0.1/mysql/

Accueil

-

Choisissez une base de données

Bienvenue à phpMyAdmin 2.2.6

MySQL 3.23.49-max-nt sur le serveur localhost - utilisateur : root@localhost

MySQL

- Créer une base de données [Documentation]
- Afficher l'état du serveur MySQL [Documentation]
- Afficher les variables du serveur MySQL [Documentation]
- Afficher les processus [Documentation]
- Recharger MySQL [Documentation]
- Utilisateurs et privilèges [Documentation]
- Statistiques sur les bases de données

phpMyAdmin

- Language:

French (fr)
- Documentation de phpMyAdmin**
 - Afficher les informations relatives à PHP
 - Site officiel de phpMyAdmin

[ChangeLog] [CVS] [Lists]

La liste déroulante sous *Accueil* permet de voir les bases de données actuelles.

Le nombre entre parenthèses est le nombre de tables. Si on choisit une base, les tables de celles-ci s'affichent :



La page Web offre un certain nombre d'opérations sur la base :

Base de données *mysql* sur le serveur *localhost*

	Table	Action						Enregistrements	Type	Taille
<input type="checkbox"/>	columns_priv	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	0	MyISAM	1,0 Ko
<input type="checkbox"/>	db	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	1	MyISAM	3,1 Ko
<input type="checkbox"/>	func	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	0	MyISAM	1,0 Ko
<input type="checkbox"/>	host	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	0	MyISAM	1,0 Ko
<input type="checkbox"/>	tables_priv	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	0	MyISAM	1,0 Ko
<input type="checkbox"/>	user	Afficher	Sélectionner	Insérer	Propriétés	Supprimer	Vider	1	MyISAM	2,4 Ko
6 table(s)		Somme						2	--	9,6 Ko
<div><div><div></div><div>Tout cocher / Tout décocher</div></div><div>Pour la sélection : <div></div></div></div>										

Si on clique sur le lien *Afficher* de *user* :

Base de données *mysql* - table *user* sur le serveur *localhost*

requête SQL

requête SQL : [Modifier]
SELECT * FROM 'user' LIMIT 0, 30

		Host	User	Password	Select_priv	Insert_priv	Update_priv
Modifier	Effacer	localhost	root		Y	Y	Y

Insérer un nouvel enregistrement

Il n'y a ici qu'un seul utilisateur : root, qui est l'administrateur de MySQL. En suivant le lien *Modifier*, on pourrait changer son mot de passe qui est actuellement vide, ce qui n'est pas conseillé pour un administrateur.

Nous n'en dirons pas plus sur *PhpMyAdmin* qui est un logiciel riche et qui mériterait un développement de plusieurs pages.

PHP

Nous avons vu comment obtenir PHP au travers de l'application EasyPhp. Pour obtenir PHP directement, on ira sur le site <http://www.php.net>.

PHP n'est pas utilisable que dans le cadre du Web. On peut l'utiliser comme langage de scripts sous Windows. Créez le script suivant et sauvegardez-le sous le nom *date.php* :

```
<?
// script php affichant l'heure
$maintenant=date("j/m/y, H:i:s",time()); echo
"Nous sommes le $maintenant";
?>
```

Dans une fenêtre DOS, placez-vous dans le répertoire de *date.php* et exécutez-le :

```
E:\data\serge\php\essais>"e:\program files\easyphp\php\php.exe"date.php
X-Powered-By: PHP/4.2.0
Content-type: text/html
```

```
Nous sommes le 18/07/02, 09:31:01
```

PERL

Il est préférable qu'Internet Explorer soit déjà installé. S'il est présent, Active Perl va le configurer afin qu'il accepte des scripts PERL dans les pages HTML, scripts qui seront exécutés par IE lui-même côté client. Le site de Active Perl est à l'URL <http://www.activestate.com> l'installation, PERL sera installé dans un répertoire que nous appellerons **<perl>**. Il contient l'arborescence suivante :

```
DEISL1 ISU 32 403 23/06/00 17:16 DeIsL1.isu
BIN <REP>23/06/00 17:15 bin
LIB <REP> 23/06/00 17:15 lib
HTML <REP> 23/06/00 17:15 html
EG <REP> 23/06/00 17:15 eg
SITE <REP> 23/06/00 17:15 site
HTMLHELP <REP> 28/06/00 18:37 htmlhelp
```


L'exécutable *perl.exe* est dans <perl>\bin. Perl est un langage de scripts fonctionnant sous Windows et Unix. Il est de plus utilisé dans la programmation WEB. Écrivons un premier script :

```
# script PERL affichant l'heure
# modules use
strict;
# programme
my ($secondes,$minutes,$heure)=localtime(time); print
"Il est $heure:$minutes:$secondes\n";
```

Sauvegardez ce script dans un fichier *heure.pl*. Ouvrez une fenêtre DOS, placez-vous dans le répertoire du script précédent et exécutez-le :

```
dos>e:\perl\bin\perl.exe heure.pl
Il est 9:34:21
```

Vbscript, Javascript, Perlscript

Ces langages sont des langages de script pour windows. Ils peuvent fonctionner dans différents conteneurs tels

- *Windows Scripting Host* pour une utilisation directe sous Windows notamment pour écrire des scripts d'administration système
- *Internet Explorer*. Il est alors utilisé au sein de pages HTML auxquelles il amène une certaine interactivité impossible à atteindre avec le seul langage HTML.
- *Internet Information Server* (IIS) le serveur Web de Microsoft sur NT/2000 et son équivalent *Personal Web Server* (PWS) sur Win9x. Dans ce cas, vbscript est utilisé pour faire de la programmation côté serveur web, technologie appelée ASP (*Active Server Pages*) par Microsoft.

On récupère le fichier d'installation à l'URL : <http://msdn.microsoft.com/scripting> et on suit les liens *Windows Script*. Sont installés :

- le conteneur *Windows Scripting Host*, conteneur permettant l'utilisation de divers langages de scripts, tels Vbscript et Javascript mais aussi d'autres tel PerlScript qui est amené avec ActivePerl.
- un interpréteur VBscript
- un interpréteur Javascript

Présentons quelques tests rapides. Construisons le programme *vbscript* suivant :

```
' une classe
class personne
    Dim nom
    Dim age End
class

' création d'un objet personne Set
p1=new personne
With p1
    .nom="dupont"
    .age=18
End With

' affichage propriétés personne p1 With
p1
    wscript.echo "nom=" & .nom
    wscript.echo "age=" & .age
End With
```

Ce programme utilise des objets. Appelons-le *objets.vbs* (le suffixe vbs désigne un fichier vbscript). Positionnons-nous sur le répertoire dans lequel il se trouve et exécutons-le :

```
dos>cscript objets.vbs
Microsoft (R) Windows Script Host Version 5.6

nom=dupont
age=18
```

Maintenant construisons le programme javascript suivant qui utilise des tableaux :

```

// tableau dans un variant
// tableau vide
tableau=new Array();
affiche(tableau);
// tableau croît dynamiquement
for(i=0;i<3;i++){
    tableau.push(i 10);
}
// affichage tableau*
affiche(tableau);
// encore
for(i=3;i<6;i++){
    tableau.push(i 10);
}
affiche(tableau);    *

// tableaux à plusieurs dimensions WScript.echo("-----
                           ");

tableau2=new Array();
for(i=0;i<3;i++){
    tableau2.push(new Array());
    for(j=0;j<4;j++){
        tableau2[i].push(i 10+j);
    }//for j
} // for i
affiche2(tableau2);

// fin
WScript.quit(0);

// -----
function affiche(tableau) {
    // affichage tableau
    for(i=0;i<tableau.length;i++){
        WScript.echo("tableau[" + i + "]= " + tableau[i]);
    }//for
} //function

// -----
function affiche2(tableau) {
    // affichage tableau
    for(i=0;i<tableau.length;i++){
        for(j=0;j<tableau[i].length;j++){
            WScript.echo("tableau[" + i + "," + j + "]= " + tableau[i][j]);
        }// for j
    }//for i
} //function

```

Ce programme utilise des tableaux. Appelons-le *tableaux.js* (le suffixe js désigne un fichier javascript). Positionnons-nous sur le répertoire dans lequel il se trouve et exécutons-le :

```

dos>cscript tableaux.js
Microsoft (R) Windows Script Host Version 5.6

```

```

tableau[0]=0
tableau[1]=10
tableau[2]=20
tableau[0]=0
tableau[1]=10
tableau[2]=20
tableau[3]=30
tableau[4]=40
tableau[5]=50

-----

tableau[0,0]=0
tableau[0,1]=1
tableau[0,2]=2
tableau[0,3]=3
tableau[1,0]=10
tableau[1,1]=11
tableau[1,2]=12
tableau[1,3]=13
tableau[2,0]=20
tableau[2,1]=21
tableau[2,2]=22
tableau[2,3]=23

```

Un dernier exemple en Perlscript pour terminer. Il faut avoir installé Active Perl pour avoir accès à Perlscript.

```

<job id="PERL1">

```



```

<script language="PerlScript"> # du
Perl classique
%dico=("maurice"=>"juliette", "philippe"=>"marianne"); @cles= keys
%dico;
for ($i=0;$i<=$#cles;$i++){
    $cle=$cles[$i];
    $valeur=$dico{$cle};
    $WScript->echo ("clé=".$cle.", valeur=".$valeur);
}
# du perlscript utilisant les objets Windows Script
$dico=$WScript->CreateObject("Scripting.Dictionary");
$dico->add("maurice","juliette");
$dico->add("philippe","marianne");
$WScript->echo($dico->item("maurice"));
$WScript->echo($dico->item("philippe"));
</script>
</job>

```

Ce programme montre la création et l'utilisation de deux dictionnaires : l'un à la mode Perl classique, l'autre avec l'objet *Scripting Dictionary* de Windows Script. Sauvegardons ce code dans le fichier *dico.wsf* (wsf est le suffixe des fichiers Windows Script). Positionnons-nous dans le dossier de ce programme et exécutons-le :

```

dos>cscript dico.wsf

clé=philippe, valeur=marianne
clé=maurice, valeur=juliette
juliette
marianne

```

Perlscript peut utiliser les objets du conteneur dans lequel il s'exécute. Ici c'était des objets du conteneur Windows Script. Dans le contexte de la programmation Web, les scripts VBscript, Javascript, Perlscript peuvent être exécutés soit au sein du navigateur IE, soit au sein d'un serveur PWS ou IIS. Si le script est un peu complexe, il peut être judicieux de le tester hors du contexte Web, au sein du conteneur Windows Script comme il a été vu précédemment. On ne pourra tester ainsi que les fonctions du script qui n'utilisent pas des objets propres au navigateur ou au serveur. Même avec cette restriction, cette possibilité reste intéressante car il est en général assez peu pratique de déboguer des scripts s'exécutant au sein des serveurs web ou des navigateurs.

JAVA

Java est disponible à l'URL : <http://www.sun.com> (cf début de ce document) et s'installe dans une arborescence qu'on appellera **<java>** qui contient les éléments suivants :

```

22/05/2002 05:51 <DIR> .
22/05/2002 05:51 <DIR> ..
22/05/2002 05:51 <DIR> bin
22/05/2002 05:51 <DIR> jre
07/02/2002 12:52      8 277 README.txt
07/02/2002 12:52     13 853 LICENSE
07/02/2002 12:52      4 516
07/02/2002 12:52     15 290 readme.html
22/05/2002 05:51 <DIR> lib
22/05/2002 05:51 <DIR> include
22/05/2002 05:51 <DIR> demo
07/02/2002 12:52     10 377 848 src.zip
11/02/2002 12:55 <DIR> docs

```

Dans **bin**, on trouvera **javac.exe**, le compilateur Java et **java.exe** la machine virtuelle Java. On pourra faire les tests suivants :

1. Écrire le script suivant :

```

//programme Java affichant l'heure

import java.io. ; *
import java.util. ; *

public class heure{
    public static void main(String arg[]){
        // on récupère date & heure Date
        maintenant=new Date();
        // on affiche
        System.out.println("Il est "+maintenant.getHours()+" : "+maintenant.getMinutes()+" : "+maintenant.getSeconds());
    }
}

```

```
}//main  
}//class
```

2. Sauvegarder ce programme sous le nom *heure.java*. Ouvrir une fenêtre DOS. Se mettre dans le répertoire du fichier *heure.java* et le compiler :

```
dos>c:\jdk1.3\bin\javac heure.java  
Note: heure.java uses or overrides a deprecated API.  
Note: Recompile with -deprecation for details.
```

Dans la commande ci-dessus `c:\jdk1.3\bin\javac` doit être remplacé par le chemin exact du compilateur *javac.exe*. Vous devez obtenir dans le même répertoire que *heure.java* un fichier *heure.class* qui est le programme qui va maintenant être exécuté par la machine virtuelle *java.exe*.

3. Exécuter le programme :

```
dos>c:\jdk1.3\bin\java heure  
Il est 10:44:2
```

Serveur Apache

Nous avons vu que l'on pouvait obtenir le serveur Apache avec l'application EasyPhp. Pour l'avoir directement, on ira sur le site d'Apache : <http://www.apache.org>. L'installation crée une arborescence où on trouve tous les fichiers nécessaires au serveur. Appelons **<apache>** ce répertoire. Il contient une arborescence analogue à la suivante :

```
UNINST  ISU      118 805 23/06/00 17:09 Uninst.isu  
HTDOCS  <REP>      23/06/00 17:09 htdocs  
APACHE~1 DLL    299 008 25/02/00 21:11 ApacheCore.dll  
ANNOUN~1      3 000 23/02/00 16:51 Announcement  
ABOUT ~1     13 197 31/03/99 18:42 ABOUT_APACHE  
APACHE  EXE     20 480 25/02/00 21:04 Apache.exe  
KEYS      36 437 20/08/99 11:57 KEYS  
LICENSE   2 907 01/01/99 13:04 LICENSE  
MAKEFI~1 TMP  27 370 11/01/00 13:47 Makefile.tmp  
README    2 109 01/04/98 6:59 README  
README NT   3 223 19/03/99 9:55 README.NT  
WARNIN~1 TXT  339 21/09/98 13:09 WARNING-NT.TXT  
BIN        <REP>      23/06/00 17:09 bin  
MODULES    <REP>      23/06/00 17:09 modules  
ICONS      <REP>      23/06/00 17:09 icons  
LOGS       <REP>      23/06/00 17:09 logs  
CONF       <REP>      23/06/00 17:09 conf  
CGI-BIN    <REP>      23/06/00 17:09 cgi-bin  
PROXY      <REP>      23/06/00 17:09 proxy  
INSTALL LOG  3 779 23/06/00 17:09 install.log
```

conf dossier des fichiers de configuration d'Apache
logs dossier des fichiers de logs (suivi) d'Apache
bin les exécutables d'Apache

Configuration


Dans le dossier **<Apache>\conf**, on trouve les fichiers suivants : *httpd.conf*, *srn.conf*, *access.conf*. Dans les dernières versions d'Apache, les trois fichiers ont été réunis dans *httpd.conf*. Nous avons déjà présenté les points importants de ce fichier de configuration. Dans les exemples qui suivent c'est la version Apache d'EasyPhp qui a servi aux tests et donc son fichier de configuration. Dans celui-ci *DocumentRoot* qui désigne la racine de l'arborescence des pages Web est *e:\program files\easyphp\www*.

Lien PHP - Apache

Pour tester, créer le fichier **intro.php** avec la seule ligne suivante :


```
<? phpinfo() ?>
```

et le mettre à la racine des pages du serveur Apache(*DocumentRoot* ci-dessus). Demander l'URL **http://localhost/intro.php**. On doit voir une liste d'informations *php* :

PHP Version 4.2.0

System	Windows NT 5.0 build 2195
Build Date	Apr 20 2002 18:36:03
Server API	Apache
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINNT\php.ini
Debug Build	no
Thread Safety	enabled

This program makes use of the Zend Scripting Language Engine:
Zend Engine v1.2.0, Copyright (c) 1998-2002 Zend Technologies



Le script PHP suivant affiche l'heure. Nous l'avons déjà rencontré :

```
<?
// time : nb de millisecondes depuis 01/01/1970
// "format affichage date-heure
// d: jour sur 2 chiffres
// m: mois sur 2 chiffres
// y : année sur 2 chiffres
// H : heure 0,23
// i : minutes
// s: secondes
print "Nous sommes le " . date("d/m/y H:i:s",time());
?>
```

Plaçons ce fichier texte à la racine des pages du serveur Apache (*DocumentRoot*) et appelons-le *date.php*. Demandons avec un navigateur l'URL **http://localhost/date.php**. On obtient la page suivante :

Address  http://localhost/date.php

Nous sommes le 18/07/02, 09:46:58

Lien PERL-APACHE

Il est fait grâce à une ligne de la forme : *ScriptAlias /cgi-bin/ "E:/Program Files/EasyPHP/cgi-bin/"* du fichier **<apache>\conf\httpd.conf**. Sa syntaxe est **ScriptAlias /cgi-bin/ "<cgi-bin>"** où **<cgi-bin>** est le dossier où on pourra placer des scripts CGI. CGI (Common Gateway Interface) est une norme de dialogue serveur WEB <--> Applications. Un client demande au serveur Web une page dynamique, c.a.d. une page générée par un programme. Le serveur WEB doit donc demander à un programme de générer la page. CGI définit le dialogue entre le serveur et le programme, notamment le mode de transmission des informations entre ces deux entités.

Si besoin est, modifiez la ligne *ScriptAlias /cgi-bin/ "<cgi-bin>"* et relancez le serveur Apache. Faites ensuite le test suivant :

1. Écrire le script :

```
#!c:\perl\bin\perl.exe
# script PERL affichant l'heure #

modules
use strict;

# programme
my ($secondes,$minutes,$heure)=localtime(time); print
<<FINHTML
Content-Type: text/html

<html>
<head>
<title>heure</title>
</head>
<body>
<h1>Il est $heure:$minutes:$secondes</h1>
</body>
FINHTML
;
```

2. Mettre ce script dans `<cgi-bin>\heure.pl` où `<cgi-bin>` est le dossier pouvant recevoir des scripts CGI (cf *httpd.conf*). La première ligne `#!c:\perl\bin\perl.exe` désigne le chemin de l'exécutable *perl.exe*. Le modifier si besoin est.
3. Lancer Apache si ce n'est fait
4. Demander avec un navigateur l'URL `http://localhost/cgi-bin/heure.pl`. On obtient la page suivante :



Il est 9:52:55

Le serveur PWS

7.9.1 Installation

Le serveur PWS (Personal Web Server) est une version personnelle du serveur IIS (Internet Information server) de Microsoft. Ce dernier est disponible sur les machines NT et 2000. Sur les machines win9x, PWS est normalement disponible avec le paquetage d'installation Internet Explorer. Cependant il n'est pas installé par défaut. Il faut prendre une installation personnalisée d'IE et demander l'installation de PWS. Il est par ailleurs disponibles dans le *NT 4.0 Option pack for Windows 95*.

Premiers tests

La racine des pages Web du serveur PWS est *lecteur:\inetpub\wwwroot* où *lecteur* est le disque sur lequel vous avez installé PWS. Nous supposons dans la suite que ce lecteur est D. Ainsi l'url `http://machine/repl/page1.html` correspondra au fichier `d:\inetpub\wwwroot\repl\page1.html`. Le serveur PWS interprète tout fichier de suffixe **.asp** (Active Server pages) comme étant un script qu'il doit exécuter pour produire une page HTML.

PWS travaille par défaut sur le port 80. Le serveur web Apache aussi... Il faut donc arrêter Apache pour travailler avec PWS si vous avez les deux serveurs. L'autre solution est de configurer Apache pour qu'il travaille sur un autre port. Ainsi dans le fichier *httpd.conf* de configuration d'Apache, on remplace la ligne **Port 80** par **Port 81**, Apache travaillera désormais sur le port 81 et pourra être utilisé en même temps que PWS. Si PWS ayant été lancé, on demande l'URL `http://localhost`, on obtient une page analogue à la suivante :



Bienvenue dans le Serveur Web personnel Microsoft® 4.0

Lien PHP - PWS

1. Ci-dessous on trouvera un fichier **.reg** destiné à modifier la base de registres. Double-cliquer sur ce fichier pour modifier la base. Ici la *dll* nécessaire se trouve dans *d:\php4* avec l'exécutable de php. Modifier si besoin est. Les \ doivent être doublés dans le chemin de la dll.

REGEDIT4

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="d:\php4\php4isapi.dll"
```

2. Relancer la machine pour que la modification de la base de registres soit prise en compte.
3. Créer un dossier **php** dans *d:\inetpub\wwwroot* qui est la racine du serveur PWS. Ceci fait, activez PWS et prendre l'onglet « Avancé ». Sélectionner le bouton « Ajouter » pour créer un dossier virtuel :

Répertoire/Parcourir : *d:\inetpub\wwwroot\php*
Alias : *php*
Cocher la case exécuter.

4. Valider le tout et relancer PWS. Mettre dans *d:\inetpub\wwwroot\php* le fichier **intro.php** ayant la seule ligne suivante :

<? phpinfo() ?>

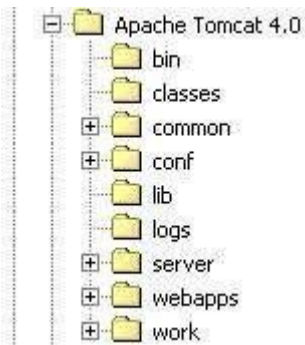
5. Demander au serveur PWS l'URL *http://localhost/php/intro.php*. On doit voir la liste d'informations php déjà présentées avec Apache.

Tomcat : servlets Java et pages JSP (Java Server Pages)

Tomcat est un serveur Web permettant de générer des pages HTML grâce à des servlets (programmes Java exécutés par le serveur web) où des pages JSP (Java Server Pages), pages mélangeant code Java et code HTML. C'est l'équivalent des pages ASP (Active Server Pages) du serveur IIS/PWS de Microsoft où là on mélange code VBScript ou Javascript avec du code HTML.

Installation

Tomcat est disponible à l'URL : **<http://jakarta.apache.org>**. On récupère un fichier .exe d'installation. Lorsqu'on lance ce programme, il commence par indiquer quel JDK il va utiliser. En effet Tomcat a besoin d'un JDK pour s'installer et ensuite compiler et exécuter les servlets Java. Il faut donc que vous ayez installé un JDK Java avant d'installer Tomcat. Le JD le plus récent est conseillé. L'installation va créer une arborescence <tomcat> :



consiste simplement à décompresser cette archive dans un répertoire. Prenez un répertoire ne contenant dans son chemin que des noms sans espace (pas par exemple "Program Files"), ceci parce qu'il y a un bogue dans le processus d'installation de Tomcat. Prenez par exemple C:\tomcat ou D:\tomcat. Appelons ce répertoire **<tomcat>**. On y trouvera dedans un dossier appelé **jakarta-tomcat** et dans celui-ci l'arborescence suivante :

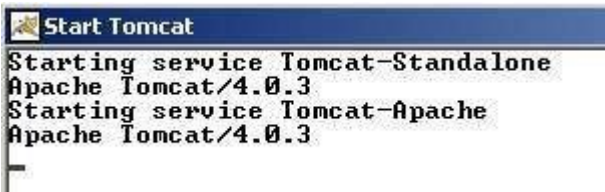
LOGS	<REP>	15/11/00	9:04	logs
LICENSE		2 876 18/04/00	15:56	LICENSE
CONF	<REP>	15/11/00	8:53	conf
DOC	<REP>	15/11/00	8:53	doc
LIB	<REP>	15/11/00	8:53	lib
SRC	<REP>	15/11/00	8:53	src
WEBAPPS	<REP>	15/11/00	8:53	webapps
BIN	<REP>	15/11/00	8:53	bin
WORK	<REP>	15/11/00	9:04	work

Démarrage/Arrêt du serveur Web Tomcat


Tomcat est un serveur Web comme l'est Apache ou PWS. Pour le lancer, on dispose de liens dans le menu des programmes :

Start Tomcat pour lancer Tomcat
 Stop Tomcat pour l'arrêter

Lorsqu'on lance Tomcat, une fenêtre Dos s'affiche avec le contenu suivant :



On peut mettre cette fenêtre Dos en icône. Elle restera présente pendant tant que Tomcat sera actif. On peut alors passer aux premiers tests. Le serveur Web Tomcat travaille sur le port 8080. Une fois Tomcat lancé, prenez un navigateur Web et demandez l'URL **http://localhost:8080**. Vous devez obtenir la page suivante :

Address  http://localhost:8080/index.html



Tomcat
Version 4.0.3



If you're seeing

As you may have gue
filesystem at:

Suivez le lien **Servlet Examples** :

Hello World

Request Info

Request Headers


Request Parameters


Cookies

 [Execute](#)

 [Execute](#)

 [Execute](#)


 [Execute](#)

 [Execute](#)

 [Source](#)

 [Source](#)

 [Source](#)

 [Source](#)

 [Source](#)

Cliquez sur le lien *Execute* de *RequestParameters* puis sur celui de *Source*. Vous aurez un premier aperçu de ce qu'est une servlet Java. Vous pourrez faire de même avec les liens sur les pages JSP.

Pour arrêter Tomcat, on utilisera le lien *Stop Tomcat* dans le menu des programmes.

Jbuilder

Jbuilder est un environnement de développement d'applications Java. Pour construire des servlets Java où il n'y a pas d'interfaces graphiques, il n'est pas indispensable d'avoir un tel environnement. Un éditeur de textes et un JDK font l'affaire. Seulement JBuilder apporte avec lui quelques plus par rapport à la technique précédente :

- facilité de débogage : le compilateur signale les lignes erronées d'un programme et il est facile de s'y positionner
- suggestion de code : lorsqu'on utilise un objet Java, JBuilder donne en ligne la liste des propriétés et méthodes de celui-ci. Cela est très pratique lorsqu'on sait que la plupart des objets Java ont de très nombreuses propriétés et méthodes qu'il est difficile de se rappeler.

On trouvera JBuilder sur le site <http://www.borland.com/jbuilder>. Il faut remplir un formulaire pour obtenir le logiciel. Une clé d'activation est envoyée par mél. Pour installer JBuilder 7, il a par exemple été procédé ainsi :

- trois fichiers zip ont été obtenus : pour l'application, pour la documentation, pour les exemples. Chacun de ces zip fait l'objet d'un lien séparé sur le site de JBuilder.
- on a installé d'abord l'application, puis la documentation et enfin les exemples
- lorsqu'on lance l'application la première fois, une clé d'activation est demandée : c'est celle qui vous a été envoyée par mél. Dans la version 7, cette clé est en fait la totalité d'un fichier texte que l'on peut placer, par exemple, dans le dossier

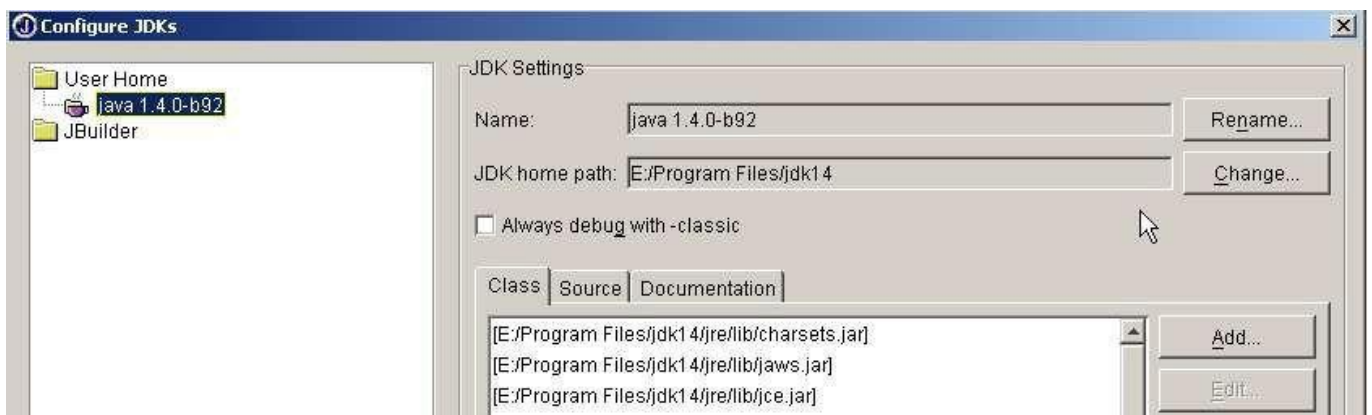
d'installation de JB7. Au moment où la clé est demandée, on désigne alors le fichier en question. Ceci fait, la clé ne sera plus redemandée.

Il y a quelques configurations utiles à faire si on veut utiliser JBuilder pour construire des servlets Java. En effet, la version dite Jbuilder personnel est une version allégée qui ne vient notamment pas avec toutes les classes nécessaires pour faire du développement web en Java. On peut faire en sorte que JBuilder utilise les bibliothèques de classes amenées par Tomcat. On procède ainsi :

- lancer JBuilder



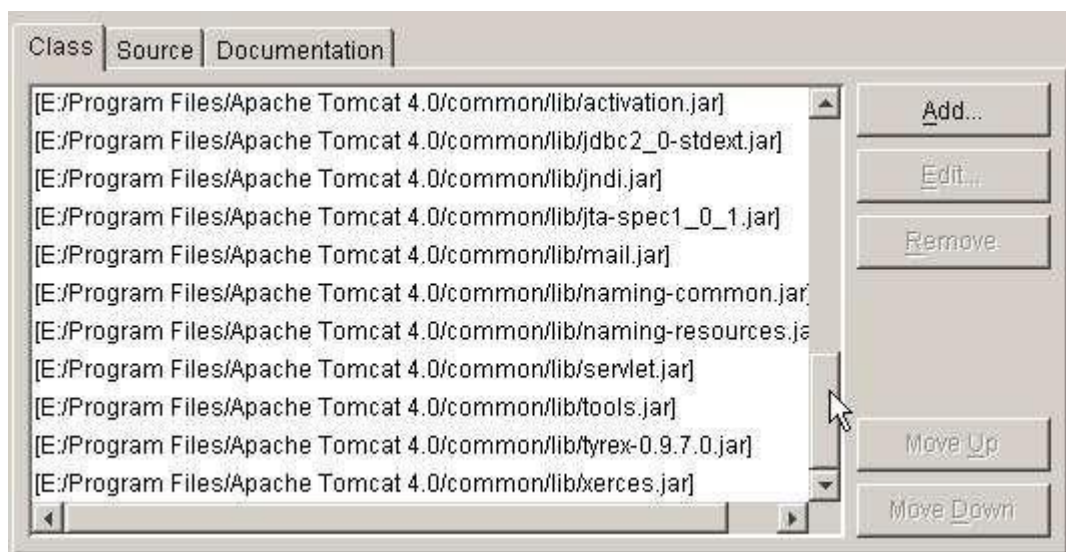
- activer l'option *Tools/Configure JDKs*



Dans la partie *JDK Settings* ci-dessus, on a normalement dans le champ *Name* un JDK 1.3.1. Si vous avez un JDK plus récent, utilisez le bouton *Change* pour désigner le répertoire d'installation de ce dernier. Ci-dessus, on a désigné le répertoire *E:\Program Files\jdk14* où avait installé un JDK 1.4. Désormais, JBuilder utilisera ce JDK pour ses compilations et exécutions. Dans la partie (Class, Source, Documentation) on a la liste de toutes les bibliothèques de classes qui seront explorées par JBuilder, ici les classes du JDK 1.4. Les classes de celui-ci ne suffisent pas pour faire du développement web en Java. Pour ajouter d'autres bibliothèques de classes on utilise le bouton *Add* et on désigne les fichiers *.jar* supplémentaires que l'on veut utiliser. Les fichiers *.jar* sont des bibliothèques de classes. Tomcat 4.x amène avec lui toutes les bibliothèques de classes nécessaires au développement web. Elles se trouvent dans `<tomcat>\common\lib` où `<tomcat>` est le répertoire d'installation de Tomcat :



Avec le bouton *Add*, on va ajouter ces bibliothèques, une à une, à la liste des bibliothèques explorées par JBuilder :



A partir de maintenant, on peut compiler des programmes java conformes à la norme J2EE, notamment les servlets Java. Jbuilder ne sert qu'à la compilation, l'exécution étant ultérieurement assurée par Tomcat selon des modalités expliquées dans le cours.

8. Code source de programmes

8.1 Le client TCP générique

Beaucoup de services créés à l'origine de l'Internet fonctionnent selon le modèle du serveur d'écho étudié précédemment : les échanges client-serveur se font pas échanges de lignes de texte. Nous allons écrire un client tcp générique qui sera lancé de la façon suivante : **java cltTCPgenerique serveur port**

Ce client TCP se connectera sur le port *port* du serveur *serveur*. Ceci fait, il créera deux threads :

1. un thread chargé de lire des commandes tapées au clavier et de les envoyer au serveur
2. un thread chargé de lire les réponses du serveur et de les afficher à l'écran

Pourquoi deux threads? Chaque service TCP-IP a son protocole particulier et on trouve parfois les situations suivantes :

- le client doit envoyer plusieurs lignes de texte avant d'avoir une réponse
- la réponse d'un serveur peut comporter plusieurs lignes de texte

Aussi la boucle envoi d'une unique ligne au serveur - réception d'une unique ligne envoyée par le serveur ne convient-elle pas toujours. On va donc créer deux boucles dissociées :

- une boucle de lecture des commandes tapées au clavier pour être envoyées au serveur. L'utilisateur signalera la fin des commandes avec le mot clé *fin*.
- une boucle de réception et d'affichage des réponses du serveur. Celle-ci sera une boucle infinie qui ne sera interrompue que par la fermeture du flux réseau par le serveur ou par l'utilisateur au clavier qui tapera la commande *fin*.

Pour avoir ces deux boucles dissociées, il nous faut deux threads indépendants. Montrons un exemple d'exécution où notre client tcp générique se connecte à un service SMTP (SendMail Transfer Protocol). Ce service est responsable de l'acheminement du courrier électronique à leurs destinataires. Il fonctionne sur le port 25 et a un protocole de dialogue de type échanges de lignes de texte.

```
Dos>java clientTCPgenerique istia.univ-angers.fr 25
Commandes :
<-- 220 istia.univ-angers.fr ESMTP Sendmail 8.11.6/8.9.3; Mon, 13 May 2002 08:37:26 +0200
help
<-- 502 5.3.0 Sendmail 8.11.6 -- HELP not implemented
mail from: machin@univ-angers.fr
<-- 250 2.1.0 machin@univ-angers.fr... Sender ok
rcpt to: serge.tahe@istia.univ-angers.fr
<-- 250 2.1.5 serge.tahe@istia.univ-angers.fr... Recipientok
data
<-- 354 Enter mail, end with "." on a line by itself
Subject: test

ligne1
ligne2
ligne3
.
<-- 250 2.0.0 g4D6bks25951 Message accepted for delivery
quit
<-- 221 2.0.0 istia.univ-angers.fr closingconnection
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]
```

Commentons ces échanges client-serveur :

- le service SMTP envoie un message de bienvenue lorsqu'un client se connecte à lui :

```
<-- 220 istia.univ-angers.fr ESMTP Sendmail 8.11.6/8.9.3; Mon, 13 May 2002 08:37:26 +0200
```

- certains services ont une commande *help* donnant des indications sur les commandes utilisables avec le service. Ici ce n'est pas le cas. Les commandes SMTP utilisées dans l'exemple sont les suivantes :
 - **mail from:** *expéditeur*, pour indiquer l'adresse électronique de l'expéditeur du message
 - **rcpt to:** *destinataire*, pour indiquer l'adresse électronique du destinataire du message. S'il y a plusieurs destinataires, on ré-émet autant de fois que nécessaire la commande **rcpt to:** pour chacun des destinataires.
 - **data** qui signale au serveur SMTP qu'on va envoyer le message. Comme indiqué dans la réponse du serveur, celui-ci est une suite de lignes terminée par une ligne contenant le seul caractère point. Un message peut avoir des entêtes séparés du corps du message par une ligne vide. Dans notre exemple, nous avons mis un sujet avec le mot clé **Subject:**
- une fois le message envoyé, on peut indiquer au serveur qu'on a terminé avec la commande **quit**. Le serveur ferme alors la connexion réseau. Le thread de lecture peut détecter cet événement et s'arrêter.
- l'utilisateur tape alors **fin** au clavier pour arrêter également le thread de lecture des commandes tapées au clavier.

Si on vérifie le courrier reçu, nous avons la chose suivante (Outlook) :

From: machin@univ-angers.fr **To:**
Subject: test

ligne1
ligne2
ligne3

On remarquera que le service SMTP ne peut détecter si un expéditeur est valide ou non. Aussi ne peut-on jamais faire confiance au champ *from* d'un message. Ici l'expéditeur *machin@univ-angers.fr* n'existait pas.

Ce client tcp générique peut nous permettre de découvrir le protocole de dialogue de services internet et à partir de là construire des classes spécialisées pour des clients de ces services. Découvrons le protocole de dialogue du service POP (Post Office Protocol) qui permet de retrouver ses méls stockés sur un serveur. Il travaille sur le port 110.

```
Dos> java clientTCPgenerique istia.univ-angers.fr 110
Commandes :
<-- +OK Qpopper (version 4.0.3) at istia.univ-angers.fr starting.
help
<-- -ERR Unknown command: "help".
user st
<-- +OK Password required for st.
pass monpassword
<-- +OK st has 157 visible messages (0 hidden) in 11755927 octets.
list
<-- +OK 157 visible messages (11755927 octets)
<-- 1 892847
<-- 2 171661
...
<-- 156 2843
<-- 157 2796
<-- .
retr 157
<-- +OK 2796 octets
<-- Received: from lagaffe.univ-angers.fr (lagaffe.univ-angers.fr [193.49.144.1])
<--      by istia.univ-angers.fr (8.11.6/8.9.3) with ESMTTP id g4D6wZs26600;
<--      Mon, 13 May 2002 08:58:35 +0200
<-- Received: from jaume ([193.49.146.242])
<--      by lagaffe.univ-angers.fr (8.11.1/8.11.2/GeO20000215) with SMTP id g4D6wSd37691;
<--      Mon, 13 May 2002 08:58:28 +0200 (CEST)
...
<-- -----
<-- NOC-RENATER2                Tl. : 0800 77 47 95
<-- Fax : (+33) 01 40 78 64 00 , Email : noc-r2@cssi.renater.fr
<-- -----
<-- .
quit
<-- +OK Pop server at istia.univ-angers.fr signing off.
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]
```

Les principales commandes sont les suivantes :

- **user login**, où on donne son login sur la machine qui détient nos méls
- **pass password**, où on donne le mot de passe associé au login précédent
- **list**, pour avoir la liste des messages sous la forme numéro, taille en octets
- **retr i**, pour lire le message n° i
- **quit**, pour arrêter le dialogue.

Découvrons maintenant le protocole de dialogue entre un client et un serveur Web qui lui travaille habituellement sur le port 80 :

```
Dos> java clientTCPgenerique istia.univ-angers.fr 80
Commandes :
GET /index.html HTTP/1.0
```

```

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>
<--
<-- <head>
<-- <meta http-equiv="Content-Type"
<-- content="text/html; charset=iso-8859-1">
<-- <meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<-- <title>Bienvenue a l'ISTIA - Universite d'Angers</title>
<-- </head>
<--
<--
<-- face="Verdana"> - Dernire mise jour le <b>10 janvier 2002</b></font></p>
<-- </body>
<-- </html>
<--
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]

```

Un client Web envoie ses commandes au serveur selon le schéma suivant :

```

commande1
commande2
...
commanden
[ligne vide]

```

Ce n'est qu'après avoir reçu la ligne vide que le serveur Web répond. Dans l'exemple nous n'avons utilisé qu'une commande :

```
GET /index.html HTTP/1.0
```

qui demande au serveur l'URL **/index.html** et indique qu'il travaille avec le protocole HTTP version 1.0. La version la plus récente de ce protocole est 1.1. L'exemple montre que le serveur a répondu en renvoyant le contenu du fichier *index.html* puis qu'il a fermé la connexion puisqu'on voit le thread de lecture des réponses se terminer. Avant d'envoyer le contenu du fichier *index.html*, le serveur web a envoyé une série d'entêtes terminée par une ligne vide :

```

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>

```

La ligne **<html>** est la première ligne du fichier */index.html*. Ce qui précède s'appelle des entêtes HTTP (HyperText Transfer Protocol). Nous n'allons pas détailler ici ces entêtes mais on se rappellera que notre client générique y donne accès, ce qui peut être utile pour les comprendre. La première ligne par exemple :

```
<-- HTTP/1.1 200 OK
```

indique que le serveur Web contacté comprend le protocole HTTP/1.1 et qu'il a bien trouvé le fichier demandé (200 OK), 200 étant un code de réponse HTTP. Les lignes

```

<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html

```


disent au client qu'il va recevoir 11251 octets représentant du texte HTML (HyperText Markup Language) et qu'à la fin de l'envoi, la connexion sera fermée.

On a donc là un client tcp très pratique. Il fait sans doute moins que le programme *telnet* que nous avons utilisé précédemment mais il était intéressant de l'écrire nous-mêmes. Le programme du client tcp générique est le suivant :

```
// paquetages importés
import java.io. ; import
java.net. ;

public class clientTCPgenerique{

    // reçoit en paramètre les caractéristiques d'un service sous la forme
    // serveur port
    // se connecte au service
    // crée un thread pour lire des commandes tapées au clavier
    // celles-ci seront envoyées au serveur
    // crée un thread pour lire les réponses du serveur
    // celles-ci seront affichées à l'écran
    // le tout se termine avec la commande fin tapée au clavier

    // variable d'instance private
    static Socket client;

    public static void main(String[] args){

        // syntaxe
        final String syntaxe="pg serveur port";

        // nombre d'arguments
        if(args.length != 2)
            erreur(syntaxe,1);

        // on note le nom du serveur
        String serveur=args[0];

        // le port doit être entier >0 int
        port=0;
        boolean erreurPort=false;
        Exception E=null;
        try{
            port=Integer.parseInt(args[1]);
        }catch(Exception e){
            E=e;
            erreurPort=true;
        }
        erreurPort=erreurPort || port <=0;
        if(erreurPort)
            erreur(syntaxe+"%n"+"Port incorrect ("+"E+""),2);

        client=null;
        // il peut y avoir des problèmes try{
        // on se connecte au service client=new
        Socket(serveur,port);
        }catch(Exception ex){
            // erreur
            erreur("Impossible de se connecter au service ("+" serveur
            +","+"port+""), erreur : "+ex.getMessage(),3);

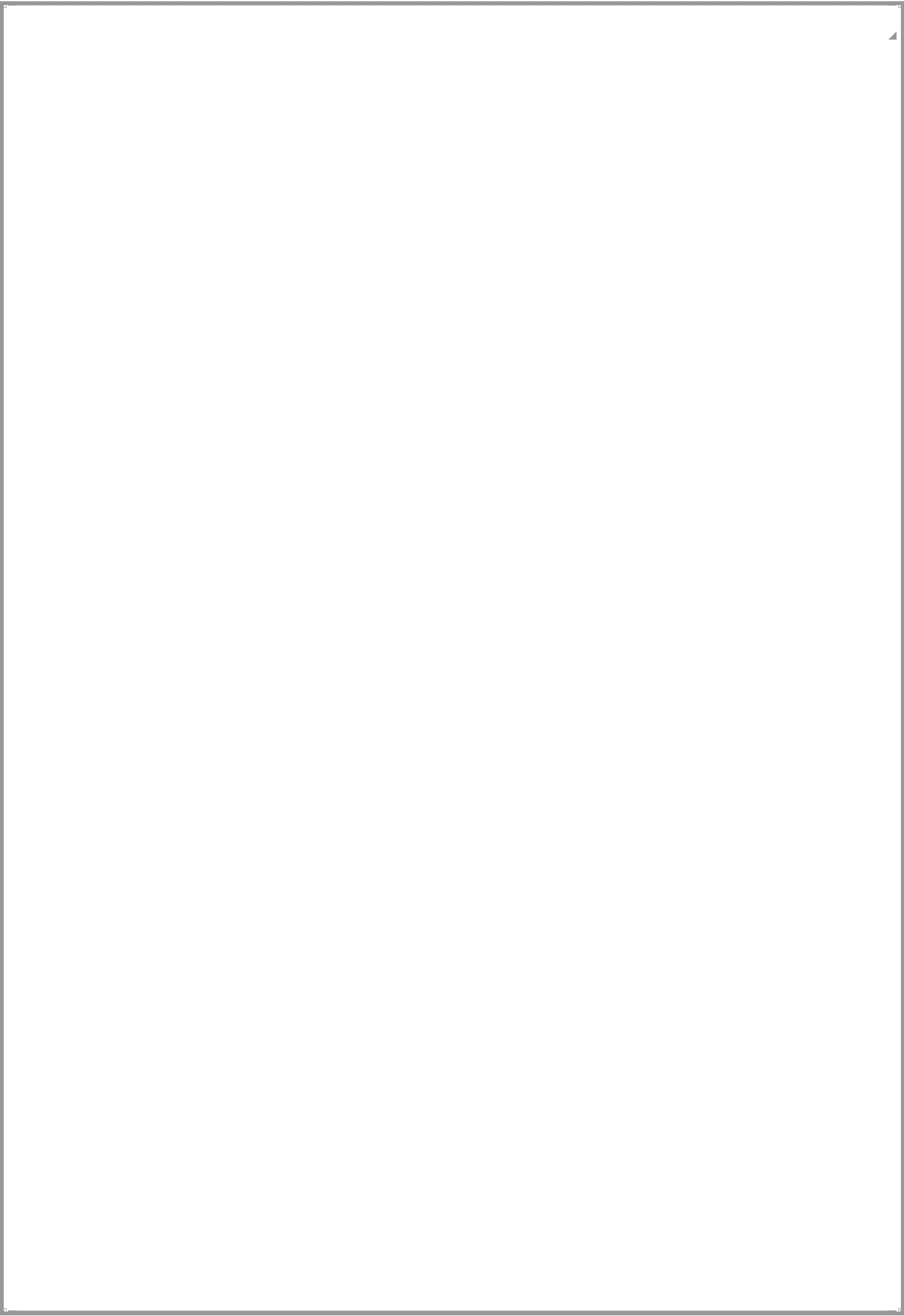
            // fin
            return;
        }//catch

        // on crée les threads de lecture/écriture new
        ClientSend(client).start();
        new ClientReceive(client).start();

        // fin thread main
        return;
    }// main

    // affichage des erreurs
    public static void erreur(String msg, int exitCode){
        // affichage erreur
        System.err.println(msg);
        // arrêt avec erreur
        System.exit(exitCode);
    }//erreur
}//classe

class ClientSend extends Thread {
    // classe chargée de lire des commandes tapées au clavier
    // et de les envoyer à un serveur via un client tcp passé en paramètre
```

```

private Socket client; // le client tcp

// constructeur
public ClientSend(Socket client){
    // on note le client tcp
    this.client=client;
} //constructeur

// méthode Run du thread public
void run(){

    // données locales
    PrintWriter OUT=null;          // flux d'écriture réseau
    BufferedReader IN=null;         // flux clavier
    String commande=null;           // commande lue au clavier

    // gestion des erreurs
    try{
        // création du flux d'écriture réseau
        OUT=new PrintWriter(client.getOutputStream(),true);
        // création du flux d'entrée clavier
        IN=new BufferedReader(new InputStreamReader(System.in));
        // boucle saisie-envoi des commandes
        System.out.println("Commandes : ");
        while(true){
            // lecture commande tapée au clavier
            commande=IN.readLine().trim();
            // fini ?
            if (commande.toLowerCase().equals("fin")) break;
            // envoi commande au serveur
            OUT.println(commande);
            // commande suivante
        } //while
    } catch (Exception ex) {
        // erreur
        System.err.println("Envoi : L'erreur suivante s'est produite : " + ex.getMessage());
    } //catch
    // fin - on ferme les flux try{
    OUT.close(); client.close();
    } catch (Exception ex) {}
    // on signale la fin du thread
    System.out.println("[Envoi : fin du thread d'envoi des commandes au serveur]");
} //run
} //classe

class ClientReceive extends Thread{
    // classe chargée de lire les lignes de texte destinées à un
    // client tcp passé en paramètre private

    Socket client; // le client tcp

    // constructeur
    public ClientReceive(Socket client){
        // on note le client tcp
        this.client=client;
    } //constructeur

    // méthode Run du thread public
    void run(){

        // données locales
        BufferedReader IN=null; // flux lecture réseau String
        réponse=null;           // réponse serveur

        // gestion des erreurs
        try{
            // création du flux lecture réseau
            IN=new BufferedReader(new InputStreamReader(client.getInputStream()));
            // boucle lecture lignes de texte du flux IN while(true){
            // lecture flux réseau
            réponse=IN.readLine();
            // flux fermé ?
            if(réponse==null) break;
            // affichage
            System.out.println("<-- "+réponse);
        } //while
    } catch (Exception ex) {
        // erreur
        System.err.println("Réception : L'erreur suivante s'est produite : " + ex.getMessage());
    } //catch
    // fin - on ferme les flux try{
    IN.close(); client.close();
    } catch (Exception ex) {}
}

```

```
// on signale la fin du thread
System.out.println("[Réception : fin du thread de lecture des réponses du serveur]");
} //run
} //classe
```

Le serveurTcpgénérique

Maintenant nous nous intéressons à un serveur

- qui affiche à l'écran les commandes envoyées par ses clients
- leur envoie comme réponse les lignes de texte tapées au clavier par un utilisateur. C'est donc ce dernier qui fait office de serveur.

Le programme est lancé par : **java serveurTCPgenerique portEcoule**, où *portEcoule* est le port sur lequel les clients doivent se connecter. Le service au client sera assuré par deux threads :

- un thread se consacrant exclusivement à la lecture des lignes de texte envoyées par le client
- un thread se consacrant exclusivement à la lecture des réponses tapées au clavier par l'utilisateur. Celui-ci signalera par la commande **fin** qu'il clôt la connexion avec le client.

Le serveur crée deux threads par client. S'il y a n clients, il y aura $2n$ threads actifs en même temps. Le serveur lui ne s'arrête jamais sauf par un Ctrl-C tapé au clavier par l'utilisateur. Voyons quelques exemples.

Le serveur est lancé sur le port 100 et on utilise le client générique pour lui parler. La fenêtre du client est la suivante :

```
dos> java clientTCPgenerique localhost 100
Commandes :
commande 1 du client 1
<-- réponse 1 au client 1
commande 2 du client 1
<-- réponse 2 au client 1
fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du thread de lecture des réponses du serveur]
[fin du thread d'envoi des commandes au serveur]
```

Les lignes commençant par <-- sont celles envoyées du serveur au client, les autres celles du client vers le serveur. La fenêtre du serveur est la suivante :

```
Dos> java serveurTCPgenerique 100
Serveur générique lancé sur le port 100
Thread de lecture des réponses du serveur au client 1 lancé
1 : Thread de lecture des demandes du client 1 lancé
<-- commande 1 du client 1
réponse 1 au client 1
1 : <-- commande 2 du client 1
réponse 2 au client 1
1 : [fin du Thread de lecture des demandes du client 1]
fin
[fin du Thread de lecture des réponses du serveur au client 1]
```

Les lignes commençant par <-- sont celles envoyées du client au serveur. Les lignes *N* : sont les lignes envoyées du serveur au client n° *N*. Le serveur ci-dessus est encore actif alors que le client 1 est terminé. On lance un second client pour le même serveur :

```
Dos> java clientTCPgenerique localhost 100
Commandes :
commande 3 du client 2
<-- réponse 3 au client 2
fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du thread de lecture des réponses du serveur]
[fin du thread d'envoi des commandes au serveur]
```

La fenêtre du serveur est alors celle-ci :

```
Dos> java serveurTCPgenerique 100
Serveur générique lancé sur le port 100
Thread de lecture des réponses du serveur au client 1 lancé
```

```

1 : Thread de lecture des demandes du client 1 lancé
<-- commande 1 du client 1
réponse 1 au client 1
1 : <-- commande 2 du client 1
réponse 2 au client 1
1 : [fin du Thread de lecture des demandes du client 1]
fin
[fin du Thread de lecture des réponses du serveur au client 1]
Thread de lecture des réponses du serveur au client 2 lancé
2 : Thread de lecture des demandes du client 2 lancé
<-- commande 3 du client 2
réponse 3 au client 2
2 : [fin du Thread de lecture des demandes du client 2]
fin
[fin du Thread de lecture des réponses du serveur au client 2]
^C

```

Simulons maintenant un serveur web en lançant notre serveur générique sur le port 88 :

```

Dos> java serveurTCPgenerique 88
Serveur générique lancé sur le port 88

```

Prenons maintenant un navigateur et demandons l'URL `http://localhost:88/exemple.html`. Le navigateur va alors se connecter sur le port 88 de la machine `localhost` puis demander la page `/exemple.html` :



Regardons maintenant la fenêtre de notre serveur :

```

Dos>java serveurTCPgenerique 88
Serveur générique lancé sur le port 88
Thread de lecture des réponses du serveur au client 2 lancé
2 : Thread de lecture des demandes du client 2 lancé
<-- GET /exemple.html HTTP/1.1
<-- Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, */*
<-- Accept-Language: fr
<-- Accept-Encoding: gzip, deflate
<-- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705; .NET CLR 1.0.2
914)
<-- Host: localhost:88
<-- Connection: Keep-Alive
<--

```

On découvre ainsi les entêtes HTTP envoyés par le navigateur. Cela nous permet de découvrir peu à peu le protocole HTTP. Lors d'un précédent exemple, nous avons créé un client Web qui n'envoyait que la seule commande GET. Cela avait été suffisant. On voit ici que le navigateur envoie d'autres informations au serveur. Elles ont pour but d'indiquer au serveur quel type de client il a en face de lui. On voit aussi que les entêtes HTTP se terminent par une ligne vide.

Elaborons une réponse à notre client. L'utilisateur au clavier est ici le véritable serveur et il peut élaborer une réponse à la main. Rappelons-nous la réponse faite par un serveur Web dans un précédent exemple :

```

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>

```

Essayons de donner une réponse analogue :

```
...
<-- Host: localhost:88
<-- Connection: Keep-Alive
<--
2 : HTTP/1.1 200 OK
2 : Server: serveur tcp generique
2 : Connection: close
2 : Content-Type: text/html
2 :
2 : <html>
2 :   <head><title>Serveur generique</title></head>
2 :   <body>
2 :     <center>
2 :       <h2>Reponse du serveur generique</h2>
2 :     </center>
2 :   </body>
2 : </html>
2 : fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du Thread de lecture des demandes du client 2]
[fin du Thread de lecture des réponses du serveur au client 2]
```

Les lignes commençant par 2 : sont envoyées du serveur au client n° 2. La commande *fin* clôt la connexion du serveur au client. Nous nous sommes limités dans notre réponse aux entêtes HTTP suivants :

```
HTTP/1.1 200 OK
2 : Server: serveur tcp generique
2 : Connection: close
2 : Content-Type: text/html
2 :
```

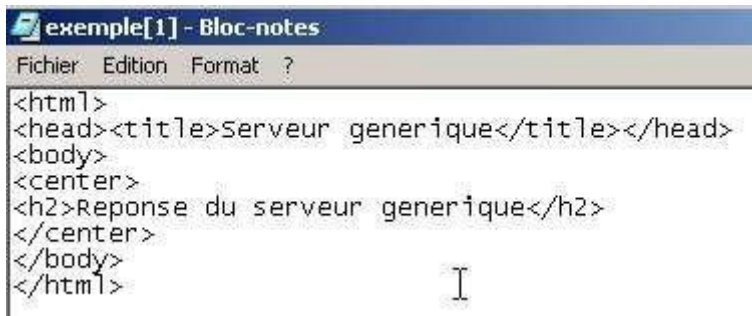
Nous ne donnons pas la taille du fichier que nous allons envoyer (*Content-Length*) mais nous contentons de dire que nous allons fermer la connexion (*Connection: close*) après envoi de celui-ci. Cela est suffisant pour le navigateur. En voyant la connexion fermée, il saura que la réponse du serveur est terminée et affichera la page HTML qui lui a été envoyée. Cette dernière est la suivante :

```
2 : <html>
2 :   <head><title>Serveur generique</title></head>
2 :   <body>
2 :     <center>
2 :       <h2>Reponse du serveur generique</h2>
2 :     </center>
2 :   </body>
2 : </html>
```

L'utilisateur ferme ensuite la connexion au client en tapant la commande *fin*. Le navigateur sait alors que la réponse du serveur est terminée et peut alors l'afficher :



Si ci-dessus, on fait *View/Source* pour voir ce qu'a reçu le navigateur, on obtient :



```
<html>
<head><title>Serveur generique</title></head>
<body>
<center>
<h2>Reponse du serveur generique</h2>
</center>
</body>
</html>
```

c'est à dire exactement ce qu'on a envoyé depuis le serveur générique.

Le code du serveur tcp générique est le suivant:

```
// paquetages import
java.io. ; import *
java.net. ;

public class serveurTCPgenerique{

    // programme principal
    public static void main (String[] args){

        // reçoit le port d'écoute des demandes des clients
        // crée un thread pour lire les demandes du client
        // celles-ci seront affichées à l'écran
        // crée un thread pour lire des commandes tapées au clavier
        // celles-ci seront envoyées comme réponse au client
        // le tout se termine avec la commande fin tapée au clavier

        final String syntaxe="Syntaxe : pg port";
        // variable d'instance
        // y-a-t-il un argument
        if(args.length != 1)
            erreur(syntaxe,1);

        // le port doit être entier >0 int
        port=0;
        boolean erreurPort=false;
        Exception E=null;
        try{
            port=Integer.parseInt(args[0]);
        }catch(Exception e){
            E=e;
            erreurPort=true;
        }
        erreurPort=erreurPort || port <=0;
        if(erreurPort)
            erreur(syntaxe+"%n"+"Port incorrect (" +E+"")",2);

        // on crée le service d'écoute
        ServerSocket ecoute=null;
        int nbClients=0; // nbre de clients traités try{
            // on crée le service ecoute=new
            ServerSocket(port);
            // suivi
            System.out.println("Serveur générique lancé sur le port " + port);

            // boucle de service aux clients Socket
            client=null;
            while (true){ // boucle infinie - sera arrêtée par Ctrl-C
                // attente d'un client
                client=ecoute.accept();

                // le service est assuré des threads séparés nbClients++;

                // on crée les threads de lecture/écriture new
                ServeurSend(client,nbClients).start(); new
                ServeurReceive(client,nbClients).start();

                // on retourne à l'écoute des demandes
            } // fin while
        } catch(Exception ex){
            // on signale l'erreur
            erreur("L'erreur suivante s'est produite : " + ex.getMessage(),3);
        } // catch
    } // fin main
```

```

// affichage des erreurs
public static void erreur(String msg, int exitCode){
    // affichage erreur
    System.err.println(msg);
    // arrêt avec erreur
    System.exit(exitCode);
} //erreur
} //classe

class ServeurSend extends Thread{
    // classe chargée de lire des réponses tapées au clavier
    // et de les envoyer à un client via un client tcp passé au constructeur

    Socket client; // le client tcp int
    numClient;      // n° de client

    // constructeur
    public ServeurSend(Socket client, int numClient){
        // on note le client tcp
        this.client=client;
        // et son n°
        this.numClient=numClient;
    } //constructeur

    // méthode Run du thread public
    void run() {

        // données locales
        PrintWriter OUT=null;      // flux d'écriture réseau
        String réponse=null;        // réponse lue au clavier
        BufferedReader IN=null;    // flux clavier

        // suivi
        System.out.println("Thread de lecture des réponses du serveur au client "+ numClient + " lancé");
        // gestion des erreurs
        try{
            // création du flux d'écriture réseau
            OUT=new PrintWriter(client.getOutputStream(),true);
            // création du flux clavier
            IN=new BufferedReader(new InputStreamReader(System.in));
            // boucle saisie-envoi des commandes
            while(true){
                // identification client
                System.out.print("--> " + numClient + " : ");
                // lecture réponse tapée au clavier
                réponse=IN.readLine().trim();
                // fini ?
                if (réponse.toLowerCase().equals("fin")) break;
                // envoi réponse au serveur
                OUT.println(réponse);
                // réponse suivante
            } //while
        } catch (Exception ex) {
            // erreur
            System.err.println("L'erreur suivante s'est produite : " + ex.getMessage());
        } //catch
        // fin - on ferme les flux try{
        OUT.close(); client.close();
        } catch (Exception ex) {}
        // on signale la fin du thread
        System.out.println("[fin du Thread de lecture des réponses du serveur au client "+ numClient+"]");
    } //run
} //classe

class ServeurReceive extends Thread{
    // classe chargée de lire les lignes de texte envoyées au serveur
    // via un client tcp passé au constructeur

    Socket client; // le client tcp int
    numClient;      // n° de client

    // constructeur
    public ServeurReceive(Socket client, int numClient){
        // on note le client tcp
        this.client=client;
        // et son n°
        this.numClient=numClient;
    } //constructeur

    // méthode Run du thread public
    void run() {

        // données locales
        BufferedReader IN=null;      // flux lecture réseau

```



```

String réponse=null;          // réponse serveur

// suivi
System.out.println("Thread de lecture des demandes du client "+ numClient + " lancé");
// gestion des erreurs
try{
    // création du flux lecture réseau
    IN=new BufferedReader(new InputStreamReader(client.getInputStream()));
    // boucle lecture lignes de texte du flux IN while(true){
    // lecture flux réseau
    réponse=IN.readLine();
    // flux fermé ?
    if(réponse==null) break;
    // affichage
    System.out.println("<-- "+réponse);
    }//while
}catch(Exception ex){
    // erreur
    System.err.println("L'erreur suivante s'est produite : " + ex.getMessage());
}//catch
// fin - on ferme les flux try{
    IN.close();client.close();
}catch(Exception ex){}
// on signale la fin du thread
System.out.println("[fin du Thread de lecture des demandes du client "+ numClient+"]");
} //run
} //classe

```

9. JAVASCRIPT

Nous montrons dans cette partie trois exemples d'utilisation de Javascript dans les pages WEB. Nous nous centrons sur la gestion des formulaires mais Javascript peut faire bien davantage.

Récupérer les informations d'un formulaire

L'exemple ci-dessous montre comment récupérer au sein du navigateur les données entrées par l'utilisateur au sein d'un formulaire. Cela permet en général de faire des pré-traitements avant de les envoyer au serveur.

Le formulaire

On a un formulaire rassemblant les composants les plus courants et d'un bouton **Afficher** permettant d'afficher les saisies faites par l'utilisateur.

Un formulaire traité par Javascript

Un champ textuel simple

Un champ textuel sur plusieurs lignes

Des boutons radio : ☒ FM ☐ GO ☐ PO Des choix multiples : ☐ un ☐ deux ☐ trois

Un menu déroulant : 50 F

Une liste :

Un mot de passe :

Un champ de contexte caché :

Informations du formulaire

Effacer

Afficher

Le code

```
<html>
<head>
<title>Un formulaire traité par Javascript</title>
<script language="javascript">
  function afficher(){
    // affiche dans une liste les infos du formulaire

    // on efface d'abord
    effacerInfos();

    // on affiche la valeur des # champs
    with(document.frmExemple){
      // champ caché
      ecrire("champ caché="+cache.value);
      // champ textuel simple
      ecrire("champ textuel simple="+simple.value);
      // champ textuel multiple
      ecrire("champ textuel multiple="+lignes.value);
      // boutons radio
      for(i=0;i<radio.length;i++){
        texte="radio["+i+"]="+radio[i].value;
        if(radio[i].checked) texte+=", coché";
        ecrire(texte);
      }//for
      // cases à cocher
      for(i=0;i<qcm.length;i++){
        texte="qcm["+i+"]="+qcm[i].value;
        if(qcm[i].checked) texte+=", coché";
        ecrire(texte);
      }//for
      //liste déroulante
      ecrire("index sélectionné dans le menu="+menu.selectedIndex);
      for(i=0;i<menu.length;i++){
        texte="menu["+i+"]="+menu.options[i].text;
        if(menu.options[i].selected) texte+=", sélectionné";
        ecrire(texte);
      }//for
      //liste à choix multiple
      for(i=0;i<lstVoitures.length;i++){
```

```

        texte="lstVoitures["+i+"]="+lstVoitures.options[i].text;
        if(lstVoitures.options[i].selected) texte+=" ,sélectionné";
        ecrire(texte);
    }//for
    //mot de passe
    ecrire("mot de passe="+passwd.value);
    }//with
}//function

function ecrire(texte){
    // écrit texte dans la liste des infos
    frmInfos.lstInfos.options[frmInfos.lstInfos.length]=new Option(texte);
}//écrire

function effacerInfos(){
    frmInfos.lstInfos.length=0;
}//effacerInfos
</script>

```

</head>

```

<body bgcolor="#C0C0C0" onload="afficher()">
<center>
<h2>Un formulaire traité par Javascript</h2>
<hr>
<form method="POST" name="frmExemple">
    <input type="hidden" name="cache" value="secret">
    <table border="0">
        <tr>
            <td align="center">Un champ textuel simple</td>
            <td align="center" width="100">&nbsp;&nbsp;&nbsp;</td>
            <td align="center">Un champ textuel sur plusieurs lignes</td>
        </tr>
        <tr>
            <td align="center"><input type="text" size="20" name="simple"></td>
            <td align="center" width="100">&nbsp;&nbsp;&nbsp;</td>
            <td align="center">
                <textarea name="lignes" rows="2" cols="40">Ce texte est modifiable</textarea>
            </td>
        </tr>
    </table>
    <table border="0">
        <tr>
            <td><strong>Des boutons radio :</strong></td>
            <td>
                <input type="radio" checked name="radio" value="FM">FM
            </td>
            <td>
                <input type="radio" name="radio" value="GO">GO
            </td>
            <td>
                <input type="radio" name="radio" value="PO">PO
            </td>
            <td>&nbsp;&nbsp;&nbsp;</td>
        </tr>
        <tr>
            <td><strong>Des choix multiples :</strong></td>
            <td>
                <input type="checkbox" name="qcm" value="un">un
            </td>
            <td>
                <input type="checkbox" name="qcm" value="deux">deux
            </td>
            <td>
                <input type="checkbox" name="qcm" value="trois">trois
            </td>
        </tr>
    </table>
    <table border="0">
        <tr>
            <td>Un menu déroulant : </td>
            <td>
                <select name="menu" size="1">
                    <option>50 F</option>
                    <option>60 F</option>
                    <option>70 F</option>
                    <option>100 F</option>
                </select>
            </td>
            <td>Une liste :</td>
            <td>
                <select name="lstVoitures" multiple size="3">
                    <option>Renault</option>

```

```

        <option>Citroën</option>
        <option>Peugeot</option>
        <option>Fiat</option>
        <option>Audi</option>
    </select>
</td>
</tr>
</table>
<table border="0">
    <tr>
        <td>Un mot de passe : </td>
        <td><input type="password" size="21" name="passwd"></td>
        <td>&nbsp;</td>
        <td>Un champ de contexte caché : </td>
    </tr>
</table>
</form>
<hr>
<h2>Informations du formulaire</h2>
<form name="frmInfos">
    <table>
        <tr>
            <td><input type="button" value="Effacer" onclick="effacerInfos()"></td>
            <td>
                <select name="lstInfos" multiple size="3">
                </select>
            </td>
        </tr>
        <tr>
            <td><input type="button" name="cmdAfficher" value="Afficher" onclick="afficher()">
            </td>
        </tr>
    </table>
</form>
</body>
</html>

```

Les expressions régulières en Javascript

Côté navigateur, Javascript peut être utilisé pour vérifier la validité des données entrées par l'utilisateur avant de les envoyer au serveur. Cela peut souvent se faire à l'aide d'expressions régulières. Voici un programme de test de celles-ci.

La page de test

Les expressions régulières en Javascript

Expression régulière	Chaîne de test
<input type="text" value="^(\\d+)(.*)\\d+\$"/>	<input type="text" value="45abcd67"/>
<input type="button" value="Jouer le test"/>	<input type="button" value="Ajouter aux exemples"/>

Résultats de l'instruction champs=expression régulière.exec(chaine)

Il y a correspondance
champs[0]=[45abcd67]
champs[1]=[45]

Exemples

Modèles	Chaînes	<input type="button" value="Jouer l'exemple"/>
<input type="text" value="^(\\d+)(.*)\\d+\$"/>	<input type="text" value="45abcd67"/>	

Le code de la page

```
<html>
<head>
<title>Les expressions régulières en Javascript</title>
<script language="javascript">
function afficherInfos(){
with(document.frmRegExp){
// qq chose à faire ?
if (! verifier()) return;
// c'est bon - on efface les résultats précédents
effacerInfos();
// vérification du modèle
modele=new RegExp(txtModele.value);
champs=modele.exec(txtChaine.value);
if(champs==null)
// pas de correspondance entre modèle et chaîne
ecrireInfos("pas de correspondance");
else{
// correspondance - on affiche les résultats obtenus
ecrireInfos("Il y a correspondance");
for(i=0;i<champs.length;i++)
ecrireInfos("champs["+i+"]="+champs[i]+"");
}
}
}
}
}

function ecrireInfos(texte){
// écrit texte dans la liste des infos
document.frmRegExp.lstInfos.options[document.frmRegExp.lstInfos.length]=new Option(texte);
}

function effacerInfos(){
frmRegExp.lstInfos.length=0;
}

function jouer(){
```

```

// teste le modèle contre la chaîne dans l'exemple choisi
with(document.frmRegExp){
    txtModele.value=lstModeles.options[lstModeles.selectedIndex].text
    txtChaine.value=lstChaines.options[lstChaines.selectedIndex].text
    afficherInfos();
}
}
}

function ajouter(){
    //ajoute le test courant aux exemples
    with(document.frmRegExp){
        // qq chose à faire ?
        if (! verifier()) return;
        // ajout
        lstModeles.options[lstModeles.length]=new Option(txtModele.value);
        lstChaines.options[lstChaines.length]=new Option(txtChaine.value);
        // raz saisies
        txtModele.value="";
        txtChaine.value="";
    }
}

function verifier(){
    // vérifie que les champs de saisie sont non vides
    with(document.frmRegExp){
        champs=/^\s*$/ .exec(txtModele.value);
        if(champs!=null){
            alert("Vous n'avez pas indiqué de modèle");
            txtModele.focus();
            return false;
        }
        champs=/^\s*$/ .exec(txtChaine.value);
        if(champs!=null){
            alert("Vous n'avez pas indiqué de chaîne de test");
            txtChaine.focus();
            return false;
        }
        // c'est bon
        return true;
    }
}

```

</script>

</head>

<body bgcolor="#C0C0C0">

<center>

<h2>Les expressions régulières en Javascript</h2>

<hr>

<form name="frmRegExp">

<table>

<tr>

<td>Expression régulière</td>

<td>Chaîne de test</td>

</tr>

<tr>

<td><input type="text" name="txtModele" size="20"></td>

<td><input type="text" name="txtChaine" size="20"></td>

</tr>

<tr>

<td>

<input type="button" name="cmdAfficher" value="Jouer le test" onclick="afficherInfos()">

</td>

<td>

<input type="button" name="cmdAjouter" value="Ajouter aux exemples" onclick="ajouter()">

</td>

</tr>

</table>

<hr>

<h2>Résultats de l'instruction champs=expression régulière.exec(chaine)</h2>

<table>

<tr>

<td>

<select name="lstInfos" size="3">

</select>

</td>

</tr>

</table>

<hr>

<h2>Exemples</h2>

```

<table>
<tr>
  <td align="center">Modèles</td>
  <td align="center">Chaînes</td>
</tr>
<tr>
  <td>
    <select name="lstModeles" size="1">
      <option>^%d+$</option>
      <option>^(%d+) (%d+)$</option>
      <option>^(%d+) (.) (%d+)$</option>
      <option>^(%d+) (%s+) (%d+)$</option>
    </select>
  </td>
  <td>
    <select name="lstChaines" size="1">
      <option>67</option>
      <option>56 84</option>
      <option>45abcd67</option>
      <option>45      67</option>
    </select>
  </td>
</tr>
<tr>
  <td colspan="2">
    <input type="button" name="cmdJouer" value="Jouer l'exemple" onclick="jouer()">
  </td>
</tr>
</form>
</body>
</html>

```

Gestion des listes en JavaScript

9.3.1 Le formulaire

Les listes en Javascript

liste 1

aaa
xxx

Vider

Sélection simple

liste 2

qqqq

Vider

liste 1

aaa
xxx

Vider

Sélection multiple

liste 2

qqqq

Vider

9.3.2 Le code

```

<html>
  <head>
    <title>Les listes en Javascript</title>

    <script language="javascript">
      // ajouter
      function ajouter(L1,L2,T){
        // ajoute la valeur du champ T aux listes L1,L2

```

```

// qq chose à faire ?
champs=/^\s*$/ .exec(T.value);
if(champs!=null){
    // le champ est vide
    alert("Vous n'avez pas indiqué la valeur à ajouter");
    txtElement.focus();
    return;
} //if
// on ajoute l'élément
L1.options[L1.length]=new Option(T.value);
L2.options[L2.length]=new Option(T.value);
T.value="";
} //ajouter

//vider
function vider(L){
    // vide la liste L
    L.length=0;
} //vider

//transfert
function transfert(L1,L2,simple){
    //transfère dans L2 les éléments sélectionnés dans la liste L1

    // qq chose à faire ?
    // index de l'élément sélectionné dans L1
    index1=L1.selectedIndex;
    if(index1== -1){
        alert("Vous n'avez pas sélectionné d'élément");
        return;
    } //if
    // quel est le mode de sélection des éléments des listes
    if(simple){ // sélection simple
        element1=L1.options[index1].text;
        //ajout dans L2
        L2.options[L2.length]=new Option(element1);
        //suppression dans L1
        L1.options[index1]=null;
    } //simple
    if(! simple){ //sélection multiple
        //on parcourt la liste 1 en sens inverse
        for(i=L1.length-1;i>=0;i--){
            //élément sélectionné ?
            if(L1.options[i].selected){
                //on l'ajoute à L2
                L2.options[L2.length]=new Option(L1.options[i].text);
                //on le supprime de L1
                L1.options[i]=null;
            } //if
        } //for i
    } //if ! simple
} //transfert
</script>

```

</head>

```

<body bgcolor="#C0C0C0">
  <center>
    <h2>Les listes en Javascript</h2>
    <hr>
    <form name="frmListes">
      <table>
        <tr>
          <td>

```

```

            <input type="button" name="cmdAjouter" value="Ajouter" onclick="ajouter
(1st1A,1st1B,txtElement)">
          </td>
        </tr>

```

```

          <td>
            <input type="text" name="txtElement">
          </td>
        </tr>
      </table>
    <table>
      <tr>
        <td align="center">liste 1</td>

```

```

        <td align="center"><input type="button" value=">>" onclick="transfert(1st1A,1st2A,true)"></td>
        <td align="center"><input type="button" value="<<" onclick="transfert(1st2A,1st1A,true)"></td>
      </tr>
    </table>
    <td align="center">liste 2</td>
    <td width="30"></td>

```



```

<td align="center">liste 1</td>
<td align="center"><input type="button" value="">>" onclick="transfert(lst1B,lst2B,false) "</td>
<td align="center"><input type="button" value=""><" onclick="transfert(lst2B,lst1B,false) "</td>
<td align="center">liste 2</td>
</tr>
<tr>
<td></td>
<td align="center">
<select name="lst1A" size="5">
</select>
</td>
<td align="center">
<select name="lst2A" size="5">
</select>
</td>
<td></td>
<td></td>
<td></td>
<td align="center">
<select name="lst1B" size="5" multiple >
</select>
</td>
<td align="center">
<select name="lst2B" size="5" multiple>
</select>
</td>
</tr>
<tr>
<td></td>
<td align="center"><input type="button" value="Vider" onclick="vider(lst1A) "</td>
<td align="center"><input type="button" value="Vider" onclick="vider(lst2A) "</td>
<td></td>
<td></td>
<td></td>
<td align="center"><input type="button" value="Vider" onclick="vider(lst1B) "</td>
<td align="center"><input type="button" value="Vider" onclick="vider(lst2B) "</td>
<td></td>
</tr>
<tr>
<td></td>
<td colspan="2"><strong>Sélection simple</strong></td>
<td></td>
<td></td>
<td></td>
<td colspan="2"><strong>Sélection multiple</strong></td>
<td></td>
</tr>
</table>
<hr>
</form>
</body>
</html>

```