

Développement WEB avec ASP.NET 1.1

- Volume 1 -

serge.tahe@istia.univ-angers.fr
avril 2004

Introduction

Ce document est un support de cours : ce n'est pas un cours complet. Des approfondissements nécessitent l'aide d'un enseignant et par ailleurs un certain nombre de thèmes n'ont pas été abordés. Son écriture a été influencée par le passé de l'auteur qui avait écrit auparavant deux documents sur le développement web, en Java tout d'abord puis en PHP. Ces deux documents ont une structure analogue permettant de comparer les deux technologies sur les mêmes exemples. Il a été ici fait de même pour le développement en ASP.NET. Cela donne un document assez différent de ce qu'on trouve en librairie où quasiment tous les livres insistent sur le fait qu'ASP.NET permet de développer une application web comme on développe une application windows. L'interface qui est présentée dans le navigateur du client peut être construite comme une interface windows :

- avec des IDE tels que Visual Studio.NET ou WebMatrix, l'interface utilisateur est construite avec des objets graphiques que l'on dépose dans la fenêtre de conception
- ces objets ont des propriétés, méthodes et génèrent des événements

Ici, on dit le strict minimum sur ces concepts considérés comme les plus novateurs de ASP.NET... Ces concepts importants mais non fondamentaux sont présentés dans le volume 2 de ce cours. Dans ce volume 1, il a semblé plus important d'insister sur les fondements du développement web qu'on retrouvera quelque soit la technologie utilisée (Java, PHP, ASP.NET). Les extensions propriétaires de ASP.NET qui permettent une meilleure productivité seront vues ultérieurement. Nous insistons beaucoup dans notre présentation du développement web sur l'architecture MVC (Modèle, Vue, Contrôleur) souvent préconisée pour construire des applications web. Ce concept est indépendant de la technologie utilisée. Il se trouve qu'il cadre mal avec celui de concevoir une application web comme une application windows préconisé par la technologie ASP.NET. C'est l'autre raison qui a fait que ce concept tant vanté dans la littérature ASP.NET a été relégué dans le volume 2.

Parce que ce document est destiné à des étudiants, nous n'utilisons pour nos exemples que des outils librement disponibles sur internet. Le lecteur pourra ainsi se les procurer et tester les exemples sur sa machine personnelle. L'annexe "Les outils du web" donne des indications pour récupérer et installer ces outils.

Ce document comporte peut-être encore des erreurs : toute suggestion constructive est la bienvenue à l'adresse serge.tabe@istia.univ-angers.fr.

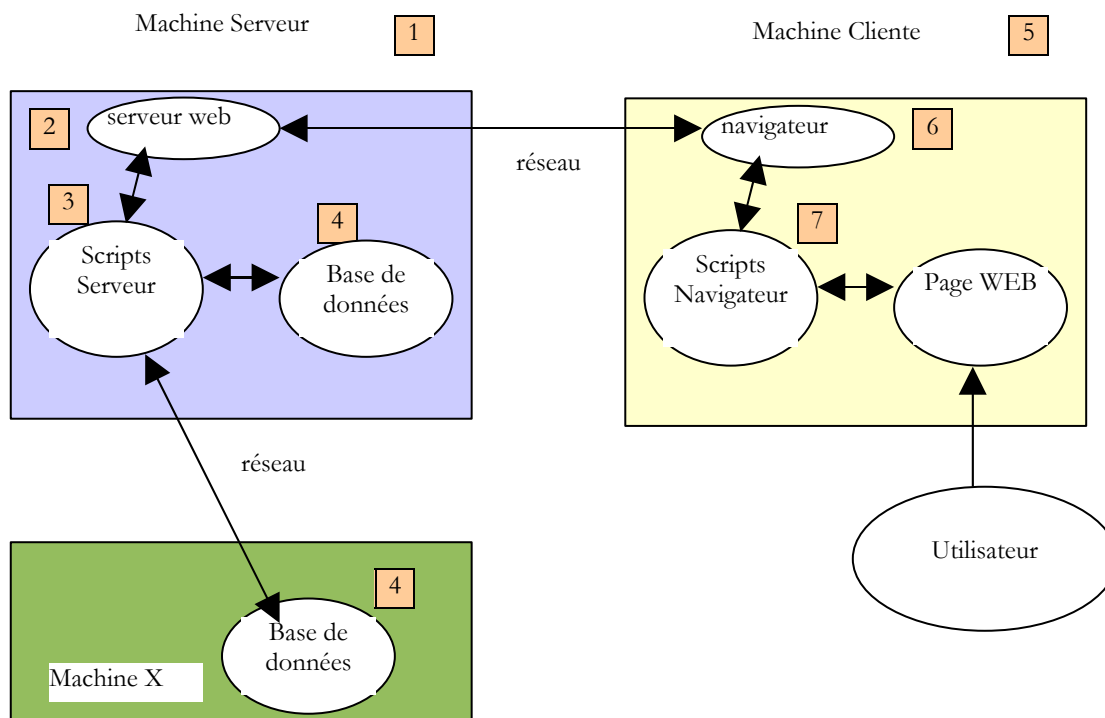
Serge Tahé

Avril 2004

1 Les bases

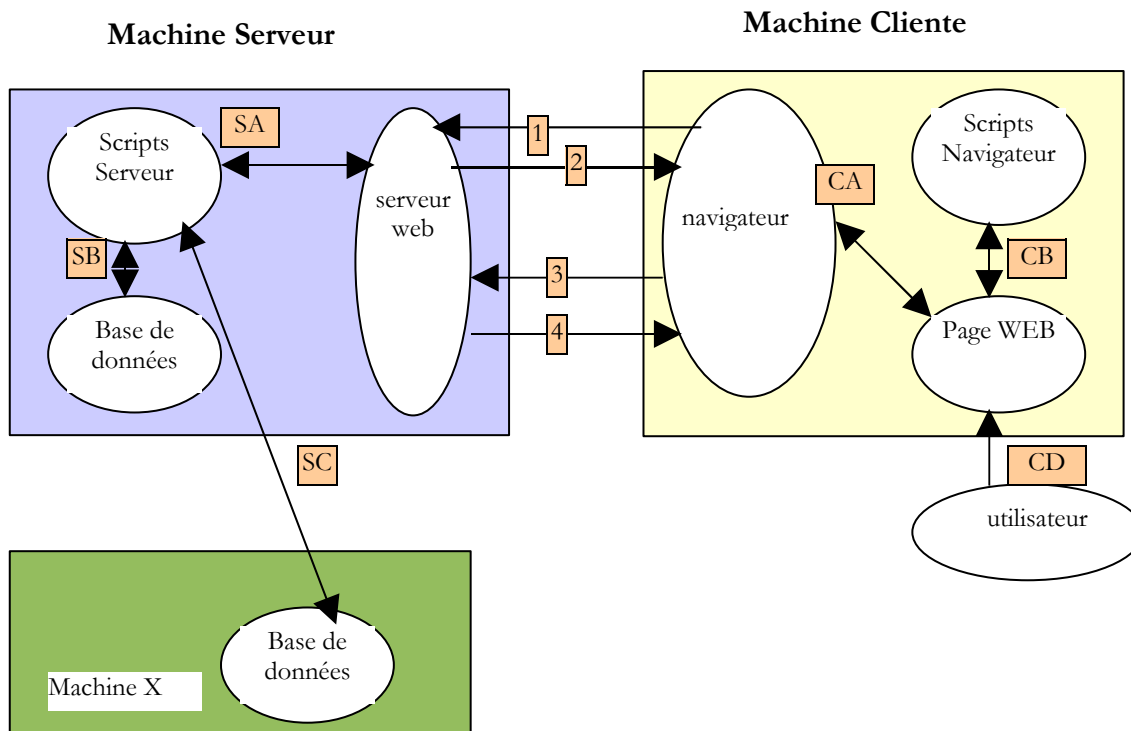
Dans ce chapitre, nous présentons les bases de la programmation Web. Il a pour but essentiel de faire découvrir les grands principes de la programmation Web qui sont indépendants de la technologie particulière utilisée pour les mettre en oeuvre. Il présente de nombreux exemples qu'il est conseillé de tester afin de "s'imprégner" peu à peu de la philosophie du développement web. Les outils gratuits nécessaires à leurs tests sont présentés en fin de document dans l'annexe intitulée "Les outils du web".

1.1 Les composants d'une application Web



Numéro	Rôle	Exemples courants
1	OS Serveur	Linux, Windows
2	Serveur Web	Apache (Linux, Windows) IIS (NT), PWS(Win9x), Cassini (Windows+plate-forme .NET)
3	Scripts exécutés côté serveur. Ils peuvent l'être par des modules du serveur ou par des programmes externes au serveur (CGI).	PERL (Apache, IIS, PWS) VBSCRIPT (IIS,PWS) JAVASCRIPT (IIS,PWS) PHP (Apache, IIS, PWS) JAVA (Apache, IIS, PWS) C#, VB.NET (IIS)
4	Base de données - Celle-ci peut être sur la même machine que le programme qui l'exploite ou sur une autre via Internet.	Oracle (Linux, Windows) MySQL (Linux, Windows) Postgres (Linux, Windows) Access (Windows) SQL Server (Windows)
5	OS Client	Linux, Windows
6	Navigateur Web	Netscape, Internet Explorer, Mozilla, Opera
7	Scripts exécutés côté client au sein du navigateur. Ces scripts n'ont aucun accès aux disques du poste client.	VBscript (IE) Javascript (IE, Netscape) Perlscript (IE)

1.1 Les échanges de données dans une application web avec formulaire



Numéro	Rôle
1	Le navigateur demande une URL pour la 1ère fois (<i>http://machine/url</i>). Aucun paramètre n'est passé.
2	Le serveur Web lui envoie la page Web de cette URL. Elle peut être statique ou bien dynamiquement générée par un script serveur (SA) qui a pu utiliser le contenu de bases de données (SB, SC). Ici, le script détectera que l'URL a été demandée sans passage de paramètres et générera la page WEB initiale. Le navigateur reçoit la page et l'affiche (CA). Des scripts côté navigateur (CB) ont pu modifier la page initiale envoyée par le serveur. Ensuite par des interactions entre l'utilisateur (CD) et les scripts (CB) la page Web va être modifiée. Les formulaires vont notamment être remplis.
3	L'utilisateur valide les données du formulaire qui doivent alors être envoyées au serveur web. Le navigateur redemande l'URL initiale ou une autre selon les cas et transmet en même temps au serveur les valeurs du formulaire. Il peut utiliser pour ce faire deux méthodes appelées GET et POST. A réception de la demande du client, le serveur déclenche le script (SA) associé à l'URL demandée, script qui va détecter les paramètres et les traiter.
4	Le serveur délivre la page WEB construite par programme (SA, SB, SC). Cette étape est identique à l'étape 2 précédente. Les échanges se font désormais selon les étapes 2 et 3.

1.2 Notations

Dans la suite, nous supposons qu'un certain nombre d'outils ont été installés et adopterons les notations suivantes :

notation	signification
<apache>	racine de l'arborescence du serveur apache
<apache-DocumentRoot>	racine des pages Web délivrées par Apache. C'est sous cette racine que doivent se trouver les pages Web. Ainsi l'URL <i>http://localhost/page1.htm</i> correspond au fichier <i><apache-DocumentRoot>\page1.htm</i> .
<apache-cgi-bin>	racine de l'arborescence lié à l'alias <i>cgi-bin</i> et où l'on peut placer des scripts CGI pour Apache. Ainsi

l'URL `http://localhost/cgi-bin/test1.pl` correspond au fichier `<apache-cgi-bin>\test1.pl`.

<code><IIS-DocumentRoot></code>	racine des pages Web délivrées par IIS, PWS ou Cassini. C'est sous cette racine que doivent se trouver les pages Web. Ainsi l'URL <code>http://localhost/page1.htm</code> correspond au fichier <code><IIS-DocumentRoot>\page1.htm</code> .
<code><perl></code>	racine de l'arborescence du langage Perl. L'exécutable <code>perl.exe</code> se trouve en général dans <code><perl>\bin</code> .
<code><php></code>	racine de l'arborescence du langage PHP. L'exécutable <code>php.exe</code> se trouve en général dans <code><php></code> .
<code><java></code>	racine de l'arborescence de java. Les exécutables liés à java se trouvent dans <code><java>\bin</code> .
<code><tomcat></code>	racine du serveur Tomcat. On trouve des exemples de servlets dans <code><tomcat>\webapps\examples\servlets</code> et des exemples de pages JSP dans <code><tomcat>\webapps\examples\jsp</code>

On se reportera pour chacun de ces outils à l'annexe qui donne une aide pour leur installation.

1.3 Pages Web statiques, Pages Web dynamiques

Une page statique est représentée par un fichier HTML. Une page dynamique est, elle, générée "à la volée" par le serveur web. Nous vous proposons dans ce paragraphe divers tests avec différents serveurs web et différents langages de programmation afin de montrer l'universalité du concept web. Nous utiliserons deux serveurs web notés Apache et IIS. Si IIS est un produit commercial, il est cependant décliné en deux versions plus limitées mais gratuites :

- PWS pour les machines Win9x
- Cassini pour les machines Windows 2000 et XP

Le dossier `<IIS-DocumentRoot>` est habituellement le dossier [lecteur:\inetpub\wwwroot] où [lecteur] est le disque (C, D, ...) où a été installé IIS. Il en est de même pour PWS. Pour Cassini, le dossier `<IIS-DocumentRoot>` dépend de la façon dont le serveur a été lancé. Dans l'annexe, il est montré que le serveur Cassini peut être lancé dans une fenêtre Dos (ou par un raccourci) de la façon suivante :

```
dos>webserver /port:N /path:"P" /vpath:"/V"
```

L'application [WebServer] appelée également serveur web Cassini admet trois paramètres :

- **/port** : n° de port du service web. Peut-être quelconque. A par défaut la valeur 80
- **/path** : chemin physique d'un dossier du disque
- **/vpath** : dossier virtuel associé au dossier physique précédent. On prêtera attention au fait que la syntaxe n'est pas `/path=chemin` mais `/vpath:chemin`, contrairement à ce que dit le panneau d'aide de Cassini.

Si Cassini est lancé de la façon suivante :

```
dos>webserver /port:N /path:"P" /vpath:"/"
```

alors le dossier P est la racine de l'arborescence web du serveur Cassini. C'est donc ce dossier qui est désigné par `<IIS-DocumentRoot>`. Ainsi dans l'exemple suivant :

```
dos>webserver /path:"d:\data\devel\webmatrix" /vpath:"/"
```

le serveur Cassini travaillera sur le port 80 et la racine de son arborescence `<IIS-DocumentRoot>` est le dossier [d:\data\devel\webmatrix]. Les pages Web à tester devront se trouver sous cette racine.

Dans la suite, chaque application web sera représentée par un unique fichier qu'on pourra construire avec n'importe quel éditeur de texte. Aucun IDE n'est requis.

1.3.1 Page statique HTML (HyperText Markup Language)

Considérons le code HTML suivant :

```
<html>
<head>
  <title>essai 1 : une page statique</title>
</head>
<body>
  <center>
    <h1>Une page statique...</h1>
  </body>
```

</html>

qui produit la page web suivante :



Les tests

- lancer le serveur Apache
- mettre le script *essai1.html* dans *<apache-DocumentRoot>*
- visualiser l'URL *http://localhost/essai1.html* avec un navigateur
- arrêter le serveur Apache

- lancer le serveur IIS/PWS/Cassini
- mettre le script *essai1.html* dans *<IIS-DocumentRoot>*
- visualiser l'URL *http://localhost/essai1.html* avec un navigateur

1.3.2 Une page ASP (Active Server Pages)

Le script *essai2.asp* :

```
<html>
<head>
  <title>essai 1 : une page asp</title>
</head>
<body>
  <center>
    <h1>Une page asp générée dynamiquement par le serveur PWS</h1>
    <h2>Il est <% =time %></h2>
    <br>
    A chaque fois que vous rafraîchissez la page, l'heure change.
  </body>
</html>
```

produit la page web suivante :

Une page asp générée dynamiquement par le serveur PWS

Il est 11:41:40

A chaque fois que vous rafraîchissez la page, l'heure change.

Le test

- lancer le serveur IIS/PWS
- mettre le script *essai2.asp* dans *<IIS-DocumentRoot>*
- demander l'URL *http://localhost/essai2.asp* avec un navigateur

1.3.3 Un script PERL (Practical Extracting and Reporting Language)

Le script *essai3.pl* :

```
#!/d:\perl\bin\perl.exe

($secondes,$minutes,$heure)=localtime(time);

print <<HTML
Content-type: text/html
```

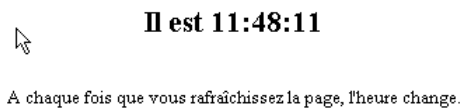
```

<html>
  <head>
    <title>essai 1 : un script Perl</title>
  </head>
  <body>
    <center>
      <h1>Une page générée dynamiquement par un script Perl</h1>
      <h2>Il est $heure:$minutes:$secondes</h2>
      <br>
      A chaque fois que vous rafraîchissez la page, l'heure change.
    </body>
</html>
HTML
;

```

La première ligne est le chemin de l'exécutable *perl.exe*. Il faut l'adapter si besoin est. Une fois exécuté par un serveur Web, le script produit la page suivante :

Une page générée dynamiquement par un script Perl



Il est 11:48:11

A chaque fois que vous rafraîchissez la page, l'heure change.

Le test

- serveur Web : Apache
- pour information, visualisez le fichier de configuration *srm.conf* ou *httpd.conf* selon la version d'Apache dans `<apache>\conf` et rechercher la ligne parlant de *cgi-bin* afin de connaître le répertoire `<apache-cgi-bin>` dans lequel placer *essai3.pl*.
- mettre le script *essai3.pl* dans `<apache-cgi-bin>`
- demander l'url `http://localhost/cgi-bin/essai3.pl`

A noter qu'il faut davantage de temps pour avoir la page *perl* que la page *asp*. Ceci parce que le script Perl est exécuté par un interpréteur Perl qu'il faut charger avant qu'il puisse exécuter le script. Il ne reste pas en permanence en mémoire.

1.3.4 Un script PHP (HyperText Processor)

Le script `essai4.php`

```

<html>
  <head>
    <title>essai 4 : une page php</title>
  </head>
  <body>
    <center>
      <h1>Une page PHP générée dynamiquement</h1>
      <h2>
<?
  $maintenant=time();
  echo date("j/m/y, h:i:s",$maintenant);
?>
      </h2>
      <br>
      A chaque fois que vous rafraîchissez la page, l'heure change.
    </body>
</html>

```

Le script précédent produit la page web suivante :

12/09/00, 11:54:19

A chaque fois que vous rafraîchissez la page, l'heure change.

Les tests

- consulter le fichier de configuration *srm.conf* ou *httpd.conf* d'Apache dans `<Apache>\conf`
- pour information, vérifier les lignes de configuration de *php*
- lancer le serveur Apache
- mettre *essai4.php* dans `<apache-DocumentRoot>`
- demander l'URL `http://localhost/essai4.php`

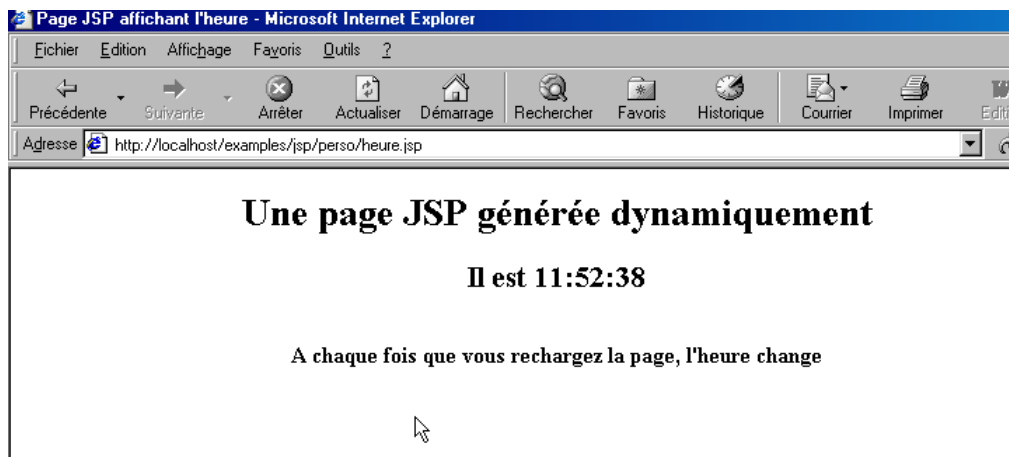
- lancer le serveur IIS/PWS
- pour information, vérifier la configuration de PWS à propos de *php*
- mettre *essai4.php* dans `<IIS-DocumentRoot>\php`
- demander l'URL `http://localhost/essai4.php`

1.3.5 Un script JSP (Java Server Pages)

Le script heure.jsp

```
<% //programme Java affichant l'heure %>
<%@ page import="java.util.*" %>
<%
// code JAVA pour calculer l'heure
Calendar calendrier=Calendar.getInstance();
int heures=calendrier.get(Calendar.HOUR_OF_DAY);
int minutes=calendrier.get(Calendar.MINUTE);
int secondes=calendrier.get(Calendar.SECOND);
// heures, minutes, secondes sont des variables globales
// qui pourront être utilisées dans le code HTML
%>
<% // code HTML %>
<html>
<head>
<title>Page JSP affichant l'heure</title>
</head>
<body>
<center>
<h1>Une page JSP générée dynamiquement</h1>
<h2>Il est <%=heures%>:<%=minutes%>:<%=secondes%></h2>
<br>
<h3>A chaque fois que vous rechargez la page, l'heure change</h3>
</body>
</html>
```

Une fois exécuté par le serveur web, ce script produit la page suivante :



Les tests

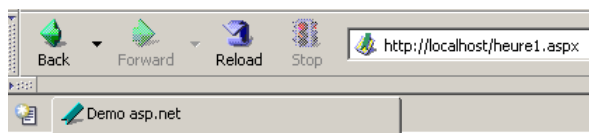
- mettre le script *heure.jsp* dans `<tomcat>\jakarta-tomcat\webapps\examples\jsp` (Tomcat 3.x) ou dans `<tomcat>\webapps\examples\jsp` (Tomcat 4.x)
- lancer le serveur Tomcat
- demander l'URL `http://localhost:8080/examples/jsp/heure.jsp`

1.3.6 Une page ASP.NET

Le script `heure1.aspx` :

```
<html>
<head>
  <title>Démo asp.net </title>
</head>
<body>
  Il est <% =Date.Now.ToString("hh:mm:ss") %>
</body>
</html>
```

Une fois exécuté par le serveur web, ce script produit la page suivante :



Il est 17:07:59

Ce test nécessite d'avoir une machine windows où la plate-forme .NET a été installée (cf annexe).

- mettre le script *heure1.aspx* dans `<IIS-DocumentRoot>`
- lancer le serveur IIS/CASSINI
- demander l'URL `http://localhost/heure1.aspx`

1.3.7 Conclusion

Les exemples précédents ont montré que :

- une page HTML pouvait être générée dynamiquement par un programme. C'est tout le sens de la programmation Web.
- que les langages et les serveurs web utilisés pouvaient être divers. Actuellement on observe les grandes tendances suivantes :
 - les tandems Apache/PHP (Windows, Linux) et IIS/PHP (Windows)
 - la technologie ASP.NET sur les plate-formes Windows qui associent le serveur IIS à un langage .NET (C#, VB.NET, ...)
 - la technologie des servlets Java et pages JSP fonctionnant avec différents serveurs (Tomcat, Apache, IIS) et sur différentes plate-formes (Windows, Linux).

1.4 Scripts côté navigateur

Une page HTML peut contenir des scripts qui seront exécutés par le navigateur. Les langages de script côté navigateur sont nombreux. En voici quelques-uns :

Langage	Navigateurs utilisables
Vbscript	IE
Javascript	IE, Netscape
PerlScript	IE
Java	IE, Netscape

Prenons quelques exemples.

1.4.1 Une page Web avec un script Vbscript, côté navigateur

La page vbs1.html

```
<html>
<head>
  <title>essai : une page web avec un script vb</title>
  <script language="vbscript">
    function reagir
      alert "Vous avez cliqué sur le bouton OK"
    end function
  </script>
</head>
<body>
<center>
  <h1>Une page Web avec un script VB</h1>
  <table>
    <tr>
      <td>Cliquez sur le bouton</td>
      <td><input type="button" value="OK" name="cmdOK" onclick="reagir"></td>
    </tr>
  </table>
</body>
</html>
```

La page HTML ci-dessus ne contient pas simplement du code HTML mais également un programme destiné à être exécuté par le navigateur qui aura chargé cette page. Le code est le suivant :

```
<script language="vbscript">
  function reagir
    alert "Vous avez cliqué sur le bouton OK"
  end function
</script>
```

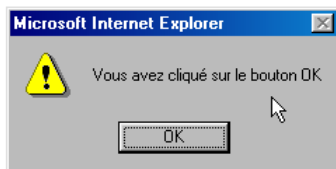
Les balises `<script></script>` servent à délimiter les scripts dans la page HTML. Ces scripts peuvent être écrits dans différents langages et c'est l'option *language* de la balise `<script>` qui indique le langage utilisé. Ici c'est VBScript. Nous ne chercherons pas à détailler ce langage. Le script ci-dessus définit une fonction appelée *reagir* qui affiche un message. Quand cette fonction est-elle appelée ? C'est la ligne de code HTML suivante qui nous l'indique :

```
<input type="button" value="OK" name="cmdOK" onclick="reagir">
```

L'attribut *onclick* indique le nom de la fonction à appeler lorsque l'utilisateur cliquera sur le bouton OK. Lorsque le navigateur aura chargé cette page et que l'utilisateur cliquera sur le bouton OK, on aura la page suivante :

Une page Web avec un script VB

Cliquez sur le bouton



Les tests

Seul le navigateur IE est capable d'exécuter des scripts VBScript. Netscape nécessite des compléments logiciels pour le faire. On pourra faire les tests suivants :

- serveur Apache
- script *vbs1.html* dans `<apache-DocumentRoot>`
- demander l'url `http://localhost/vbs1.html` avec le navigateur IE

- serveur IIS/PWS

Les bases

- script *vbs1.html* dans *<pbs-DocumentRoot>*
- demander l'URL *http://localhost/vbs1.html* avec le navigateur IE

1.4.2 Une page Web avec un script Javascript, côté navigateur

La page : *js1.html*

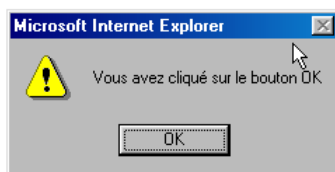
```
<html>
<head>
  <title>essai 4 : une page web avec un script Javascript</title>
  <script language="javascript">
    function reagir(){
      alert ("Vous avez cliqué sur le bouton OK");
    }
  </script>
</head>

<body>
  <center>
  <h1>Une page Web avec un script Javascript</h1>
  <table>
    <tr>
      <td>Cliquez sur le bouton</td>
      <td><input type="button" value="OK" name="cmdOK" onclick="reagir()"></td>
    </tr>
  </table>
</body>
</html>
```

On a là quelque chose d'identique à la page précédente si ce n'est qu'on a remplacé le langage VBScript par le langage Javascript. Celui-ci présente l'avantage d'être accepté par les deux navigateurs IE et Netscape. Son exécution donne les mêmes résultats :

Une page Web avec un script Javascript

Cliquez sur le bouton

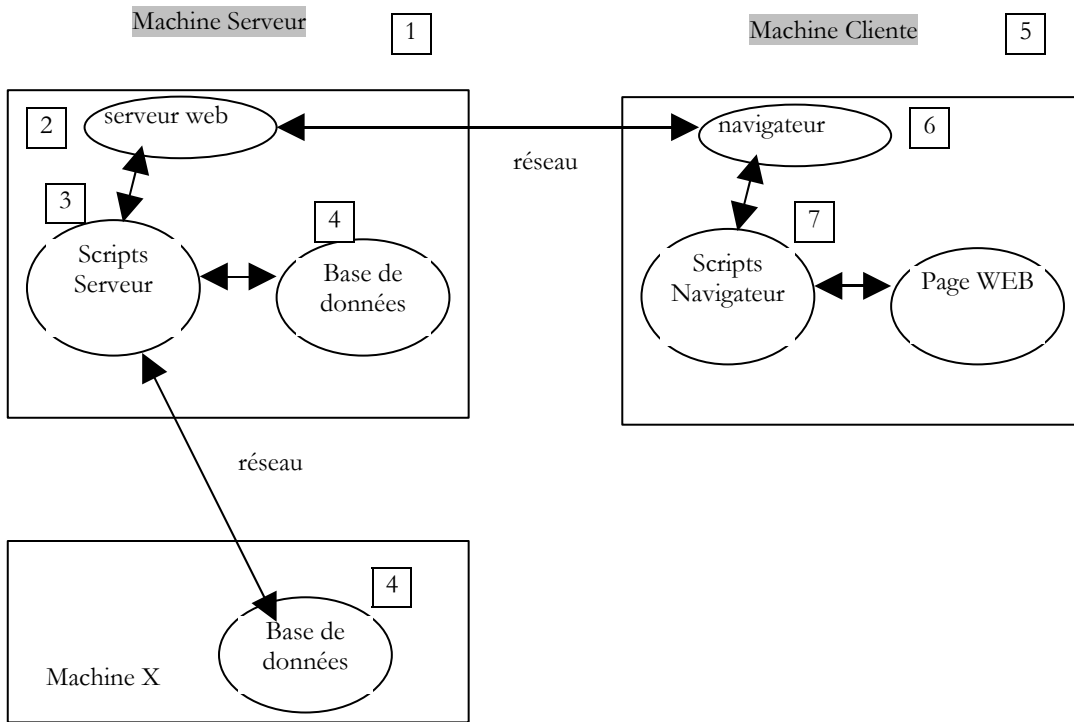


Les tests

- serveur Apache
- script *js1.html* dans *<apache-DocumentRoot>*
- demander l'URL *http://localhost/js1.html* avec le navigateur IE ou Netscape
- serveur IIS/PWS
- script *js1.html* dans *<pbs-DocumentRoot>*
- demander l'URL *http://localhost/js1.html* avec le navigateur IE ou Netscape

1.5 Les échanges client-serveur

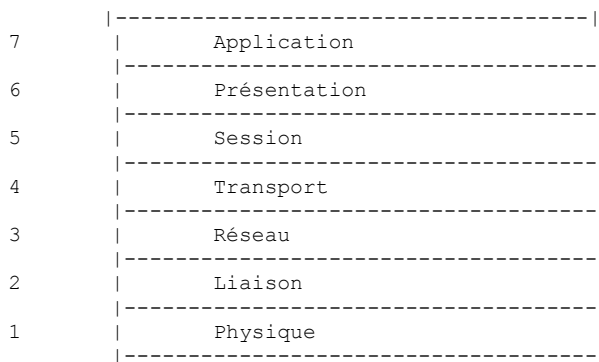
Revenons à notre schéma de départ qui illustre les acteurs d'une application web :



Nous nous intéressons ici aux échanges entre la machine cliente et la machine serveur. Ceux-ci se font au travers d'un réseau et il est bon de rappeler la structure générale des échanges entre deux machines distantes.

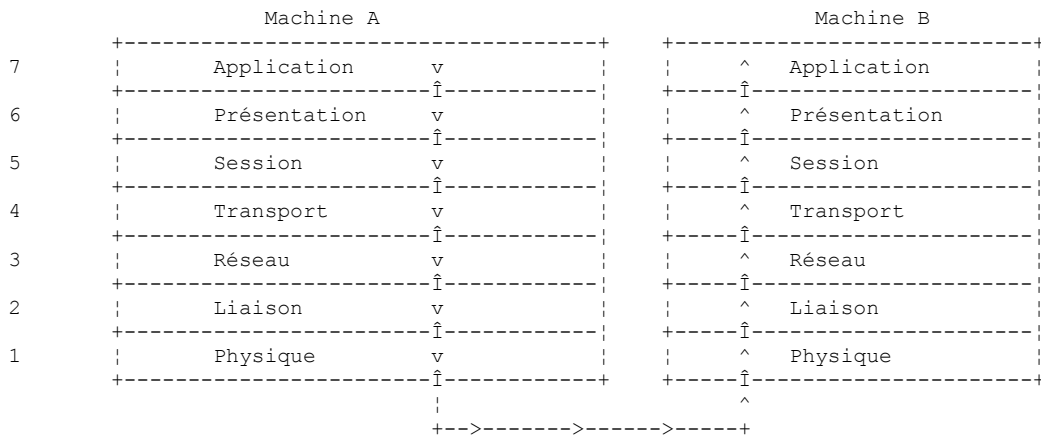
1.5.1 Le modèle OSI

Le modèle de réseau ouvert appelé **OSI** (**O**pen **S**ystems **I**nterconnection Reference Model) défini par l'**ISO** (**I**nternational **S**tandards **O**rganisation) décrit un réseau idéal où la communication entre machines peut être représentée par un modèle à sept couches :



Chaque couche reçoit des services de la couche inférieure et offre les siens à la couche supérieure. Supposons que deux applications situées sur des machines A et B différentes veulent communiquer : elles le font au niveau de la couche *Application*. Elles n'ont pas besoin de connaître tous les détails du fonctionnement du réseau : chaque application remet l'information qu'elle souhaite transmettre à la couche du dessous : la couche *Présentation*. L'application n'a donc à connaître que les règles d'interfaçage avec la couche *Présentation*. Une fois l'information dans la couche *Présentation*, elle est passée selon d'autres règles à la couche *Session* et ainsi de suite, jusqu'à ce que l'information arrive sur le support physique et soit transmise physiquement à la machine destination. Là, elle subira le traitement inverse de celui qu'elle a subi sur la machine expéditeur.

A chaque couche, le processus expéditeur chargé d'envoyer l'information, l'envoie à un processus récepteur sur l'autre machine appartenant à la même couche que lui. Il le fait selon certaines règles que l'on appelle le **protocole** de la couche. On a donc le schéma de communication final suivant :



Le rôle des différentes couches est le suivant :

- Physique** Assure la transmission de bits sur un support physique. On trouve dans cette couche des équipements terminaux de traitement des données (E.T.T.D.) tels que terminal ou ordinateur, ainsi que des équipements de terminaison de circuits de données (E.T.C.D.) tels que modulateur/démodulateur, multiplexeur, concentrateur. Les points d'intérêt à ce niveau sont :
 - . le choix du codage de l'information (analogique ou numérique)
 - . le choix du mode de transmission (synchrone ou asynchrone).

- Liaison données** de Masque les particularités physiques de la couche Physique. Détecte et corrige les erreurs de transmission.

- Réseau** Gère le chemin que doivent suivre les informations envoyées sur le réseau. On appelle cela le *roulage* : déterminer la route à suivre par une information pour qu'elle arrive à son destinataire.

- Transport** Permet la communication entre deux applications alors que les couches précédentes ne permettaient que la communication entre machines. Un service fourni par cette couche peut être le multiplexage : la couche transport pourra utiliser une même connexion réseau (de machine à machine) pour transmettre des informations appartenant à plusieurs applications.

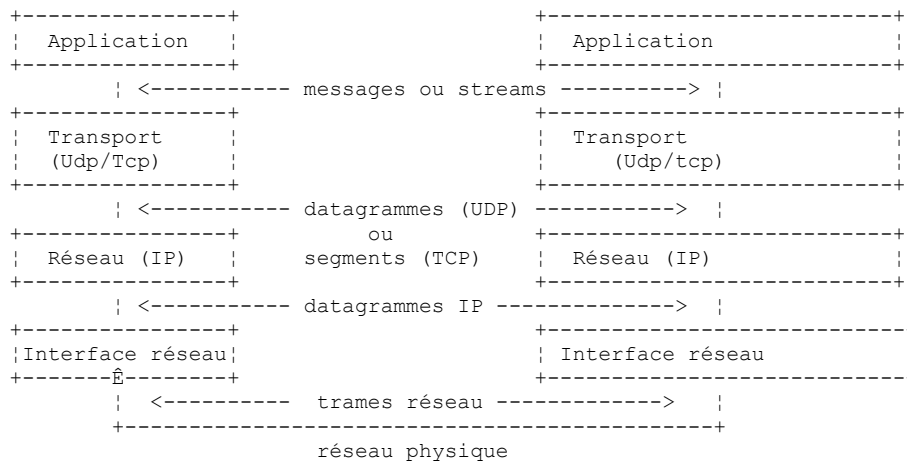
- Session** On va trouver dans cette couche des services permettant à une application d'ouvrir et de maintenir une session de travail sur une machine distante.

- Présentation** Elle vise à uniformiser la représentation des données sur les différentes machines. Ainsi des données provenant d'une machine A, vont être "habillées" par la couche *Présentation* de la machine A, selon un format standard avant d'être envoyées sur le réseau. Parvenues à la couche *Présentation* de la machine destinatrice B qui les reconnaîtra grâce à leur format standard, elles seront habillées d'une autre façon afin que l'application de la machine B les reconnaisse.

- Application** A ce niveau, on trouve les applications généralement proches de l'utilisateur telles que la messagerie électronique ou le transfert de fichiers.

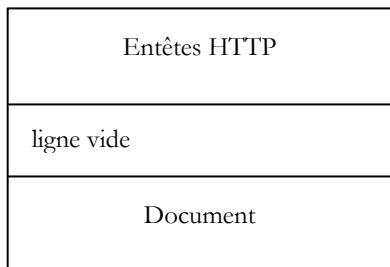
1.5.2 Le modèle TCP/IP

Le modèle OSI est un modèle idéal. La suite de protocoles TCP/IP s'en approche sous la forme suivante :



- l'**interface réseau** (la carte réseau de l'ordinateur) assure les fonctions des couches 1 et 2 du modèle OSI
- la couche **IP** (Internet Protocol) assure les fonctions de la couche 3 (réseau)
- la couche **TCP** (Transfer Control Protocol) ou **UDP** (User Datagram Protocol) assure les fonctions de la couche 4 (transport). Le protocole TCP s'assure que les paquets de données échangés par les machines arrivent bien à destination. Si ce n'est pas le cas, il renvoie les paquets qui se sont égarés. Le protocole UDP ne fait pas ce travail et c'est alors au développeur d'applications de le faire. C'est pourquoi sur l'internet qui n'est pas un réseau fiable à 100%, c'est le protocole TCP qui est le plus utilisé. On parle alors de réseau **TCP-IP**.
- la couche **Application** recouvre les fonctions des niveaux 5 à 7 du modèle OSI.

Les applications web se trouvent dans la couche *Application* et s'appuient donc sur les protocoles TCP-IP. Les couches *Application* des machines clientes et serveur s'échangent des messages qui sont confiées aux couches 1 à 4 du modèle pour être acheminées à destination. Pour se comprendre, les couches application des deux machines doivent "parler" un même langage ou protocole. Celui des applications Web s'appelle **HTTP** (HyperText Transfer Protocol). C'est un protocole de type texte, c.a.d. que les machines échangent des lignes de texte sur le réseau pour se comprendre. Ces échanges sont normalisés, c.a.d. que le client dispose d'un certain nombre de messages pour indiquer exactement ce qu'il veut au serveur et ce dernier dispose également d'un certain nombre de messages pour donner au client sa réponse. Cet échange de messages a la forme suivante :



Client --> Serveur

Lorsque le client fait sa demande au serveur web, il envoie

1. des lignes de texte au format HTTP pour indiquer ce qu'il veut
2. une ligne vide
3. optionnellement un document

Serveur --> Client

Lorsque le serveur fait sa réponse au client, il envoie

1. des lignes de texte au format HTTP pour indiquer ce qu'il envoie
2. une ligne vide
3. optionnellement un document

Les échanges ont donc la même forme dans les deux sens. Dans les deux cas, il peut y avoir envoi d'un document même s'il est rare qu'un client envoie un document au serveur. Mais le protocole HTTP le prévoit. C'est ce qui permet par exemple aux abonnés d'un fournisseur d'accès de télécharger des documents divers sur leur site personnel hébergé chez ce fournisseur d'accès. Les documents échangés peuvent être quelconques. Prenons un navigateur demandant une page web contenant des images :

Les bases

1. le navigateur se connecte au serveur web et demande la page qu'il souhaite. Les ressources demandées sont désignées de façon unique par des URL (Uniform Resource Locator). Le navigateur n'envoie que des entêtes HTTP et aucun document.
2. le serveur lui répond. Il envoie tout d'abord des entêtes HTTP indiquant quel type de réponse il envoie. Ce peut être une erreur si la page demandée n'existe pas. Si la page existe, le serveur dira dans les entêtes HTTP de sa réponse qu'après ceux-ci il va envoyer un document **HTML** (HyperText Markup Language). Ce document est une suite de lignes de texte au format HTML. Un texte HTML contient des balises (marqueurs) donnant au navigateur des indications sur la façon d'afficher le texte.
3. le client sait d'après les entêtes HTTP du serveur qu'il va recevoir un document HTML. Il va analyser celui-ci et s'apercevoir peut-être qu'il contient des références d'images. Ces dernières ne sont pas dans le document HTML. Il fait donc une nouvelle demande au même serveur web pour demander la première image dont il a besoin. Cette demande est identique à celle faite en 1, si ce n'est que la ressource demandée est différente. Le serveur va traiter cette demande en envoyant à son client l'image demandée. Cette fois-ci, dans sa réponse, les entêtes HTTP préciseront que le document envoyé est une image et non un document HTML.
4. le client récupère l'image envoyée. Les étapes 3 et 4 vont être répétées jusqu'à ce que le client (un navigateur en général) ait tous les documents lui permettant d'afficher l'intégralité de la page.

1.5.3 Le protocole HTTP

Découvrons le protocole HTTP sur des exemples. Que s'échangent un navigateur et un serveur web ?

1.5.3.1 La réponse d'un serveur HTTP

Nous allons découvrir ici comment un serveur web répond aux demandes de ses clients. Le service Web ou service HTTP est un service TCP-IP qui travaille habituellement sur le port 80. Il pourrait travailler sur un autre port. Dans ce cas, le navigateur client serait obligé de préciser ce port dans l'URL qu'il demande. Une URL a la forme générale suivante :

protocole://machine[:port]/chemin/infos

avec

protocole	http pour le service web. Un navigateur peut également servir de client à des services ftp, news, telnet, ..
machine	nom de la machine où officie le service web
port	port du service web. Si c'est 80, on peut omettre le n° du port. C'est le cas le plus fréquent
chemin	chemin désignant la ressource demandée
infos	informations complémentaires données au serveur pour préciser la demande du client

Que fait un navigateur lorsqu'un utilisateur demande le chargement d'une URL ?

1. il ouvre une communication TCP-IP avec la machine et le port indiqués dans la partie **machine[:port]** de l'URL. Ouvrir une communication TCP-IP, c'est créer un "tuyau" de communication entre deux machines. Une fois ce tuyau créé, toutes les informations échangées entre les deux machines vont passer dedans. La création de ce tuyau TCP-IP n'implique pas encore le protocole HTTP du Web.
2. le tuyau TCP-IP créé, le client va faire sa demande au serveur Web et il va la faire en lui envoyant des lignes de texte (des commandes) au format HTTP. Il va envoyer au serveur la partie **chemin/infos** de l'URL
3. le serveur lui répondra de la même façon et dans le même tuyau
4. l'un des deux partenaires prendra la décision de fermer le tuyau. Cela dépend du protocole HTTP utilisé. Avec le protocole HTTP 1.0, le serveur ferme la connexion après chacune de ses réponses. Cela oblige un client qui doit faire plusieurs demandes pour obtenir les différents documents constituant une page web à ouvrir une nouvelle connexion à chaque demande, ce qui a un coût. Avec le protocole HTTP/1.1, le client peut dire au serveur de garder la connexion ouverte jusqu'à ce qu'il lui dise de la fermer. Il peut donc récupérer tous les documents d'une page web avec une seule connexion et fermer lui-même la connexion une fois le dernier document obtenu. Le serveur détectera cette fermeture et fermera lui aussi la connexion.

Pour découvrir les échanges entre un client et un serveur web, nous allons utiliser un outil appelé **curl**. Curl est une application DOS permettant d'être client de services Internet supportant différents protocoles (HTTP, FTP, TELNET, GOPHER, ...). curl est disponible à l'URL <http://curl.haxx.se/>. On téléchargera ici de préférence la version Windows win32-openssl, la version win32-ssl nécessitant des dll supplémentaires non délivrées dans le paquetage curl. Celui-ci contient un ensemble de fichiers qu'il suffit de décompresser dans un dossier que nous appellerons désormais <curl>. Ce dossier contient un exécutable appelé [curl.exe]. Ce sera notre client pour interroger des serveurs web. Ouvrons une fenêtre Dos et plaçons-nous dans le dossier <curl> :

```
dos>dir curl.exe
22/03/2004 13:29          299 008 curl.exe

E:\curl2>curl
curl: try 'curl --help' for more information
```

```
dos>curl --help | more
```

```
Usage: curl [options...] <url>
```

```
Options: (H) means HTTP/HTTPS only, (F) means FTP only
```

```
-a/--append          Append to target file when uploading (F)
```

```
-A/--user-agent <string> User-Agent to send to server (H)
```

```
--anyauth           Tell curl to choose authentication method (H)
```

```
-b/--cookie <name=string/file> Cookie string or file to read cookies from (H)
```

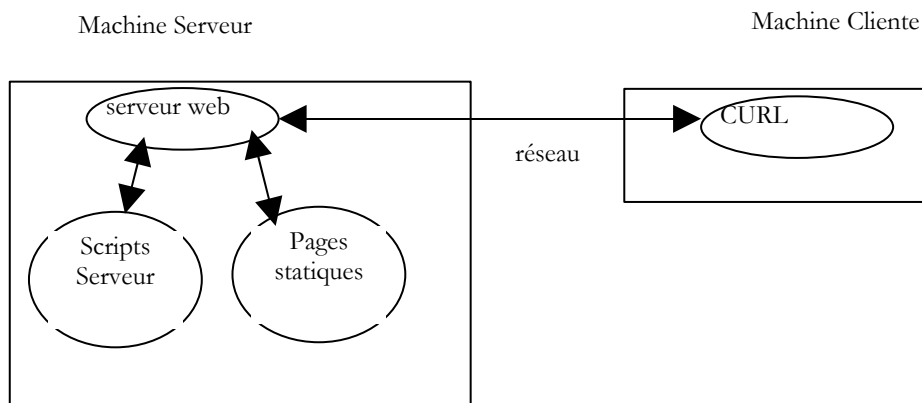
```
--basic            Enable HTTP Basic Authentication (H)
```

```
-B/--use-ascii     Use ASCII/text transfer
```

```
-c/--cookie-jar <file> Write cookies to this file after operation (H)
```

```
....
```

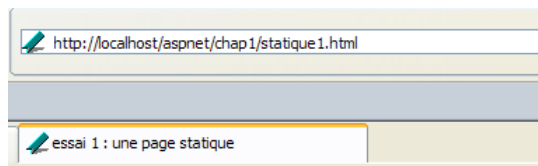
Utilisons cette application pour interroger un serveur web et découvrir les échanges entre le client et le serveur. Nous serons dans la situation suivante :



Le serveur web pourra être quelconque. Nous cherchons ici à découvrir les échanges qui vont se produire entre le client web [curl] et le serveur web. Précédemment, nous avons créé la page HTML statique suivante :

```
<html>
<head>
  <title>essai 1 : une page statique</title>
</head>
<body>
  <center>
    <h1>Une page statique...</h1>
  </body>
</html>
```

que nous visualisons dans un navigateur :



Une page statique...

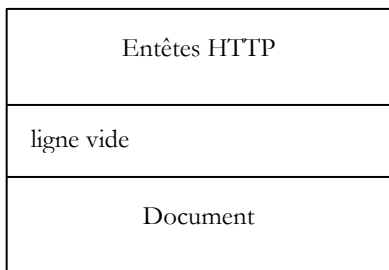
On voit que l'URL demandée est : *http://localhost/aspnet/chap1/statique1.html*. La machine du service web est donc *localhost* (=machine locale) et le port 80. Si on demande à voir le texte HTML de cette page Web (Affichage/Source) on retrouve le texte HTML initialement créé :


```
essai[1] - Bloc-notes
Fichier Edition Format ?
<html>
<head>
<title>essai 1 : une page statique</title>
</head>
<body>
<center>
<h1>Une page statique...</h1>
</body>
</html>
```

Maintenant utilisons notre client CURL pour demander la même URL :

```
dos>curl http://localhost/aspnet/chap1/statique1.html
<html>
<head>
<title>essai 1 : une page statique</title>
</head>
<body>
<center>
<h1>Une page statique...</h1>
</body>
</html>
```

Nous voyons que le serveur web lui a envoyé un ensemble de lignes de texte qui représente le code HTML de la page demandée. Nous avons dit précédemment que la réponse d'un serveur web se faisait sous la forme :



Or ici, nous n'avons pas vu les entêtes HTTP. Ceci parce que [curl] par défaut ne les affiche pas. L'option **--include** permet de les afficher :

```
E:\curl2>curl --include http://localhost/aspnet/chap1/statique1.html
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Mon, 22 Mar 2004 16:51:00 GMT
X-AspNet-Version: 1.1.4322
Cache-Control: public
ETag: "1C4102CEE8C6400:1C4102CFBBE2250"
Content-Type: text/html
Content-Length: 161
Connection: Close

<html>
<head>
<title>essai 1 : une page statique</title>
</head>
<body>
<center>
<h1>Une page statique...</h1>
</body>
</html>
```

Le serveur a bien envoyé une série d'entêtes HTTP suivie d'une ligne vide :

```
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Mon, 22 Mar 2004 16:51:00 GMT
X-AspNet-Version: 1.1.4322
Cache-Control: public
ETag: "1C4102CEE8C6400:1C4102CFBBE2250"
Content-Type: text/html
Content-Length: 161
Connection: Close
```

HTTP/1.1 200 OK le serveur dit

- qu'il comprend le protocole HTTP version 1.1
- qu'il a la ressource demandée (code 200, message OK)

Server: le serveur s'identifie. Ici c'est un serveur Cassini
 Date: ... la date/heure de la réponse
 X-ASPNet-Version: ... entête spécifique au serveur Cassini
 Cache-Control: public donne des indication au client sur la possibilité de mettre en cache la réponse qui lui est envoyée. L'attribut [public] indique au client qu'il peut mettre la page en cache. Un attribut [no-cache] aurait indiqué au client qu'il ne devait pas mettre en cache la page.
 ETag: ...
 Content-type: text/html le serveur dit qu'il va envoyer du texte (text) au format HTML (html).
 Content-Length: 161 nombre de bytes du document qui va être envoyé après les entêtes HTTP. Ce nombre est en fait la taille en octets du fichier *essai1.html*:

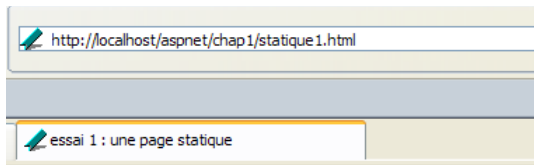
```
dos>dir essai1.html
08/07/2002 10:00                161 essai1.html
```

Connection: close le serveur dit qu'il fermera la connexion une fois le document envoyé

Le client reçoit ces entêtes HTTP et sait maintenant qu'il va recevoir 161 octets représentant un document HTML. Le serveur envoie ces 161 octets immédiatement derrière la ligne vide qui signalait la fin des entêtes HTTP :

```
<html>
<head>
  <title>essai 1 : une page statique</title>
</head>
<body>
  <center>
    <h1>Une page statique...</h1>
  </body>
</html>
```

On reconnaît là, le fichier HTML construit initialement. Si notre client était un navigateur, après réception de ces lignes de texte, il les interpréterait pour présenter à l'utilisateur au clavier la page suivante :

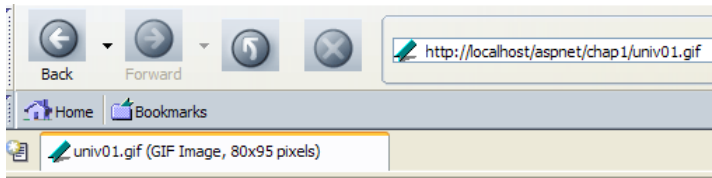


Une page statique...

Utilisons une nouvelle fois notre client [curl] pour demander la même ressource mais cette fois-ci en demandant seulement les entêtes de la réponse avec l'option **--head** :

```
dos>curl --head http://localhost/aspnet/chap1/statique1.html
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Tue, 23 Mar 2004 07:11:54 GMT
Cache-Control: public
ETag: "1C410A504D60680:1C410A58621AD3E"
Content-Type: text/html
Content-Length: 161
Connection: Close
```

Nous obtenons le même résultat que précédemment sans le document HTML. Maintenant demandons une image aussi bien avec un navigateur qu'avec le client TCP générique. Tout d'abord avec un navigateur :



Le fichier *univ01.gif* a 4052 octets :

```
dos>dir univ01.gif
23/03/2004 08:14          4 052 univ01.gif
```

Utilisons maintenant le client [curl] :

```
dos>curl --head http://localhost/aspnet/chap1/univ01.gif
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Tue, 23 Mar 2004 07:18:44 GMT
Cache-Control: public
ETag: "1C410A6795D7500:1C410A6868B1476"
Content-Type: image/gif
Content-Length: 4052
Connection: Close
```

On notera les points suivants dans le cycle demande- réponse ci-dessus :

--head

Content-Length: 4052

Content-Type: image/gif

- nous ne demandons que les entêtes HTTP de la ressource. En effet, une image est un fichier binaire et non un fichier texte et son affichage à l'écran en tant que texte ne donne rien de lisible.
- c'est la taille du fichier *univ01.gif*
- le serveur indique à son client qu'il va lui envoyer un document de type *image/gif*, c.a.d. une image au format GIF. Si l'image avait été au format JPEG, le type du document aurait été *image/jpeg*. Les types des documents sont standardisés et sont appelés des types MIME (Multi-purpose Internet Mail Extension).

1.5.3.2 La demande d'un client HTTP

Maintenant, posons-nous la question suivante : si nous voulons écrire un programme qui "parle" à un serveur web, quelles commandes doit-il envoyer au serveur web pour obtenir une ressource donnée ? Nous avons dans les exemples précédents vu ce que recevait le client mais pas ce que le client envoyait. Nous allons employer l'option [--verbose] de curl pour voir également ce qu'envoie le client au serveur. Commençons par demander la page statique :

```
dos>curl --verbose http://localhost/aspnet/chap1/statique1.html
* About to connect() to localhost:80
* Connected to portable1_tahc (127.0.0.1) port 80
> GET /aspnet/chap1/statique1.html HTTP/1.1
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4
Host: localhost
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

< HTTP/1.1 200 OK
< Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
< Date: Tue, 23 Mar 2004 07:37:06 GMT
< Cache-Control: public
< ETag: "1C410A504D60680:1C410A58621AD3E"
< Content-Type: text/html
< Content-Length: 161
< Connection: Close
<html>
<head>
  <title>essai 1 : une page statique</title>
</head>
<body>
```

```
<center>
  <h1>Une page statique...</h1>
</body>
</html>
* Closing connection #0
```

Tout d'abord, le client [curl] établit une connexion tcp/ip avec le port 80 de la machine localhost (=127.0.0.1)

```
* About to connect() to localhost:80
* Connected to portable1_tahc (127.0.0.1) port 80
```

Une fois la connexion établie, il envoie sa demande HTTP. Celle-ci est une suite de lignes de texte terminée par une ligne vide :

```
GET /aspnet/chap1/statique1.html HTTP/1.1
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4
Host: localhost
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

La requête HTTP d'un client Web a deux fonctions :

- indiquer la ressource désirée. C'est le rôle ici de la 1ère ligne GET
- donner des informations sur le client qui fait la requête pour que le serveur puisse éventuellement adapter sa réponse à ce type de client particulier.

La signification des lignes envoyées ci-dessus par le client [curl] est la suivante :

GET ressource protocole	pour demander une ressource donnée selon une version donnée du protocole HTTP. Le serveur envoie une réponse au format HTTP suivie d'une ligne vide suivie de la ressource demandée
User-Agent	pour indiquer qui est le client
host: machine:port	pour préciser (protocole HTTP 1.1) la machine et le port du serveur web interrogé
Pargma	ici pour préciser que le client ne gère pas de cache.
Accept	types MIME précisant les types de fichiers que le client sait gérer

Recommençons l'opération avec l'option --head de [curl] :

```
dos>curl --verbose --head --output reponse.txt http://localhost/aspnet/chap1/statique1.html
* About to connect() to localhost:80
* Connected to portable1_tahc (127.0.0.1) port 80
> HEAD /aspnet/chap1/statique1.html HTTP/1.1
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4
Host: localhost
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

< HTTP/1.1 200 OK
< Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
< Date: Tue, 23 Mar 2004 07:54:22 GMT
< Cache-Control: public
< ETag: "1C410A504D60680:1C410A58621AD3E"
< Content-Type: text/html
< Content-Length: 161
< Connection: Close
```

Nous ne nous attardons que sur les entêtes HTTP envoyés par le client :

```
HEAD /aspnet/chap1/statique1.html HTTP/1.1
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4
Host: localhost
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

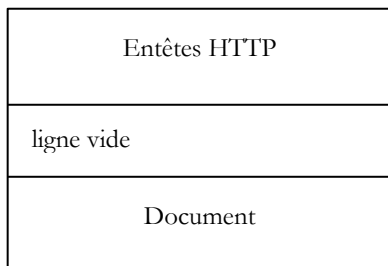
Seule la commande demandant la ressource a changé. Au lieu d'une commande GET on a maintenant une commande HEAD. Cette commande demande à ce que la réponse du serveur se limite aux entêtes HTTP et qu'il n'envoie pas la ressource demandée. La copie écran ci-dessus n'affiche pas les entêtes HTTP reçus. Ceux-ci ont été placés dans un fichier à cause de l'option [--output reponse.txt] de la commande [curl] :

```
dos>more reponse.txt
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Tue, 23 Mar 2004 07:54:22 GMT
```

```
Cache-Control: public
ETag: "1C410A504D60680:1C410A58621AD3E"
Content-Type: text/html
Content-Length: 161
Connection: Close
```

1.5.4 Conclusion

Nous avons découvert la structure de la demande d'un client web et celle de la réponse qui lui est faite par le serveur web sur quelques exemples. Le dialogue se fait à l'aide du protocole HTTP, un ensemble de commandes au format texte échangées par les deux partenaires. La requête du client et la réponse du serveur ont la même structure suivante :



Dans le cas d'une demande (appelée souvent requête) du client, la partie [Document] est le plus souvent absente. Néanmoins, il est possible pour un client d'envoyer un document au serveur. Il le fait avec une commande appelée PUT. Les deux commandes usuelles pour demander une ressource sont GET et POST. Cette dernière sera présentée un peu plus loin. La commande HEAD permet de demander seulement les entêtes HTTP. Les commandes GET et POST sont les plus utilisées par les clients web de type navigateur.

A la demande d'un client, le serveur envoie une réponse qui a la même structure. La ressource demandée est transmise dans la partie [Document] sauf si la commande du client était HEAD, auquel cas seuls les entêtes HTTP sont envoyés.

1.6 Le langage HTML

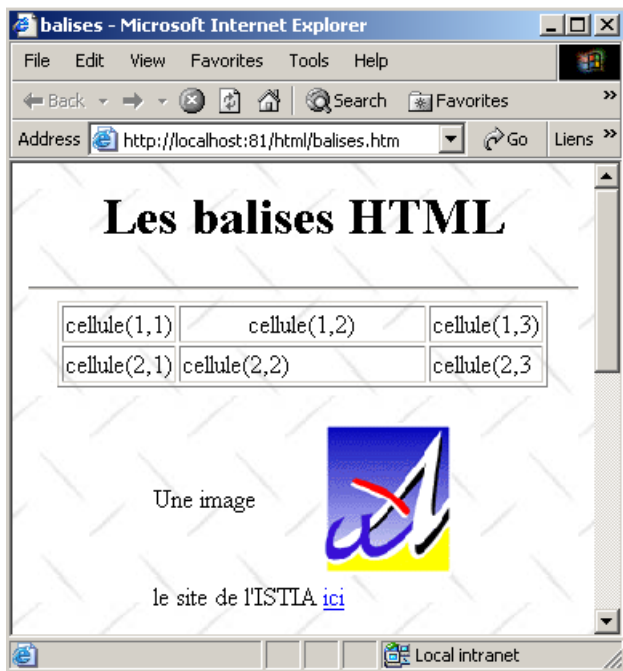
Un navigateur Web peut afficher divers documents, le plus courant étant le document HTML (HyperText Markup Language). Celui-ci est un texte formaté avec des balises de la forme `<balise>texte</balise>`. Ainsi le texte `important` affichera le texte important en gras. Il existe des balises seules telles que la balise `
` qui affiche une ligne horizontale. Nous ne passerons pas en revue les balises que l'on peut trouver dans un texte HTML. Il existe de nombreux logiciels WYSIWYG permettant de construire une page web sans écrire une ligne de code HTML. Ces outils génèrent automatiquement le code HTML d'une mise en page faite à l'aide de la souris et de contrôles prédéfinis. On peut ainsi insérer (avec la souris) dans la page un tableau puis consulter le code HTML généré par le logiciel pour découvrir les balises à utiliser pour définir un tableau dans une page Web. Ce n'est pas plus compliqué que cela. Par ailleurs, la connaissance du langage HTML est indispensable puisque les applications web dynamiques doivent générer elles-mêmes le code HTML à envoyer aux clients web. Ce code est généré par programme et il faut bien sûr savoir ce qu'il faut générer pour que le client ait la page web qu'il désire.

Pour résumer, il n'est nul besoin de connaître la totalité du langage HTML pour démarrer la programmation Web. Cependant cette connaissance est nécessaire et peut être acquise au travers de l'utilisation de logiciels WYSIWYG de construction de pages Web tels que Word, FrontPage, DreamWeaver et des dizaines d'autres. Une autre façon de découvrir les subtilités du langage HTML est de parcourir le web et d'afficher le code source des pages qui présentent des caractéristiques intéressantes et encore inconnues pour vous.

1.6.1 Un exemple

Considérons l'exemple suivant, créé avec FrontPage Express, un outil gratuit livré avec Internet Explorer. Le code généré par Frontpage a été ici épuré. Cet exemple présente quelques éléments qu'on peut trouver dans un document web tels que :

- un tableau
- une image
- un lien



Un document HTML a la forme générale suivante :

```
<html>
  <head>
    <title>Un titre</title>
    ...
  </head>
  <body attributs>
    ...
  </body>
</html>
```

L'ensemble du document est encadré par les balises **<html>...</html>**. Il est formé de deux parties :

1. **<head>...</head>** : c'est la partie non affichable du document. Elle donne des renseignements au navigateur qui va afficher le document. On y trouve souvent la balise **<title>...</title>** qui fixe le texte qui sera affiché dans la barre de titre du navigateur. On peut y trouver d'autres balises notamment des balises définissant les mots clés du document, mot clés utilisés ensuite par les moteurs de recherche. On peut trouver également dans cette partie des scripts, écrits le plus souvent en javascript ou vbscript et qui seront exécutés par le navigateur.
2. **<body attributs>...</body>** : c'est la partie qui sera affichée par le navigateur. Les balises HTML contenues dans cette partie indiquent au navigateur la forme visuelle "souhaitée" pour le document. Chaque navigateur va interpréter ces balises à sa façon. Deux navigateurs peuvent alors visualiser différemment un même document web. C'est généralement l'un des casse-têtes des concepteurs web.

Le code HTML de notre document exemple est le suivant :

```
<html>
  <head>
    <title>balises</title>
  </head>
  <body background="/images/standard.jpg">
    <center>
      <h1>Les balises HTML</h1>
      <hr>
    </center>
    <table border="1">
      <tr>
        <td>cellule (1,1)</td>
        <td valign="middle" align="center" width="150">cellule (1,2)</td>
        <td>cellule (1,3)</td>
      </tr>
      <tr>
        <td>cellule (2,1)</td>
        <td>cellule (2,2)</td>
        <td>cellule (2,3)</td>
      </tr>
    </table>
  </body>
</html>
```

```

</tr>
</table>

<table border="0">
  <tr>
    <td>Une image</td>
    <td></td>
  </tr>
  <tr>
    <td>le site de l'ISTIA</td>
    <td><a href="http://istia.univ-angers.fr">ici</a></td>
  </tr>
</table>
</body>
</html>

```

Ont été mis en relief dans le code les seuls points qui nous intéressent :

Elément	balises et exemples HTML
titre du document	<code><title>balises</title></code> <i>balises</i> apparaîtra dans la barre de titre du navigateur qui affichera le document
barre horizontale	<code>
</code> : affiche un trait horizontal
tableau	<code><table attributs>...</table></code> : pour définir le tableau <code><tr attributs>...</tr></code> : pour définir une ligne <code><td attributs>...</td></code> : pour définir une cellule exemples : <code><table border="1">...</table></code> : l'attribut <code>border</code> définit l'épaisseur de la bordure du tableau <code><td valign="middle" align="center" width="150">cellule(1,2)</td></code> : définit une cellule dont le contenu sera cellule(1,2). Ce contenu sera centré verticalement (<code>valign="middle"</code>) et horizontalement (<code>align="center"</code>). La cellule aura une largeur de 150 pixels (<code>width="150"</code>)
image	<code></code> : définit une image sans bordure (<code>border=0</code>), de hauteur 95 pixels (<code>height="95"</code>), de largeur 80 pixels (<code>width="80"</code>) et dont le fichier source est <code>/images/univ01.gif</code> sur le serveur web (<code>src="/images/univ01.gif"</code>). Ce lien se trouve sur un document web qui a été obtenu avec l'URL <code>http://localhost:81/html/balises.htm</code> . Aussi, le navigateur demandera-t-il l'URL <code>http://localhost:81/images/univ01.gif</code> pour avoir l'image référencée ici.
lien	<code>ici</code> : fait que le texte <i>ici</i> sert de lien vers l'URL <code>http://istia.univ-angers.fr</code> .
fond de page	<code><body background="/images/standard.jpg"></code> : indique que l'image qui doit servir de fond de page se trouve à l'URL <code>/images/standard.jpg</code> du serveur web. Dans le contexte de notre exemple, le navigateur demandera l'URL <code>http://localhost:81/images/standard.jpg</code> pour obtenir cette image de fond.

On voit dans ce simple exemple que pour construire l'intégralité du document, le navigateur doit faire trois requêtes au serveur :

1. `http://localhost:81/html/balises.htm` pour avoir le source HTML du document
2. `http://localhost:81/images/univ01.gif` pour avoir l'image `univ01.gif`
3. `http://localhost:81/images/standard.jpg` pour obtenir l'image de fond `standard.jpg`

L'exemple suivant présente un formulaire Web créé lui aussi avec FrontPage.

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="unMotDePasse"/>
Boîte de saisie	<div style="border: 1px solid black; padding: 2px;">ligne1 ligne2</div>
combo	<input type="text" value="choix2"/>
liste à choix simple	<div style="border: 1px solid black; padding: 2px;">liste1 liste2 liste3</div>
liste à choix multiple	<div style="border: 1px solid black; padding: 2px;">liste1 liste2 liste3</div>
bouton	<input type="button" value="Effacer"/>
envoyer	<input type="button" value="Envoyer"/>
rétablir	<input type="button" value="Rétablir"/>

Le code HTML généré par FrontPage et un peu épuré est le suivant :

```

<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" >
      <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>
      </table>
    </form>
  </body>
</html>

```



```

        <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>liste1</option>
                <option>liste2</option>
                <option selected>liste3</option>
                <option>liste4</option>
                <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>rétablir</td>
        <td>
            <input type="reset" value="Rétablir" name="cmdRétablir">
        </td>
    </tr>
</table>
<input type="hidden" name="secret" value="uneValeur">
</form>
</body>
</html>

```

L'association contrôle visuel <--> balise HTML est le suivant :

Contrôle	balise HTML
formulaire	<form method="POST" >
champ saisie	<input type="text" name="txtSaisie" size="20" value="qqz mots">
champ saisie cachée	<input type="password" name="txtMdp" size="20" value="unMotDePasse">
champ saisie multilignes	<textarea rows="2" name="areaSaisie" cols="20"> ligne1 ligne2 ligne3 </textarea>
boutons radio	<input type="radio" value="Oui" name="R1">Oui <input type="radio" name="R1" value="non" checked>Non
cases à cocher	<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
```

```
Combo <select size="1" name="cmbValeurs">
  <option>choix1</option>
  <option selected>choix2</option>
  <option>choix3</option>
</select>
```

```
liste à
sélection unique <select size="3" name="lst1">
  <option selected>liste1</option>
  <option>liste2</option>
  <option>liste3</option>
  <option>liste4</option>
  <option>liste5</option>
</select>
```

```
liste à
sélection multiple <select size="3" name="lst2" multiple>
  <option>liste1</option>
  <option>liste2</option>
  <option selected>liste3</option>
  <option>liste4</option>
  <option>liste5</option>
</select>
```

```
bouton de
type submit <input type="submit" value="Envoyer" name="cmdRenvoyer">
```

```
bouton de
type reset <input type="reset" value="Rétablir" name="cmdRétablir">
```

```
bouton de
type button <input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()">
```

Passons en revue ces différents contrôles.

1.6.1.1 Le formulaire

```
formulaire <form method="POST" >
```

```
balise HTML <form name="..." method="..." action="...">...</form>
```

```
attributs name="frmexemple" : nom du formulaire
method="..." : méthode utilisée par le navigateur pour envoyer au serveur web les valeurs récoltées dans le formulaire
action="..." : URL à laquelle seront envoyées les valeurs récoltées dans le formulaire.
```

Un formulaire web est entouré des balises `<form>...</form>`. Le formulaire peut avoir un nom (`name="xx"`). C'est le cas pour tous les contrôles qu'on peut trouver dans un formulaire. Ce nom est utile si le document web contient des scripts qui doivent référencer des éléments du formulaire. Le but d'un formulaire est de rassembler des informations données par l'utilisateur au clavier/souris et d'envoyer celles-ci à une URL de serveur web. Laquelle ? Celle référencée dans l'attribut `action="URL"`. Si cet attribut est absent, les informations seront envoyées à l'URL du document dans lequel se trouve le formulaire. Ce serait le cas dans l'exemple ci-dessus. Jusqu'à maintenant, nous avons toujours vu le client web comme "demandant" des informations à un serveur web, jamais lui "donnant" des informations. Comment un client web fait-il pour donner des informations (celles contenues dans le formulaire) à un serveur web ? Nous y reviendrons dans le détail un peu plus loin. Il peut utiliser deux méthodes différentes appelées POST et GET. L'attribut `method="méthode"`, avec méthode égale à GET ou POST, de la balise `<form>` indique au navigateur la méthode à utiliser pour envoyer les informations recueillies dans le formulaire à l'URL précisée par l'attribut `action="URL"`. Lorsque l'attribut `method` n'est pas précisé, c'est la méthode GET qui est prise par défaut.

1.6.1.2 Champ de saisie

```
Champ de saisie <input type="text" value="qqs mots" />
```

Mot de passe

champ de saisie

```
<input type="text" name="txtSaisie" size="20" value="qqs mots">  
<input type="password" name="txtMdp" size="20" value="unMotDePasse">
```

balise HTML

```
<input type="..." name="..." size=".." value="..">
```

La balise **input** existe pour divers contrôles. C'est l'attribut *type* qui permet de différencier ces différents contrôles entre eux.

attributs

type="text" : précise que c'est un champ de saisie

type="password" : les caractères présents dans le champ de saisie sont remplacés par des caractères *. C'est la seule différence avec le champ de saisie normal. Ce type de contrôle convient pour la saisie des mots de passe.

size="20" : nombre de caractères visibles dans le champ - n'empêche pas la saisie de davantage de caractères

name="txtSaisie" : nom du contrôle

value="qqs mots" : texte qui sera affiché dans le champ de saisie.

1.6.1.3 Champ de saisie multilignes

Boîte de saisie

champ de saisie multilignes

```
<textarea rows="2" name="areaSaisie" cols="20">  
  ligne1  
  ligne2  
  ligne3  
</textarea>
```

balise HTML

```
<textarea ...>texte</textarea>
```

affiche une zone de saisie multilignes avec au départ **texte** dedans

attributs

rows="2" : nombre de lignes

cols="20" : nombre de colonnes

name="areaSaisie" : nom du contrôle

1.6.1.4 Boutons radio

Etes-vous marié(e) Oui Non

boutons radio

```
<input type="radio" value="Oui" name="R1">Oui  
<input type="radio" name="R1" value="non" checked="">Non
```

balise HTML

```
<input type="radio" attribut2="valeur2" ....>texte
```

affiche un bouton radio avec **texte** à côté.

attributs

name="radio" : nom du contrôle. Les boutons radio portant le même nom forment un groupe de boutons exclusifs les uns des autres : on ne peut cocher que l'un d'eux.

value="valeur" : valeur affectée au bouton radio. Il ne faut pas confondre cette valeur avec le texte affiché à côté du bouton radio. Celui-ci n'est destiné qu'à l'affichage.

checked : si ce mot clé est présent, le bouton radio est coché, sinon il ne l'est pas.

1.6.1.5 Cases à cocher

cases à cocher

```
<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
```

Cases à cocher

 1 2 3

balise HTML `<input type="checkbox" attribut2="valeur2" ...>texte`

affiche une case à cocher avec **texte** à côté.

attributs

name="C1" : nom du contrôle. Les cases à cocher peuvent porter ou non le même nom. Les cases portant le même nom forment un groupe de cases associées.

value="valeur" : valeur affectée à la case à cocher. Il ne faut pas confondre cette valeur avec le texte affiché à côté du bouton radio. Celui-ci n'est destiné qu'à l'affichage.

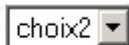
checked : si ce mot clé est présent, le bouton radio est coché, sinon il ne l'est pas.

1.6.1.6 Liste déroulante (combo)

Combo

```
<select size="1" name="cmbValeurs">
  <option>choix1</option>
  <option selected>choix2</option>
  <option>choix3</option>
</select>
```

combo



balise HTML

```
<select size=".." name="..">
  <option [selected]>...</option>
```

```
...
</select>
```

affiche dans une liste les textes compris entre les balises `<option>...</option>`

attributs

name="cmbValeurs" : nom du contrôle.

size="1" : nombre d'éléments de liste visibles. `size="1"` fait de la liste l'équivalent d'un combobox.

selected : si ce mot clé est présent pour un élément de liste, ce dernier apparaît sélectionné dans la liste. Dans notre exemple ci-dessus, l'élément de liste `choix2` apparaît comme l'élément sélectionné du combo lorsque celui-ci est affiché pour la première fois.

1.6.1.7 Liste à sélection unique

liste à
sélection
unique

```
<select size="3" name="lst1">
  <option selected>liste1</option>
  <option>liste2</option>
  <option>liste3</option>
  <option>liste4</option>
  <option>liste5</option>
</select>
```

liste à choix simple



balise HTML

```
<select size=".." name="..">
  <option [selected]>...</option>
```

```
...
</select>
```

affiche dans une liste les textes compris entre les balises `<option>...</option>`

attributs

les mêmes que pour la liste déroulante n'affichant qu'un élément. Ce contrôle ne diffère de la liste déroulante précédente que par son attribut `size>1`.

1.6.1.8 Liste à sélection multiple

liste à
sélection
unique

```
<select size="3" name="lst2" multiple>
  <option selected>liste1</option>
  <option>liste2</option>
```

```

<option selected>liste3</option>
<option>liste4</option>
<option>liste5</option>
</select>

```

liste à choix multiple



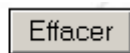
balise HTML `<select size=".." name=".." multiple>`
`<option [selected]>...</option>`
 ...
`</select>`

attributs **multiple** : permet la sélection de plusieurs éléments dans la liste. Dans l'exemple ci-dessus, les éléments *liste1* et *liste3* sont tous deux sélectionnés.

1.6.1.9 Bouton de type button

bouton de type button `<input type="button" value="Effacer" name="cmdEffacer" onclick="effacer()">`

bouton



balise HTML `<input type="button" value="..." name="..." onclick="effacer()">`

attributs **type="button"** : définit un contrôlé bouton. Il existe deux autres types de bouton, les types *submit* et *reset*.
value="Effacer" : le texte affiché sur le bouton
onclick="fonction()" : permet de définir une fonction à exécuter lorsque l'utilisateur clique sur le bouton. Cette fonction fait partie des scripts définis dans le document web affiché. La syntaxe précédente est une syntaxe *javascript*. Si les scripts sont écrits en *vbscript*, il faudrait écrire **onclick="fonction"** sans les parenthèses. La syntaxe devient identique s'il faut passer des paramètres à la fonction : **onclick="fonction(val1, val2,...)"**

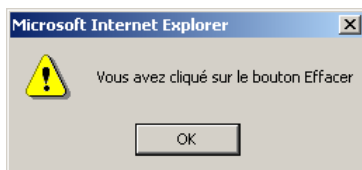
Dans notre exemple, un clic sur le bouton *Effacer* appelle la fonction javascript *effacer* suivante :

```

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

```

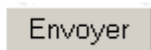
La fonction *effacer* affiche un message :



1.6.1.10 Bouton de type submit

bouton de type submit `<input type="submit" value="Envoyer" name="cmdRenvoyer">`

envoyer



balise HTML `<input type="submit" value="Envoyer" name="cmdRenvoyer">`

attributs **type="submit"** : définit le bouton comme un bouton d'envoi des données du formulaire au serveur web.

Lorsque le client va cliquer sur ce bouton, le navigateur va envoyer les données du formulaire à l'URL définie dans l'attribut **action** de la balise **<form>** selon la méthode définie par l'attribut **method** de cette même balise.

value="Envoyer" : le texte affiché sur le bouton

1.6.1.11 Bouton de type reset

bouton de type reset `<input type="reset" value="Rétablir" name="cmdRétablir">`



balise HTML `<input type="reset" value="Rétablir" name="cmdRétablir">`

attributs **type="reset"** : définit le bouton comme un bouton de réinitialisation du formulaire. Lorsque le client va cliquer sur ce bouton, le navigateur va remettre le formulaire dans l'état où il l'a reçu.

value="Rétablir" : le texte affiché sur le bouton

1.6.1.12 Champ caché

champ caché `<input type="hidden" name="secret" value="uneValeur">`

balise HTML `<input type="hidden" name="..." value="...">`

attributs **type="hidden"** : précise que c'est un champ caché. Un champ caché fait partie du formulaire mais n'est pas présenté à l'utilisateur. Cependant, si celui-ci demandait à son navigateur l'affichage du code source, il verrait la présence de la balise `<input type="hidden" value="...">` et donc la valeur du champ caché.

value="uneValeur" : valeur du champ caché.

Quel est l'intérêt du champ caché ? Cela peut permettre au serveur web de garder des informations au fil des requêtes d'un client. Considérons une application d'achats sur le web. Le client achète un premier article *art1* en quantité *q1* sur une première page d'un catalogue puis passe à une nouvelle page du catalogue. Pour se souvenir que le client a acheté *q1* articles *art1*, le serveur peut mettre ces deux informations dans un champ caché du formulaire web de la nouvelle page. Sur cette nouvelle page, le client achète *q2* articles *art2*. Lorsque les données de ce second formulaire vont être envoyées au serveur (submit), celui-ci va non seulement recevoir l'information (*q2,art2*) mais aussi (*q1,art1*) qui fait partie également partie du formulaire en tant que champ caché non modifiable par l'utilisateur. Le serveur web va alors mettre dans un nouveau champ caché les informations (*q1,art1*) et (*q2,art2*) et envoyer une nouvelle page de catalogue. Et ainsi de suite.

1.6.2 Envoi à un serveur web par un client web des valeurs d'un formulaire

Nous avons dit dans l'étude précédente que le client web disposait de deux méthodes pour envoyer à un serveur web les valeurs d'un formulaire qu'il a affiché : les méthodes GET et POST. Voyons sur un exemple la différence entre les deux méthodes. La page étudiée précédemment est une page statique. Afin d'avoir accès aux entêtes HTTP envoyés par le navigateur qui va demander ce document, nous la transformons en page dynamique pour un serveur web .NET (IIS ou Cassini). Il ne s'agit pas ici de s'intéresser à la technologie .NET qui sera abordée dans le chapitre suivant mais aux échanges client-serveur. Le code de la page ASP.NET est la suivante :

```
<%@ Page Language="vb" CodeBehind="params.aspx.vb" AutoEventWireup="false"
Inherits="ConsoleApplication1.params" %>
<script runat="server">

Private Sub Page_Init(Byval Sender as Object, Byval e as System.EventArgs)
    ' on sauvegarde la requête
    saveRequest
end sub
Private Sub saveRequest
    ' sauve la requête courante dans request.txt du dossier de la page
    dim requestFileName as String=Me.MapPath(Me.TemplateSourceDirectory)+"\request.txt"
    Me.Request.SaveAs(requestFileName,true)
end sub
```

```

</script>

<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">
....
</body>
</html>

```

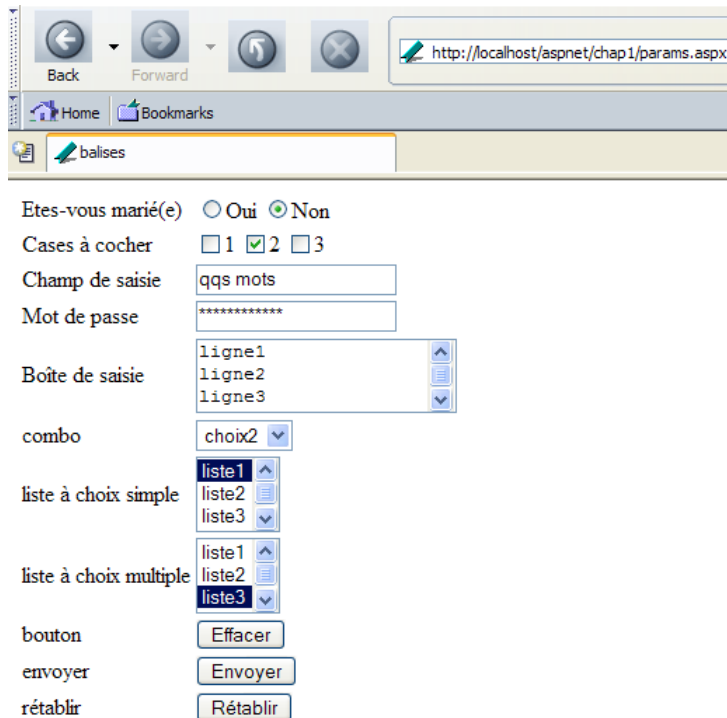
Au contenu HTML de la page étudiée, nous ajoutons une partie code en VB.NET. Nous ne commenterons pas ce code, si ce n'est pour dire qu'à chaque appel du document ci-dessus, le serveur web va sauvegarder la requête du client Web dans le fichier [request.txt] dans le dossier du document appelé.

1.6.2.1 Méthode GET

Faisons un premier test, où dans le code HTML du document, la balise FORM est définie de la façon suivante :

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

Le document précédent (HTML+code VB) est appelé [params.aspx]. Il est placé dans l'arborescence d'un serveur Web .NET (IIS/Cassini) et appelé avec l'url http://localhost/aspnet/chap1/params.aspx :



Le navigateur vient de faire une requête et nous savons que celle-ci a été enregistrée dans le fichier [request.txt]. Regardons le contenu de celui-ci :

```

GET /aspnet/chap1/params.aspx HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg
,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113

```

Nous retrouvons des éléments déjà rencontrés avec le client [curl]. D'autres apparaissent pour la première fois :

Connection: keep- le client demande au serveur de ne pas fermer la connexion après sa réponse. Cela lui permettra d'utiliser
 Les bases

alive	la même connexion pour une demande ultérieure. La connexion ne reste pas ouverte indéfiniment. Le serveur la ferme après un trop long délai d'inutilisation.
Keep-Alive	durée en secondes pendant laquelle la connexion [Keep-Alive] restera ouverte
Accept-Charset	Catégorie de caractères que le client sait gérer
Accept-Language	liste de langues préférées par le client.

Nous remplissons le formulaire est rempli de la façon suivante :

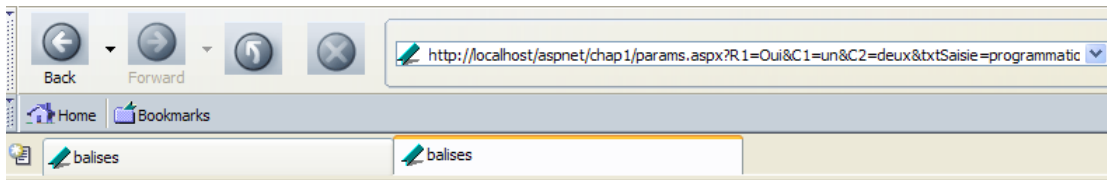
The screenshot shows a web browser window with the address bar containing `http://localhost/aspnet/chap1/params.aspx`. Below the browser, a form is displayed with the following elements:

- Radio buttons for "Etes-vous marié(e)" with "Oui" selected.
- Checkboxes for "Cases à cocher" with checkboxes 1 and 2 selected.
- A text input field containing "programmation asp.net".
- A password input field with masked characters "*****".
- A text area containing "les bases de la programmation web".
- A dropdown menu labeled "choix3" with "choix3" selected.
- A simple choice list labeled "liste à choix simple" with "liste3" selected.
- A multiple choice list labeled "liste à choix multiple" with "liste1", "liste2", and "liste3" selected.
- A button labeled "Effacer".
- A button labeled "Envoyer" which is highlighted with a mouse cursor.
- A button labeled "Rétablir".

Nous utilisons le bouton [Envoyer] ci-dessus. Son code HTML est le suivant :

```
<form method="get">
...
<input type="submit" value="Envoyer">
...
</form>
```

Sur l'activation d'un bouton de type [Submit], le navigateur envoie les paramètres du formulaire (balise `<form>`) à l'URL indiquée dans l'attribut [action] de la balise `<form action="URL">` s'il existe. Si cet attribut n'existe pas, les paramètres d formulaire sont envoyés à l'URL qui a délivré le formulaire. C'est le cas ici. Le bouton [Envoyer] devrait donc entraîner une requête du navigateur à l'URL [http://localhost/aspnet/chap1/params.aspx] avec un transfert des paramètres du formulaire. Comme la page [params.aspx] mémorise la requête reçue, nous devrions savoir comment le client a transféré ces paramètres. Essayons. Nous cliquons sur le bouton [Envoyer]. Nous recevons la réponse suivante du navigateur :



Etes-vous marié(e) Oui Non

Cases à cocher 1 2 3

Champ de saisie

Mot de passe

Boîte de saisie

combo

liste à choix simple

liste à choix multiple

bouton

envoyer

rétablir

C'est la page initiale mais on peut remarquer que l'URL a changé dans le champ [Adresse] du navigateur. Elle est devenue la suivante :

```
http://localhost/aspnet/chap1/params.aspx?R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecret&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web&cmbValeurs=choix3&lst1=liste3&lst2=liste1&lst3=liste3&cmdRenvoyer=Envoyer&secret=uneValeur
```

On constate que les choix faits dans le formulaire se retrouvent dans l'URL. Regardons le contenu du fichier [request.txt] qui a mémorisé la requête du client :

```
GET
/aspnet/chap1/params.aspx?R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecret&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web&cmbValeurs=choix3&lst1=liste3&lst2=liste1&lst3=liste3&cmdRenvoyer=Envoyer&secret=uneValeur HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-flash, text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png, image/jpeg, image/gif;q=0.2, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.7, */*;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/aspnet/chap1/params.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113
```

On retrouve une requête HTTP assez semblable à celle qui avait été faite initialement par le navigateur lorsqu'il avait demandé le document sans transmettre de paramètres. Il y a deux différences :

- GET URL HTTP/1.1** Les paramètres du formulaire ont été ajoutés derrière l'URL du document sous la forme ?param1=val1¶m2=val2&...
- Referer** le client indique par cet entête HTTP l'URL du document qu'il affichait lorsqu'il a fait la requête

Examinons de plus près comment les paramètres ont été passés dans la commande *GET URL?param1=valeur1¶m2=valeur2&... HTTP/1.1* où les *parami* sont les noms des contrôles du formulaire web et *valeuri* les valeurs qui leur sont associées. Nous présentons ci-dessous un tableau à trois colonnes :

- colonne 1 : reprend la définition d'un contrôle HTML de l'exemple
- colonne 2 : donne l'affichage de ce contrôle dans un navigateur
- colonne 3 : donne la valeur envoyée au serveur par le navigateur pour le contrôle de la colonne 1 sous la forme qu'elle a dans la requête GET de l'exemple

contrôle HTML	visuel	valeur(s) renvoyée(s)
---------------	--------	-----------------------

```
<input type="radio" value="Oui"
name="R1">Oui
<input type="radio" name="R1"
value="non" checked>Non
```

Etes-vous marié(e) Oui Non

R1=Oui

- la valeur de l'attribut *value* du bouton radio coché par l'utilisateur.

```
<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
```

Cases à cocher 1 2 3

**C1=un
C2=deux**

- valeurs des attributs *value* des cases cochées par l'utilisateur

```
<input type="text" name="txtSaisie"
size="20" value="qq mots">
```

Champ de saisie

txtSaisie=programmation+web

- texte tapé par l'utilisateur dans le champ de saisie. Les espaces ont été remplacés par le signe +

```
<input type="password" name="txtMdp"
size="20" value="unMotDePasse">
```

Mot de passe

txtMdp=ceciestsecret

- texte tapé par l'utilisateur dans le champ de saisie

```
<textarea rows="2" name="areaSaisie"
cols="20">
ligne1
ligne2
ligne3
</textarea>
```

Boîte de saisie

**areaSaisie=les+bases+de+la%0D%0A
programmation+web**

- texte tapé par l'utilisateur dans le champ de saisie. %OD%OA est la marque de fin de ligne. Les espaces ont été remplacés par le signe +

```
<select size="1" name="cmbValeurs">
<option>choix1</option>
<option selected>choix2</option>
<option>choix3</option>
</select>
```

combo

cmbValeurs=choix3

- valeur choisie par l'utilisateur dans la liste à sélection unique

```
<select size="3" name="lst1">
<option selected>liste1</option>
<option>liste2</option>
<option>liste3</option>
<option>liste4</option>
<option>liste5</option>
</select>
```

liste à choix simple

lst1=liste3

- valeur choisie par l'utilisateur dans la liste à sélection unique

```
<select size="3" name="lst2" multiple>
<option selected>liste1</option>
<option>liste2</option>
<option selected>liste3</option>
<option>liste4</option>
<option>liste5</option>
</select>
```

liste à choix multiple

**lst2=liste1
lst2=liste3**

- valeurs choisies par l'utilisateur dans la liste à sélection multiple

```
<input type="submit" value="Envoyer"
name="cmdRenvoyer">
```

cmdRenvoyer=Envoyer

- nom et attribut *value* du bouton qui a servi à envoyer les données du formulaire au serveur

```
<input type="hidden" name="secret"
value="uneValeur">
```

secret=uneValeur

- attribut *value* du champ caché

On peut se demander ce que le serveur a fait des paramètres qu'on lui a passés. En réalité rien. A la réception de la commande

```
GET
/aspnet/chap1/params.aspx?R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecret&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web&cmbValeurs=choix3&lst1=liste3&lst2=liste1&lst2=liste3&cmdRenvoyer=Envoyer&secret=uneValeur HTTP/1.1
```

Le serveur web a transmis les paramètres à l'URL au document <http://localhost/aspnet/chap1/params.aspx>, c.a.d. au document que nous avons construit initialement. Nous n'avons écrit aucun code pour récupérer et traiter les paramètres que le client nous envoie. Aussi tout se passe comme si la requête du client était simplement :

```
GET /aspnet/chap1/params.aspx
```

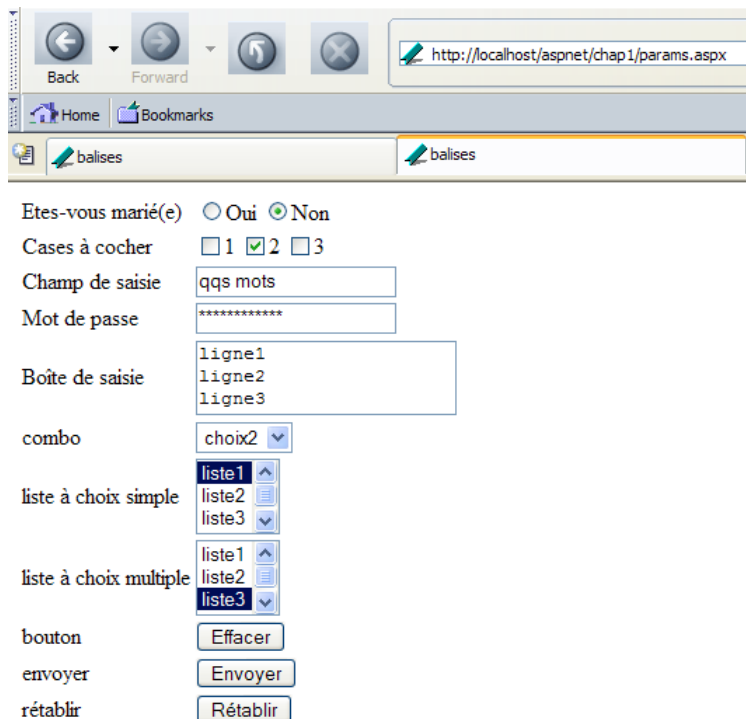
C'est pour cette raison, qu'en réponse à notre bouton [Envoyer], nous avons obtenu la même page que celle obtenue initialement en demandant l'URL [http://localhost/aspnet/chap1/params.aspx] sans paramètres.

1.6.2.2 Méthode POST

Le document HTML est maintenant programmé pour que le navigateur utilise maintenant la méthode POST pour envoyer les valeurs du formulaire au serveur web :

```
<form method="POST" >
```

Nous demandons le nouveau document via l'URL [http://localhost/aspnet/chap1/params.aspx], nous remplissons le formulaire tel que pour la méthode GET et nous transmettons les paramètres au serveur avec le bouton [Envoyer]. Nous obtenons du serveur la page réponse suivante :



The screenshot shows a web browser window with the address bar containing 'http://localhost/aspnet/chap1/params.aspx'. Below the browser window, a form is displayed with the following elements:

- Radio buttons for 'Etes-vous marié(e)' with 'Oui' and 'Non' options.
- Checkboxes for 'Cases à cocher' with values 1, 2, and 3.
- A text input field labeled 'Champ de saisie' containing 'qqs mots'.
- A password input field labeled 'Mot de passe' with masked characters.
- A multi-line text area labeled 'Boîte de saisie' containing 'ligne1', 'ligne2', and 'ligne3'.
- A dropdown menu labeled 'combo' with 'choix2' selected.
- A simple list box labeled 'liste à choix simple' with 'liste1', 'liste2', and 'liste3' options.
- A multiple list box labeled 'liste à choix multiple' with 'liste1', 'liste2', and 'liste3' options.
- A button labeled 'Effacer'.
- A button labeled 'Envoyer'.
- A button labeled 'Rétablir'.

Nous obtenons donc le même résultat qu'avec la méthode GET, c.a.d. la page initiale. On remarquera une différence : dans le champ [Adresse] du navigateur, les paramètres transmis n'apparaissent pas. Maintenant, regardons la requête envoyée par le client et qui a été mémorisée dans le fichier [request.txt] :

```
POST /aspnet/chap1/params.aspx HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Content-Length: 210
Content-Type: application/x-www-form-urlencoded
Accept: application/x-shockwave-flash,text/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/aspnet/chap1/params.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113

R1=Oui&C1=un&C2=deux&txtSaisie=programmation+web&txtMdp=ceciestsecrey&areaSaisie=les+bases+de+la%0D%0Apr
ogrammation+web&cmbValeurs=choix3&lst1=liste3&lst2=liste1&lst3=liste3&cmdRenvoyer=Envoyer&secret=uneVale
ur
```

Des nouveautés apparaissent dans la requête HTTP du client :

POST URL HTTP/1.1 la requête GET a laissé place à une requête POST. Les paramètres ne sont plus présents dans cette première ligne de la requête. On peut constater qu'ils sont maintenant placés derrière la requête HTTP après une ligne vide. Leur encodage est identique à celui qu'ils avaient dans la requête GET.

<code>Content-Length</code>	nombre de caractères "postés", c.a.d. le nombre de caractères que devra lire le serveur web après avoir reçu les entêtes HTTP pour récupérer le document que lui envoie le client. Le document en question est ici la liste des valeurs du formulaire.
<code>Content-type</code>	précise le type du document que le client enverra après les entêtes HTTP. Le type [application/x-www-form-urlencoded] indique que c'est un document contenant des valeurs de formulaire.

Il y a deux méthodes pour transmettre des données à un serveur web : GET et POST. Y-a-t-il une méthode meilleure que l'autre ? Nous avons vu que si les valeurs d'un formulaire étaient envoyées par le navigateur avec la méthode GET, le navigateur affichait dans son champ *Adresse* l'URL demandée sous la forme `URL?param1=val1¶m2=val2&...`. On peut voir cela comme un avantage ou un inconvénient :

- un avantage si on veut permettre à l'utilisateur de placer cette URL paramétrée dans ses liens favoris
- un inconvénient si on ne souhaite pas que l'utilisateur ait accès à certaines informations du formulaire tels, par exemple, les champs cachés

Par la suite, nous utiliserons quasi exclusivement la méthode POST dans nos formulaires.

1.7 Conclusion

Ce chapitre a présenté différents concepts de base du développement web :

- les différents outils et technologies disponibles (java, asp, asp.net, php, perl, vbscript, javascript)
- les échanges client-serveur via le protocole HTTP
- la conception d'un document à l'aide du langage HTML
- la conception de formulaires de saisie

Nous avons pu voir sur un exemple comment un client pouvait envoyer des informations au serveur web. Nous n'avons pas présenté comment le serveur pouvait

- récupérer ces informations
- les traiter
- envoyer au client une réponse dynamique dépendant du résultat du traitement

C'est le domaine de la programmation web, domaine que nous abordons dans le chapitre suivant avec la présentation de la technologie ASP.NET.

2 Introduction au développement web ASP.NET

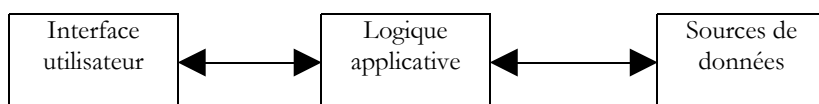
2.1 Introduction

Le chapitre précédent a présenté les principes du développement web qui sont indépendants du langage de programmation utilisé. Actuellement, trois technologies dominent le marché du développement web :

- J2EE qui est une plate-forme de développement Java. Associée à la technologie Struts, installée sur différents serveurs d'application, la plate-forme J2EE est utilisée principalement dans les grands projets. Du fait du langage utilisé - Java - une application J2EE peut fonctionner sur les principaux systèmes d'exploitation (Windows, Unix, Linux, Mac OS, ...)
- PHP qui est un langage interprété, lui aussi indépendant du système d'exploitation. Contrairement à Java, ce n'est pas un langage orienté objet. Cependant la version PHP5 devrait introduire l'objet dans le langage. D'accès simple, PHP est largement utilisé dans les petits et moyens projets.
- ASP.NET qui est une technologie ne fonctionnant que sur les machines Windows disposant de la plate-forme .NET (XP, 2000, 2003, ...). Le langage de développement utilisé peut être tout langage compatible .NET, c.a.d. plus d'une dizaine, à commencer par les langages de Microsoft (C#, VB.NET, J#), Delphi de Borland, Perl, Python, ...

Le chapitre précédent a présenté de courts exemples pour chacune de ces trois technologies. Ce document s'intéresse au développement Web ASP.NET avec le langage VB.NET. Nous supposons que ce langage est connu. Ce point est important. Nous nous intéressons ici uniquement à son utilisation dans le contexte du développement web. Explicitons davantage ce point en parlant de la méthodologie MVC de développement web.

Une application web respectant le modèle MVC sera architecturée de la façon suivante :

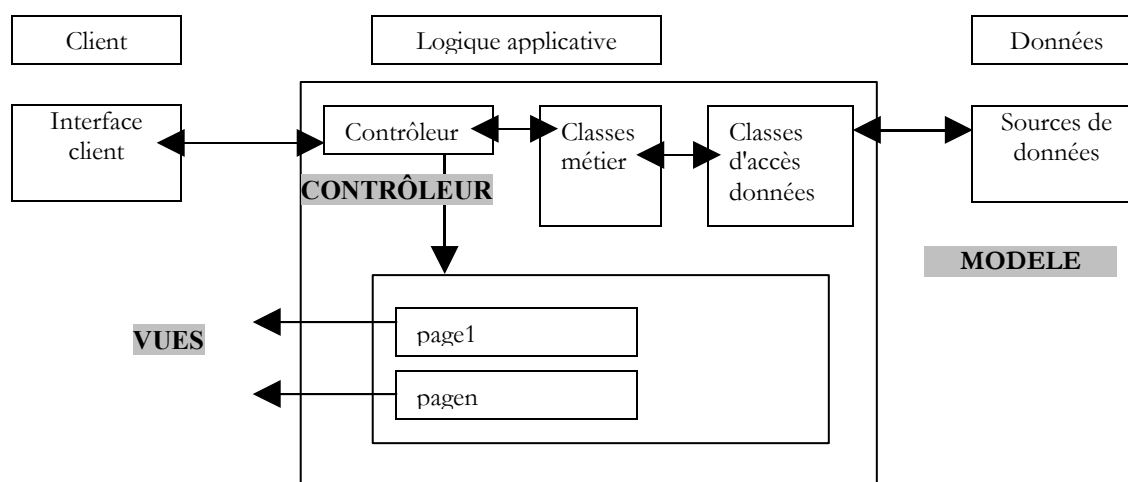


Une telle architecture, appelée 3-tiers ou à 3 niveaux, cherche à respecter le modèle MVC (Model View Controller) :

- l'interface utilisateur est le V (la vue)
- la logique applicative est le C (le contrôleur)
- les sources de données sont le M (Modèle)

L'**interface utilisateur** est souvent 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. La **source de données** est souvent une base de données mais cela peut être aussi de simples fichiers plats, un annuaire LDAP, 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.

- On mettra la logique métier de l'application dans des classes séparées de la classe qui contrôle le dialogue demande-réponse. Ainsi le bloc [Logique applicative] ci-dessus pourra être constitué des éléments suivants :



Dans le bloc [Logique Applicative], on pourra distinguer

- la classe contrôleur qui est la porte d'entrée de l'application.
- le bloc [Classes métier] qui regroupe des classes nécessaires à la logique de l'application. Elles sont indépendantes du client.
- le bloc [Classes d'accès aux données] qui regroupe des classes nécessaires pour obtenir les données nécessaires à la servlet, souvent des données persistantes (BD, fichiers, service WEB, ...)
- le bloc des pages ASP constituant les vues de l'application.

Dans les cas simples, la logique applicative est souvent réduite à deux classes :

- la classe contrôleur assurant le dialogue client-serveur : traitement de la requête, génération des diverses réponses
- la classe métier qui reçoit du contrôleur des données à traiter et lui fournit en retour des résultats. Cette classe métier gère alors elle-même l'accès aux données persistantes.

La spécificité du développement web repose sur l'écriture de la classe contrôleur et des pages de présentation. Les classes métier et d'accès aux données sont des classes .NET classiques, utilisables aussi bien dans une application web que dans une application windows ou même de type console. L'écriture de ces classes nécessite de bonnes connaissances de la programmation objet. Dans ce document, elles seront écrites en VB.NET, aussi supposons-nous que ce langage est maîtrisé. Dans cette perspective, il n'y a pas lieu de s'apesantir plus que nécessaire sur le code d'accès aux données. Dans la quasi totalité des livres sur ASP.NET, un chapitre est consacré à ADO.NET. Le schéma ci-dessus montre que l'accès aux données est fait par des classes .NET tout à fait classiques qui ignorent qu'elles sont utilisées dans un contexte web. Le contrôleur, qui est le chef d'équipe de l'application web, n'a pas à se soucier d'ADO.NET. Il doit savoir simplement à quelle classe il doit demander les données dont il a besoin et comment les demander. C'est tout. Mettre du code ADO.NET dans le contrôleur n'est pas conforme au concept MVC expliqué ci-dessus et nous ne le ferons pas.

2.2 Les outils

Ce document est destiné à des étudiants, aussi allons-nous travailler avec des outils gratuits téléchargeables sur internet :

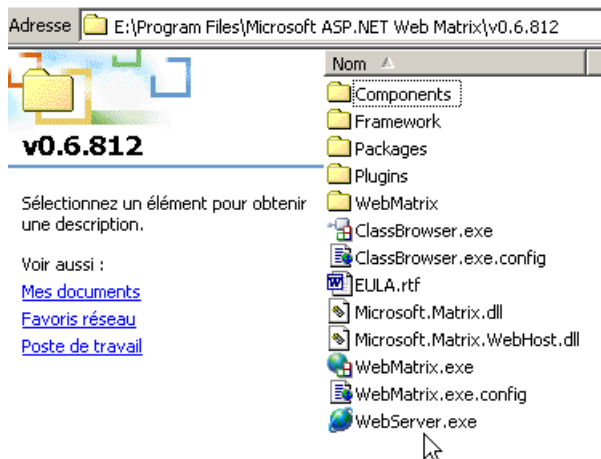
- la plate-forme .NET (compilateurs, documentation)
- l'environnement de développement WebMatrix qui amène avec lui le serveur web Cassini
- différents SGBD (MSDE, MySQL)

Le lecteur est invité à consulter l'annexe "Outils du web" qui indique où trouver et comment installer ces différents outils. La plupart du temps, nous n'aurons besoin que de trois outils :

- un éditeur de texte pour écrire les applications web.
- un outil de développement VB.NET pour écrire le code VB lorsque celui-ci est important. Ce type d'outil offre en général une aide à la saisie de code (complétion automatique de code), le signalement des erreurs syntaxiques soit dès la frappe du code, soit lors de la sa compilation.
- un serveur web pour tester les applications web écrites. On prendra Cassini dans ce document. Le lecteur disposant du serveur IIS pourra remplacer Cassini par IIS. Ils sont tous les deux compatibles .NET. Cassini est cependant limité pour répondre aux seules demandes locales (localhost) alors que IIS peut répondre aux requêtes de machines externes.

Un excellent environnement commercial pour développer en VB.NET est Visual Studio.NET de Microsoft. Cet IDE très riche permet de gérer toutes sortes de documents (code VB.NET, documents HTML, XML, feuilles de style, ...). Pour l'écriture de code, il apporte l'aide précieuse de la "complétion" automatique de code. Ceci dit, cet outil qui améliore sensiblement la productivité du développeur a le défaut de ses qualités : il enferme le développeur dans un mode de développement standard, certes efficace mais pas toujours approprié.

Il est possible d'utiliser le serveur Cassini en-dehors de [WebMatrix] et c'est ce que nous ferons souvent. L'exécutable du serveur se trouve dans <WebMatrix>\<version>\WebServer.exe où <WebMatrix> est le répertoire d'installation de [WebMatrix] et <version> son n° de version :

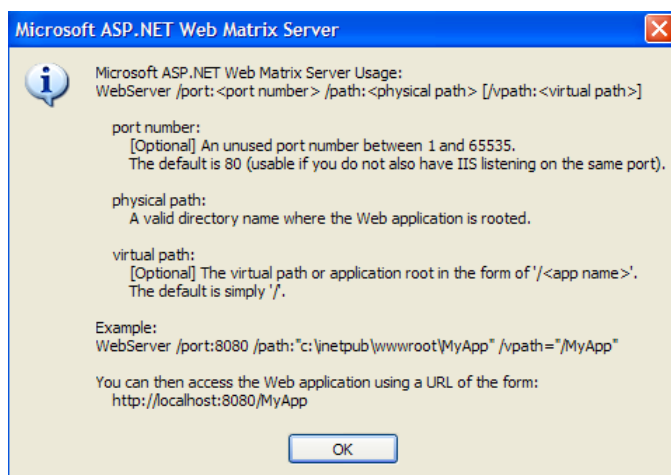


Ouvrons une fenêtre Dos et positionnons-nous dans le dossier du serveur Cassini :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>dir
...
29/05/2003  11:00                53 248 WebServer.exe
...
```

Lançons [WebServer.exe] sans paramètres :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>webserver
```



Le panneau ci-dessus nous indique que l'application [WebServer/Cassini] admet trois paramètres :

- **/port** : n° de port du service web. Peut-être quelconque. A par défaut la valeur 80
- **/path** : chemin physique d'un dossier du disque
- **/vpath** : dossier virtuel associé au dossier physique précédent. La syntaxe est **/vpath:chemin** et non **/vpath=chemin** comme indiqué dans la fenêtre Cassini ci-dessus.

Nous placerons nos exemples dans une arborescence de fichiers de racine P avec des dossiers chap1, chap2, ... pour les différents chapitres de ce document. Nous associerons à ce dossier physique P, le chemin virtuel V. Aussi lancerons-nous Cassini avec la commande Dos suivante :

```
dos> WebServer /port:80 /path:P /vpath:V
```

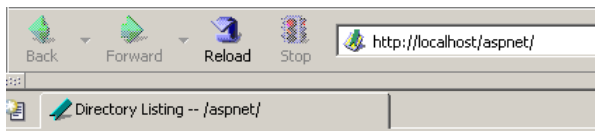
Par exemple, si on veut que la racine physique du serveur soit le dossier [D:\data\devel\aspnet\poly] et sa racine virtuelle [aspnet], la commande Dos de lancement du serveur web sera :

```
dos> WebServer /port:80 /path:D:\data\devel\aspnet\poly /vpath:/aspnet
```

On pourra mettre cette commande dans un raccourci. Une fois lancé Cassini installe une icône dans la barre des tâches. En double-cliquant dessus, on a accès à un panneau Arrêt/Marche du serveur :



Le panneau rappelle les trois paramètres avec lesquels il a été lancé. Il offre deux boutons d'arrêt/marche ainsi qu'un lien de test vers la racine de son arborescence web. Nous le suivons. Un navigateur est ouvert et l'URL [http://localhost/aspnet] demandée. Nous obtenons le contenu du dossier indiqué dans le champ [Physical Path] ci-dessus :



Directory Listing -- /aspnet/

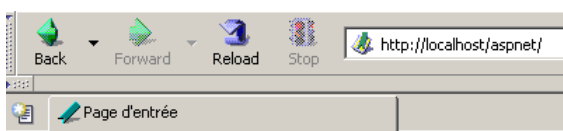
```
mardi, mars 23, 2004 05:19 <dir> chap1
mardi, mars 23, 2004 05:19 <dir> chap2
```

Dans l'exemple, l'URL demandée correspond à un dossier et non à un document web, aussi le serveur a-t-il affiché le contenu de ce dossier et non un document web particulier. Si dans ce dossier, il existe un fichier appelé [default.aspx], celui-ci sera visualisé. Construisons par exemple, le fichier suivant et plaçons-le dans la racine de l'arborescence web de Cassini (d:\data\devel\aspnet\poly ici) :

```
<html>
<head>
  <title>Page d'entrée</title>
</head>
<body>
  Page d'index...
</body>
</html>
```

```
dos>dir d:\data\devel\aspnet\poly\default.aspx
23/03/2004  18:21                107 default.aspx
```

Demandons maintenant l'URL [http://localhost/aspnet] avec un navigateur :



Page d'index...

On voit qu'en réalité c'est l'URL [http://localhost/aspnet/default.aspx] qui a été affichée. Dans la suite du document, nous indiquerons comment Cassini doit être configuré par la notation Cassini(path,vpath) où [path] est le nom du dossier racine de l'arborescence web du serveur et [vpath] le chemin virtuel associé. On se rappellera qu'avec le serveur Cassini(path,vpath), l'url [http://localhost/vpath/XX] correspond au chemin physique [path\XX]. Nous placerons tous nos documents sous une racine physique que nous appellerons <webroot>. Ainsi nous pourrions parler du fichier <webroot>\chap2\here1.aspx. Pour chaque lecteur cette racine <webroot> sera un dossier de sa machine personnelle. Ici les copies d'écran montreront que ce dossier est souvent [d:\data\devel\aspnet\poly]. Ce ne sera cependant pas toujours le cas, les tests ayant été faits sur des machines différentes.

2.3 Premiers exemples

Nous allons présenter des exemples simples de page web dynamique créée avec VB.NET. Le lecteur est invité à les tester afin de vérifier que son environnement de développement est correctement installé. Nous allons découvrir qu'il y a plusieurs façons de construire une page ASP.NET. Nous en choisirons une pour la suite de nos développements.

2.3.1 Exemple de base - variante 1

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

Nous reprenons l'exemple du chapitre précédent. Nous construisons le fichier [heure1.aspx] suivant :

```
<html>
<head>
  <title>Demo asp.net </title>
</head>
<body>
  Il est <% =Date.Now.ToString("T") %>
</body>
</html>
```

Ce code est du code HTML avec une balise spéciale <% ... %>. A l'intérieur de cette balise, on peut mettre du code VB.NET. Ici le code

```
Date.Now.ToString("T")
```

produit une chaîne de caractères C représentant l'heure du moment. La balise <% ... %> est alors remplacée par cette chaîne de caractères C. Ainsi si C est la chaîne 18:11:01, la ligne HTML contenant le code VB.NET devient :

```
| Il est 18:11:01 |
```

Plaçons le code précédent dans le fichier [<webroot>\chap2\heure1.aspx]. Lançons Cassini(<webroot>, /aspnet) et demandons avec un navigateur l'URL [http://localhost/aspnet/chap2/heure1.aspx] :



Il est 18:31:21

Une fois obtenu ce résultat, nous savons que l'environnement de développement est correctement installé. La page [heure1.aspx] a été compilée puisqu'elle contient du code VB.NET. Sa compilation a produit un fichier dll qui a été stocké dans un dossier système puis exécuté par le serveur Cassini.

2.3.2 Exemple de base - variante 2

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

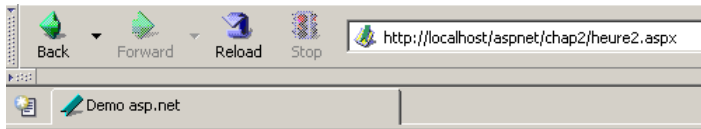
La document [heure1.aspx] mélange code HTML et code VB.NET. Dans un exemple aussi simple, cela ne pose pas problème. Si on est amené à inclure davantage de code VB.NET, on souhaitera séparer davantage le code HTML du code VB. Cela peut se faire en regroupant le code VB à l'intérieur d'une balise <script> :

```
<script runat="server">
  ' calcul des données à afficher par le code HTML
  ...
</script>
<html>
  ....
  ' affichage des valeurs calculées par la partie script
</html>
```

L'exemple [heure2.aspx] montre cette méthode :

```
<script runat="server">
  Dim maintenant as String=Date.Now.ToString("T")
</script>
<html>
<head>
  <title>Demo asp.net </title>
</head>
<body>
  Il est
  <% =maintenant %>
</body>
</html>
```

Nous plaçons le document [heure2.aspx] dans l'arborescence [<webroot>\chap2\heure2.aspx] du serveur web Cassini(<webroot>,/aspnet) et demandons le document avec un navigateur :



Il est 09:39:32

2.3.3 Exemple de base - variante 3

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

Nous poussons plus loin la démarche de séparation du code VB et du code HTML en les mettant dans deux fichiers séparés. Le code HTML sera dans le document [heure3.aspx] et le code VB dans [heure3.aspx.vb]. Le contenu de [heure3.aspx] sera le suivant :

```
<%@ Page Language="vb" src="heure3.aspx.vb" Inherits="heure3" %>
<html>
<head>
<title>Demo asp.net</title>
</head>
<body>
  Il est
  <% =maintenant %>
</body>
</html>
```

Il y a deux différences fondamentales :

- la directive [Page] avec des attributs encore inconnus
- l'utilisation de la variable [maintenant] dans le code HTML alors qu'elle n'est initialisée nulle part

La directive [Page] sert ici à indiquer que le code VB qui va initialiser la page est dans un autre fichier. C'est l'attribut [src] qui indique ce dernier. Nous allons découvrir que le code VB est celui d'une classe qui s'appelle [heure3]. De façon transparente pour le développeur, un fichier .aspx est transformé en classe dérivant d'une classe de base appelée [Page]. Ici, notre document HTML doit dériver de la classe qui définit et calcule les données qu'il doit afficher. Ici c'est la classe [heure3] définie dans le fichier [heure3.aspx.vb]. Aussi doit-on indiquer ce lien parent-fils entre le document VB [heure3.aspx.vb] et le document HTML [heure3.aspx]. C'est l'attribut [inherits] qui précise ce lien. Il doit indiquer le nom de la classe définie dans le fichier pointé par l'attribut [src].

Étudions maintenant le code VB de la page [heure3.aspx.vb] :

```
Public Class heure3
  Inherits System.Web.UI.Page

  ' données de la page web à afficher
  Protected maintenant As String

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'on calcule les données de la page web
    maintenant = Date.Now.ToString("T")
  End Sub
End Class
```

On notera les points suivants :

- le code VB définit une classe [heure3] dérivée de la classe [System.Web.UI.Page]. C'est toujours le cas, une page web devant toujours dériver de [System.Web.UI.Page].
- la classe déclare un attribut protégé (protected) [maintenant]. On sait qu'un attribut protégé est accessible directement dans les classes dérivées. C'est ce qui permet au document HTML [heure3.aspx] d'avoir accès à la valeur de la donnée [maintenant] dans son code.
- l'initialisation de l'attribut [maintenant] se fait dans une procédure [Page_Load]. Nous verrons ultérieurement qu'un objet de type [Page] est averti par le serveur Web d'un certain nombre d'événements. L'événement [Load] se produit lorsque l'objet [Page] et ses composants ont été créés. Le gestionnaire de cet événement est désigné par la directive [Handles MyBase.Load]

```
Private Sub XX(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
```

- le nom [XX] du gestionnaire de l'événement peut être quelconque. Sa signature doit elle être celle indiquée ci-dessus. Nous n'expliquerons pas celle-ci pour l'instant.

- on utilise souvent le gestionnaire de l'événement [Page.Load] pour calculer les valeurs des données dynamiques que doit afficher la page web.

Les documents [heure3.spx] et [heure3.aspx.vb] sont placés dans [<webroot>\chap2]. Puis avec un navigateur, on demande l'URL [http://localhost/aspnet/chap2/heure3.aspx] au serveur web(<webroot>,/aspnet) :



Il est 14:22:34

2.3.4 Exemple de base - variante 4

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

Nous gardons le même exemple que précédemment mais regroupons de nouveau tout le code dans un unique fichier [heure4.aspx] :

```
<script runat="server">
' données de la page web à afficher
Private maintenant As String

' evt page_load
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
'on calcule les données de la page web
maintenant = Date.Now.ToString("T")
End Sub
</script>

<html>
<head>
<title>Demo asp.net</title>
</head>
<body>
Il est
<% =maintenant %>
</body>
</html>
```

Nous retrouvons la séquence de l'exemple 2 :

```
<script runat="server">
.... code VB
</script>

<html>
... code HTML
</html>
```

Cette fois-ci, le code VB a été structuré en procédures. On retrouve la procédure [Page_Load] de l'exemple précédent. On veut montrer ici, qu'une page .aspx seule (non liée à un code VB dans un fichier séparé) est transformée de façon implicite en classe dérivée de [Page]. Aussi peut-on utiliser les attributs, méthodes et événements de cette classe. C'est ce qui est fait ici où on utilise l'événement [Load] de cette classe.

La méthode de test est identique aux précédentes :



Il est 14:31:51

2.3.5 Exemple de base - variante 5

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

Comme dans l'exemple 3, on sépare code VB et code HTML dans deux fichiers séparés. Le code VB est placé dans [heure5.aspx.vb] :

```

Public Class heure5
    Inherits System.Web.UI.Page

    ' données de la page web à afficher
    Protected maintenant As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'on calcule les données de la page web
        maintenant = Date.Now.ToString("T")
    End Sub
End Class

```

Le code HTML est placé dans [heure5.aspx] :

```

<%@ Page Inherits="heure5" %>
<html>
<head>
<title>Demo asp.net</title>
</head>
<body>
    Il est
    <% =maintenant %>
</body>
</html>

```

Cette fois-ci, la directive [Page] n'indique plus le lien entre le code HTML et le code VB. Le serveur web n'a plus la possibilité de retrouver le code VB pour le compiler (absence de l'attribut src). C'est à nous de faire cette compilation. Dans une fenêtre dos, nous compilons donc la classe VB [heure5.aspx.vb] :

```

dos>dir
23/03/2004  18:34                133 heure1.aspx
24/03/2004  09:47                232 heure2.aspx
24/03/2004  10:16                183 heure3.aspx
24/03/2004  10:16                332 heure3.aspx.vb
24/03/2004  14:31                440 heure4.aspx
24/03/2004  14:45                332 heure5.aspx.vb
24/03/2004  14:56                148 heure5.aspx

```

```

dos>vbc /r:system.dll /r:system.web.dll /t:library /out:heure5.dll heure5.aspx.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4

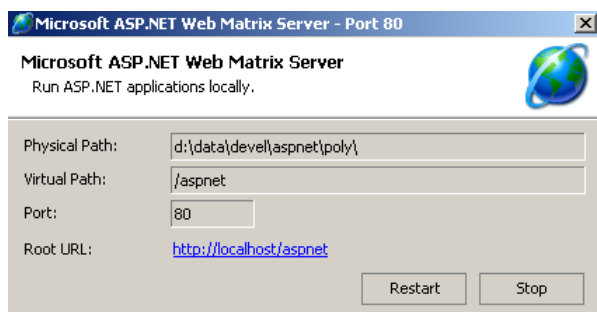
```

```

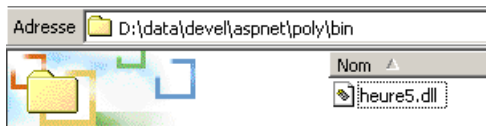
dos>dir heure5.dll
24/03/2004  14:51                3 072 heure5.dll

```

Ci-dessus, l'exécutable [vbc.exe] du compilateur était dans le PATH de la machine Dos. S'il ne l'avait pas été, il aurait fallu donner le chemin complet de [vbc.exe] qui se trouve dans l'arborescence du dossier où a été installé SDK.NET. Les classes dérivées de [Page] nécessitent des ressources présentes dans les DLL [system.dll, system.web.dll], d'où la référence de celles-ci via l'option /r du compilateur. L'option /t :library est là pour indiquer qu'on veut produire une DLL. L'option /out indique le nom du fichier à produire, ici [heure5.dll]. Ce fichier contient la classe [heure5] dont a besoin le document web [heure5.aspx]. Seulement, le serveur web cherche les DLL dont il a besoin dans des endroits bien précis. L'un de ces endroits est le dossier [bin] situé à la racine de son arborescence. Cette racine est ce que nous avons appelé <webroot>. Pour le serveur IIS, c'est généralement <lecteur>:\inetpub\wwwroot où <lecteur> est le lecteur (C, D, ...) où a été installé IIS. Pour le serveur Cassini, cette racine correspond au paramètre /path avec lequel vous l'avez lancé. Rappelons que cette valeur peut être obtenue en double-cliquant sur l'icône du serveur dans la barre des tâches :



<webroot> correspond à l'attribut [Physical Path] ci-dessus. Nous créons donc un dossier <webroot>\bin et plaçons [heure5.dll] dedans :



Nous sommes prêts. Nous demandons l'URL [http://localhost/aspnet/chap2/heure5.aspx] au serveur Cassini(<webroot>,/aspnet) :



2.3.6 Exemple de base - variante 6

Outils nécessaires : un éditeur de texte, le serveur Web Cassini

Nous avons montré jusqu'ici qu'une application web dynamique avait deux composantes :

1. du code VB pour calculer les parties dynamiques de la page
2. du code HTML incluant parfois du code VB, pour l'affichage de ces valeurs dans la page. Cette partie représente la réponse qui est envoyée au client web.

La composante 1 est appelée la composante **contrôleur** de la page et la partie 2 la composante **présentation**. La partie **présentation** doit contenir le moins de code VB possible, voire pas de code VB du tout. On verra que c'est possible. Ici, nous montrons un exemple où il n'y a qu'un contrôleur et pas de composante présentation. C'est le contrôleur qui génère lui-même la réponse au client sans l'aide de la composante présentation.

Le code de présentation devient le suivant :

```
<%@ Page src="heure6.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="heure6" %>
```

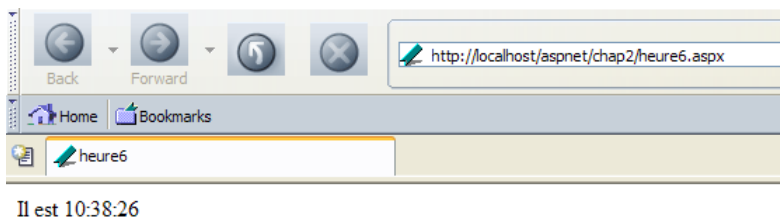
On voit qu'il n'y a plus aucun code HTML dedans. La réponse est élaborée directement dans le contrôleur :

```
Public Class heure6
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on élabore la réponse
        Dim HTML As String
        HTML = "<html><head><title>heure6</title></head><body>Il est "
        HTML += Date.Now.ToString("T")
        HTML += "</body></html>"
        ' on l'envoie
        Response.Write(HTML)
    End Sub
End Class
```

Le contrôleur élabore ici la totalité de la réponse plutôt que les seules parties dynamiques de celle-ci. De plus il l'envoie. Il le fait avec la propriété [Response] de type [HttpResponse] de la classe [Page]. C'est un objet qui représente la réponse faite par le serveur au client. La classe [HttpResponse] dispose d'une méthode [Write] pour écrire dans le flux HTML qui sera envoyé au client. Ici, nous mettons la totalité du flux HTML à envoyer dans la variable [HTML] et nous envoyons celle-ci au client par [Response.Write(HTML)].

Nous demandons l'url [http://localhost/aspnet/chap2/heure6.aspx] au serveur Cassini (<webroot>,/aspnet) :



2.3.7 Conclusion

Par la suite, nous utiliserons la méthode 3 qui place le code VB et le code HTML d'un document Web dynamique dans deux fichiers séparés. Cette méthode a l'avantage de découper une page web en deux composantes :

1. une composante **contrôleur** composée uniquement de code VB pour calculer les parties dynamiques de la page
2. une composante **présentation** qui est la réponse envoyée au client. Elle est composée de code HTML incluant parfois du code VB pour l'affichage des valeurs dynamiques. Nous viserons toujours à avoir le minimum de code VB dans la partie présentation, l'idéal étant de ne pas en avoir du tout.

Comme l'a montré la méthode 5, le contrôleur pourra être compilé indépendamment de l'application web. Cela présente l'avantage de se concentrer uniquement sur le code et d'avoir à chaque compilation la liste de toutes les erreurs. Une fois le contrôleur compilé, l'application web peut être testée. Sans compilation préalable, c'est le serveur web qui fera celle-ci, et alors les erreurs seront signalées une par une. Cela peut être jugé fastidieux.

Pour les exemples qui vont suivre, les outils suivants suffisent :

- un éditeur de texte pour construire les documents HTML et VB de l'application lorsqu'ils sont simples
- un IDE de développement .NET pour construire les classes VB.NET afin de bénéficier de l'aide apportée par ce type d'outil à l'écriture de code. Un tel outil est par exemple **CSharpDevelop** (<http://www.icsharpcode.net>). Un exemple d'utilisation est montré dans l'annexe [Les outils du développement web].
- l'outil **WebMatrix** pour construire les pages de présentation de l'application (cf l'annexe [Les outils du développement web]).
- le serveur **Cassini**

Tous ces outils sont gratuits.

3 Les fondamentaux du développement ASP.NET

3.1 La notion d'application web ASP.NET

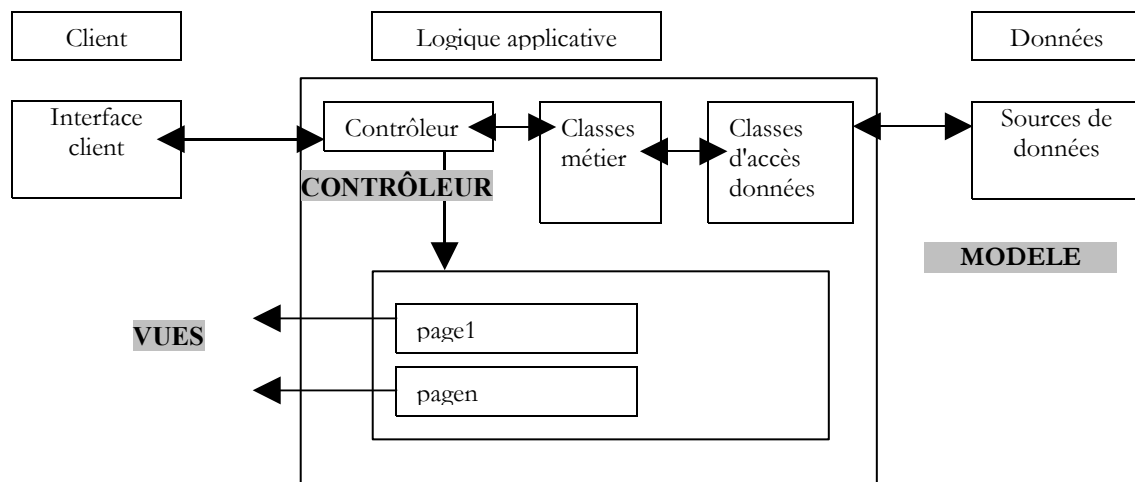
3.1.1 Introduction

Une application web est une application regroupant divers documents (HTML, code .NET, images, sons, ...). Ces documents doivent être sous une même racine qu'on appelle la racine de l'application web. A cette racine est associé un chemin virtuel du serveur web. Nous avons rencontré la notion de dossier virtuel pour le serveur web Cassini. Cette notion existe également pour le serveur web IIS. Une différence importante entre les deux serveurs est qu'à un moment donné, IIS peut avoir un nombre quelconque de dossiers virtuels alors que le serveur web Cassini n'en a qu'un, celui qui a été spécifié à son lancement. Cela signifie que le serveur IIS peut servir plusieurs applications web simultanément alors que le serveur Cassini n'en sert qu'une à la fois. Dans les exemples précédents, le serveur Cassini était toujours lancé avec les paramètres (<webroot>/aspnet) qui associaient le dossier virtuel /aspnet au dossier physique <webroot>. Le serveur web servait donc toujours la même application web. Cela ne nous a pas empêchés d'écrire et de tester des pages différentes et indépendantes à l'intérieur de cette unique application web. Chaque application web a des ressources qui lui sont propres et qui se trouvent sous sa racine physique <webroot> :

- un dossier [bin] dans lequel on peut placer des classes pré-compilées
- un fichier [global.asax] qui permet de d'initialiser l'application web dans son ensemble ainsi que l'environnement d'exécution de chacun de ses utilisateurs
- un fichier [web.config] qui permet de paramétrer le fonctionnement de l'application
- un fichier [default.aspx] qui joue le rôle de porte d'entrée de l'application
- ...

Dès qu'une application utilise l'une de ces trois ressources, elle a besoin d'un chemin physique et virtuel qui lui soient propres. Il n'y a en effet aucune raison pour que deux applications web différentes soient configurées de la même façon. Nos exemples précédents ont pu tous être placés dans la même application (<webroot>/aspnet) parce qu'ils n'utilisaient aucune des ressources précédentes.

Revenons sur l'architecture MVC préconisée dans le chapitre précédent, pour le développement d'une application web :



L'application web est formée des fichiers de classe (contrôleur, classes métier, classes d'accès aux données) et des fichiers de présentation (documents HTML, images, sons, feuilles de style,..). L'ensemble de ces fichiers sera placé sous une même racine que nous appellerons parfois <application-path>. Cette racine sera associée à un chemin virtuel <application-vpath>. L'association entre ce chemin virtuel et chemin physique se fait par configuration du serveur web. On a vu que pour le serveur Cassini, cette association se faisait au lancement du serveur. Par exemple dans une fenêtre dos, on lancerait Cassini par :

```
webservice.exe /port:80 /path:<application-path> /vpath:<application-vpath>
```

Dans le dossier <application-path>, on trouvera selon nos besoins :

- le dossier [bin] pour y placer des classes pré-compilées (dll)
- le fichier [global.asax] lorsque nous aurons besoin de faire des initialisation soit lors de l'initialisation de l'application, soit lors de celle d'une session utilisateur
- le fichier [web.config] lorsque nous aurons besoin de paramétrer l'application
- le fichier [default.aspx] lorsque nous aurons besoin d'une page par défaut dans l'application

Afin de respecter ce concept d'application web, les exemples à venir seront tous placés dans un dossier <application-path> propre à l'application auquel sera associé un dossier virtuel <application-vpath>, le serveur Cassini étant lancé de façon à lier ces deux paramètres.

3.1.2 Configurer une application web

Si <application-path> est la racine d'une application ASP.NET, on peut utiliser le fichier <application-path>\web.config pour configurer celle-ci. Ce fichier est au format XML. Voici un exemple :

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <appSettings>
    <add key="nom" value="tintin"/>
    <add key="age" value="27"/>
  </appSettings>
</configuration>
```

On prêtera attention au fait que les balises XML **sont sensibles à la casse**. Toutes les informations de configuration doivent être entre les balises <configuration> et </configuration>. Il existe de nombreuses sections de configuration utilisables. Nous n'en présentons qu'une ici, la section <appSettings> qui permet d'initialiser des données avec la balise <add>. La syntaxe de cette balise est la suivante :

```
<add key="identificateur" value="valeur"/>
```

Lorsque le serveur Web lance une application, il regarde si dans <application-path> il y a un fichier appelé [web.config]. Si oui, il le lit et mémorise ses informations dans un objet de type [ConfigurationSettings] qui sera disponible à toutes les pages de l'application tant que celle-ci est active. La classe [ConfigurationSettings] a une méthode statique [AppSettings] :

Obtient les paramètres de configuration contenus dans la section de configuration <appSettings>, élément.

```
Public Shared ReadOnly Property AppSettings As NameValueCollection
```

Valeur de propriété

[NameValueCollection](#) contenant des paires nom-valeur de paramètres de configuration.

Pour obtenir la valeur d'une clé C du fichier de configuration, on écrit [ConfigurationSettings.AppSettings("C")]. On obtient une chaîne de caractères. Pour exploiter le fichier de configuration précédent, créons une page [default.aspx]. Le code VB du fichier [default.aspx.vb] sera le suivant :

```
Imports System.Configuration

Public Class _default
  Inherits System.Web.UI.Page

  Protected nom As String
  Protected age As String

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'on récupère les informations de configuration
    nom = ConfigurationSettings.AppSettings("nom")
    age = ConfigurationSettings.AppSettings("age")
  End Sub
End Class
```

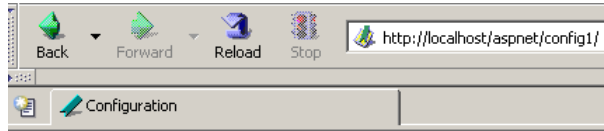
On voit qu'au chargement de la page, les valeurs des paramètres de configuration [nom] et [age] sont récupérées. Elles vont être affichées par le code de présentation de [default.aspx] :

```
<%@ Page src="default.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="_default" %>
<html>
<head>
  <title>Configuration</title>
</head>
<body>
  Nom :
  <% =nom %><br/>
  Age :
  <% =age %><br/>
</body>
</html>
```


Pour le test, on met les fichiers [web.config], [default.aspx] et [default.aspx.vb] dans le même dossier :

```
D:\data\devel\aspnet\poly\chap2\config1>dir
30/03/2004 15:06          418 default.aspx.vb
30/03/2004 14:57          236 default.aspx
30/03/2004 14:53          186 web.config
```

Soit <application-path> le dossier où se trouvent les trois fichiers de l'application. Le serveur Cassini est lancé avec les paramètres (<application-path>,/aspnet/config1). Nous demandons l'URL [http://localhost/aspnet/config1]. Comme [config1] est un dossier, le serveur web va chercher un fichier [default.aspx] dedans et l'afficher s'il le trouve. Ici, il va le trouver :



Nom : tintin
Age : 27

3.1.3 Application, Session, Contexte

3.1.3.1 Le fichier global.asax

Le code du fichier [global.asax] est toujours exécuté avant que la page demandée par la requête courante ne soit chargée. Il doit être situé dans la racine <application-path> de l'application. S'il existe, le fichier [global.asax] est utilisé à divers moments par le serveur web :

1. lorsque l'application web démarre ou se termine
2. lorsqu'une session utilisateur démarre ou se termine
3. lorsqu'une requête utilisateur démarre

Comme pour les pages .aspx, le fichier [global.asax] peut être écrit de différentes façons et en particulier en séparant code VB dans une classe contrôleur et code de présentation. C'est le choix fait par défaut par l'outil Visual Studio et nous allons faire ici de même. Il n'y a normalement aucune présentation à faire, ce rôle étant dévolu aux pages .aspx. Le contenu du fichier [global.asax] est alors réduit à une directive référençant le fichier contenant le code du contrôleur :

```
| <%@ Application src="Global.asax.vb" Inherits="Global" %> |
```

On remarquera que la directive n'est plus [Page] mais [Application]. Le code du contrôleur [global.asax.vb] associé et généré par l'outil Visual studio est le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque l'application est démarrée
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque la session est démarrée
    End Sub

    Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche au début de chaque demande
    End Sub

    Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lors d'une tentative d'authentification de l'utilisation
    End Sub

    Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsqu'une erreur se produit
    End Sub

    Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque la session se termine
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque l'application se termine
    End Sub
End Class
```

End Sub

End Class

On notera que la classe du contrôleur dérive de la classe [HttpApplication]. Dans la vie d'une application, il existe plusieurs événements importants. Ceux-ci sont gérés par des procédures dont le squelette est donné ci-dessus.

- [Application_Start] : rappelons qu'une application web est "enfermée" dans un chemin virtuel. L'application démarre dès qu'une page située dans ce chemin virtuel est demandée par un client. La procédure [Application_Start] est alors exécutée. Ce sera l'unique fois. On fera dans cette procédure, toute initialisation utile à l'application, comme par exemple créer des objets dont la durée de vie est celle de l'application.
- [Application_End] : est exécutée quand l'application est terminée. A toute application est associé un délai d'inactivité, configurable dans [web.config], au bout duquel l'application est considérée comme terminée. C'est donc le serveur web qui prend cette décision en fonction du paramétrage de l'application. Le délai d'inactivité d'une application est défini comme le temps pendant lequel aucun client n'a fait une demande pour une ressource de l'application.
- [Session_Start]/[Session_End] : A tout client est attachée une session sauf si l'application est configurée comme n'ayant pas de session. Un client n'est pas un utilisateur devant son écran. Si celui-ci a ouvert 2 navigateurs pour interroger l'application, il représente deux clients. Un client est caractérisé par un jeton de session qu'il doit joindre à chacune de ses demandes. Ce jeton de session est une suite de caractères générée aléatoirement par le serveur web et unique. Deux clients ne peuvent avoir le même jeton de session. Ce jeton va suivre le client de la façon suivante :
 - le client qui fait sa première demande n'envoie pas de jeton de session. Le serveur web reconnaît ce fait et lui en attribue un. C'est le début de la session et la procédure [Session_Start] est exécutée. Ce sera l'unique fois.
 - le client fait ses demandes suivantes en envoyant le jeton qui l'identifie. Cela va permettre au serveur web de retrouver des informations liées à ce jeton. Cela va permettre un suivi entre les différentes demandes du client.
 - l'application peut mettre à la disposition d'un client, un formulaire de fin de session. Dans ce cas, c'est le client qui demande lui-même la fin de sa session. La procédure [Session_End] sera exécutée. Ce sera l'unique fois.
 - le client peut ne jamais demander lui-même la fin de sa session. Dans ce cas, après un certain délai d'inactivité de la session, lui-aussi configurable par [web.config], la session sera terminée par le serveur web. La procédure [Session_End] sera alors exécutée.
- [Application_BeginRequest] : cette procédure est exécutée dès qu'une nouvelle demande arrive. Elle est donc exécutée à chaque requête d'un client quelconque. C'est un bon endroit pour examiner la requête avant de la transmettre à la page qui a été demandée. On peut même prendre la décision de la réorienter vers une autre page.
- [Application_Error] : est exécutée à chaque fois que se produit une erreur non gérée explicitement par le code du contrôleur [global.asax.vb]. On peut ici, réorienter la demande du client vers une page expliquant la cause de l'erreur.

Si aucun de ces événements ne doit être géré, alors le fichier [global.asax] peut être ignoré. C'est ce qui a été fait avec les premiers exemples de ce chapitre.

3.1.3.2 Exemple 1

Développons une application pour mieux appréhender les trois moments que sont : le démarrage de l'application, de la session, d'une demande client. Le fichier [global.asax] sera le suivant :

```
<%@ Application src="Global.asax.vb" Inherits="global" %>
```

Le fichier [global.asax.vb] associé sera le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque l'application est démarrée
        ' on note l'heure
        Dim startApplication As String = Date.Now.ToString("T")
        ' on la range dans le contexte de l'application
        Application.Item("startApplication") = startApplication
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque la session est démarrée
        ' on note l'heure
        Dim startSession As String = Date.Now.ToString("T")
        ' on la met dans la session
        Session.Item("startSession") = startSession
    End Sub
End Class
```

```

Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' on note l'heure
    Dim startRequest As String = Date.Now.ToString("T")
    ' on la met dans la session
    Context.Items("startRequest") = startRequest
End Sub
End Class

```

Les points importants du code sont les suivants :

- le serveur web rend disponible à la classe [HttpApplication] de [global.asax.vb] un certain nombre d'objets :
 - **Application** de type [HttpApplicationState] - représente l'application web - donne accès à un dictionnaire d'objets [Application.Item] accessible à tous les clients de l'application - permet le partage d'informations entre différents clients - l'accès simultané de plusieurs clients à une même donnée en lecture/écriture nécessite une synchronisation des clients.
 - **Session** de type [HttpSessionState] - représente un client particulier - donne accès à un dictionnaire d'objets [Session.Item] accessible à toutes les requêtes de ce client - va permettre de mémoriser des informations sur un client qu'on va pouvoir retrouver au fil des requêtes de celui-ci.
 - **Request** de type [HttpRequest] - représente la requête HTTP courante du client
 - **Response** de type [HttpResponse] - représente la réponse HTTP en cours de construction du serveur au client
 - **Server** de type [HttpServerUtility] - offre des méthodes utilitaires notamment pour transférer la requête à une autre page que celle prévue initialement.
 - **Context** de type [HttpContext] - cet objet est recréé à chaque nouvelle requête mais est partagé par toutes les pages qui participent au traitement de la requête - permet de transmettre des informations de page en page lors du traitement d'une requête grâce à son dictionnaire **Items**.
- la procédure [Application_Start] note le début de l'application dans une variable stockée dans un dictionnaire accessible au niveau application
- la procédure [Session_Start] note le début de la session dans une variable stockée dans un dictionnaire accessible au niveau session
- la procédure [Application_BeginRequest] note le début de la requête dans une variable stockée dans un dictionnaire accessible au niveau requête (c.a.d disponible pendant tout le temps de son traitement mais perdue à la fin de celui-ci)

La page cible sera la page [main.aspx] suivante :

```

<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<html>
<head>
<title>global.asax</title>
</head>
<body>
jeton de session :
<% =jeton %><br/>
début Application :
<% =startApplication %><br/>
début Session :
<% =startSession %><br/>
début Requête :
<% =startRequest %><br/>
</body>
</html>

```

Cette page de présentation affiche des valeurs calculées par son contrôleur [main.aspx.vb] :

```

Public Class main
    Inherits System.Web.UI.Page

    Protected startApplication As String
    Protected startSession As String
    Protected startRequest As String
    Protected jeton as String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère les infos de l'application et de la session
        jeton=Session.SessionId
        startApplication = Application.Item("startApplication").ToString
        startSession = Session.Item("startSession").ToString
        startRequest = Context.Items("startRequest").ToString
    End Sub
End Class

```

Le contrôleur se contente de récupérer les trois informations placées respectivement dans l'application, la session, le contexte par [global.asax.vb].

Les fondamentaux du développement asp.net

Nous testons l'application de la façon suivante :

1. les fichiers sont rassemblés dans un même dossier <application-path>



2. le serveur Cassini est lancé avec les paramètres (<application-path>, /aspnet/globalasax1)
3. un premier client demande l'url [http://localhost/aspnet/globalasax1/main.aspx] et obtient le résultat suivant :



```
jeton de session : slxp0bqmsyaystjqox250r45
début Application : 08:01:29
début Session : 08:01:30
début Requête : 08:01:30
```

4. le même client fait une nouvelle requête (option Reload du navigateur) :



```
jeton de session : slxp0bqmsyaystjqox250r45
début Application : 08:01:29
début Session : 08:01:30
début Requête : 08:02:21
```

On peut constater que seule l'heure de la requête a changé. Ceci montre deux choses :

- les procédures [Application_Start] et [Session_Start] de [global.asax] n'ont pas été exécutées lors de la seconde requête.
- les objets [Application] et [Session] où étaient stockées les heures de début de l'application et de la session sont encore disponibles pour la seconde requête.

5. on lance un second navigateur pour créer un second client et nous redemandons la même url :



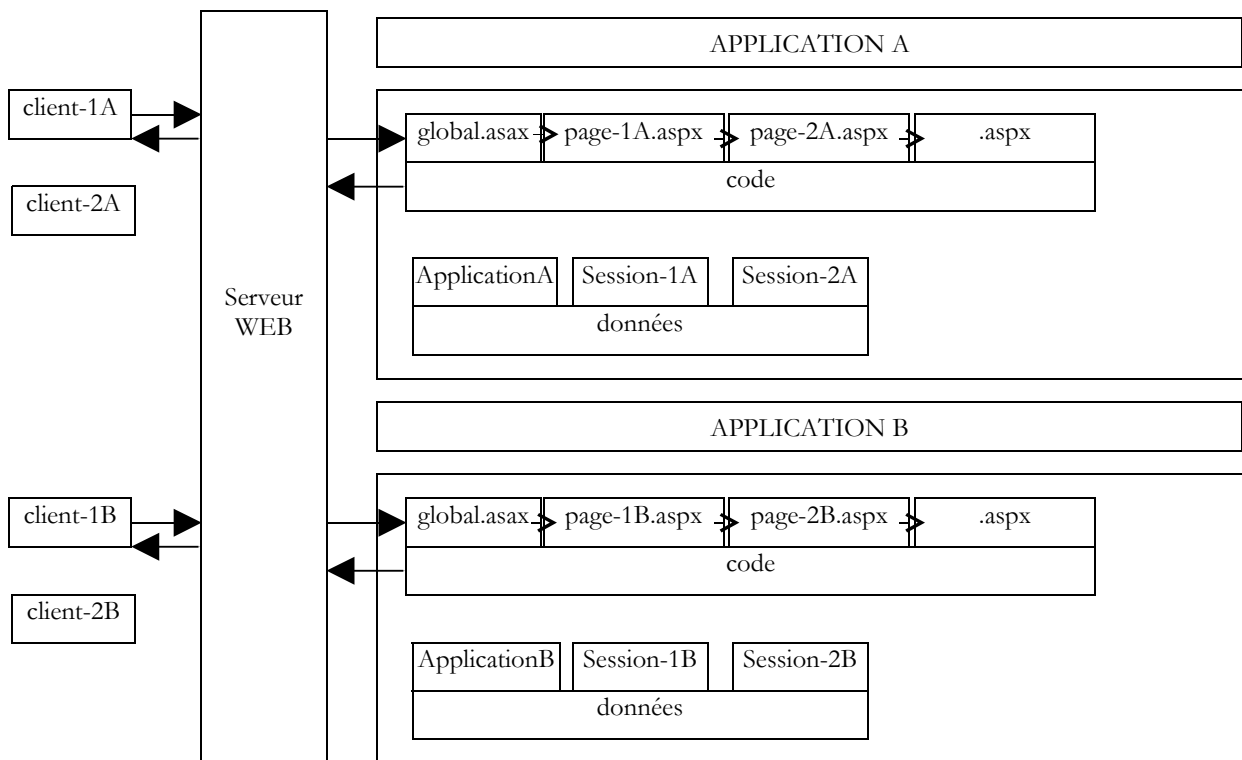
```
jeton de session : chgp1redg4cn0m55diovl45
début Application : 08:01:29
début Session : 08:02:51
début Requête : 08:02:51
```

Cette fois-ci, nous voyons que l'heure de la session a changé. Le deuxième navigateur, bien que sur la même machine, a été considéré comme un second client et une nouvelle session a été créée pour lui. On peut constater que les deux clients n'ont pas le même jeton de session. L'heure de début de l'application n'a pas changé, ce qui signifie que :

- la procédure [Application_Start] de [global.asax.vb] n'a pas été exécutée
- l'objet [Application] où a été stockée l'heure de début de l'application est accessible au second client. C'est donc dans cet objet qu'il faut stocker les informations que doivent se partager les différents clients de l'application, l'objet [Session] lui, servant à stocker des informations que doivent se partager les requêtes d'un **même client**.

3.1.3.3 Une vue d'ensemble

Avec ce que nous avons appris jusqu'à maintenant, nous sommes en mesure de faire un premier schéma explicatif du fonctionnement d'un serveur web et des application web qu'il sert :



Le schéma précédent nous montre un serveur servant deux applications notées A et B, chacune avec deux clients. Un serveur web est capable de servir plusieurs applications web simultanément. Celles-ci sont totalement indépendantes les unes des autres. Nous raisonnerons sur l'application A. Le traitement d'une requête du client-1A à l'application A va se dérouler de la façon suivante :

- le client 1A demande au serveur web une ressource qui appartient au domaine de l'application A. Cela veut dire qu'il demande une URL de la forme [http://machine:port/VA/ressource] où VA est le chemin virtuel de l'application A.
- si le serveur web détecte que c'est la 1ère demande d'une ressource de l'application A, il déclenche l'événement [Application_Start] du fichier [global.asax] de l'application A. Un objet [ApplicationA] de type [HttpApplicationState] va être construit. Les différents codes de l'application stockeront dans cet objet des données de portée [Application], c.a.d. des données concernant tous les utilisateurs. L'objet [ApplicationA] va exister jusqu'à ce que le serveur web décharge l'application A.
- si le serveur web détecte de plus qu'il a affaire à un nouveau client de l'application A, il va déclencher l'événement [Session_Start] du fichier [global.asax] de l'application A. Un objet [Session-1A] de type [HttpSessionState] va être construit. Cet objet va permettre à l'application A de stocker des objets de portée [Session], c.a.d. des objets appartenant à un client précis. L'objet [Session-1A] va exister tant que le client 1A fera des requêtes. Il va permettre un suivi de ce client. Le serveur web détecte qu'il a affaire à un nouveau client dans deux cas :
 - le client ne lui a pas envoyé de jeton de session dans les entêtes HTTP de sa requête
 - le client lui a envoyé un jeton de session qui n'existe pas (mauvais fonctionnement du client ou tentative de piratage) ou qui n'existe plus. Un jeton de session expire en effet au bout d'un certain délai d'inactivité du client (20 mn par défaut avec IIS). Ce délai est programmable.
- dans tous les cas, le serveur web va déclencher l'événement [Application_BeginRequest] du fichier [global.asax]. Cet événement démarre le traitement d'une requête client. Il est fréquent de ne pas traiter cet événement et de passer la main à la page demandée par le client qui elle traitera la requête. On peut aussi utiliser cet événement pour analyser la requête, la traiter et décider de la page qui doit être envoyée en réponse. Nous utiliserons cette technique pour mettre en place une application respectant l'architecture MVC dont nous avons parlé.
- une fois le filtre de [global.asax] passé, la requête du client est passée à une page .aspx qui va traiter la requête. Nous verrons ultérieurement qu'il est possible de passer la requête au travers d'un filtre de plusieurs pages. La dernière sera chargée d'envoyer la réponse au client. Les pages peuvent ajouter à la requête initiale du client, des informations qu'elles ont calculées. Elles peuvent stocker ces informations dans la collection **Context.Items**. En effet, toutes les pages engagées dans le traitement de la requête d'un client ont accès à ce réservoir de données.
- le code des différentes pages a accès aux réservoirs de données que sont les objets [ApplicationA], [Session-1A], ... Il faut se souvenir que le serveur web traite simultanément plusieurs clients pour l'application A. Tous ces clients ont accès à l'objet [Application A]. S'ils doivent modifier des données dans cet objet, il y a un travail de synchronisation des clients à

faire. Chaque client XA a de plus accès au réservoir de données [Session-XA]. Celui-ci lui étant réservé, il n'y a pas là de synchronisation à faire.

- le serveur web sert plusieurs applications web simultanément. Il n'y a aucune interférence entre les clients de ces différentes applications.

De ces explications, on retiendra les points suivants :

- à un moment donné, un serveur web sert de multiples clients de façon simultanée. Cela signifie qu'il n'attend pas la fin d'une requête pour en traiter une autre. A un temps T, il y a donc plusieurs requêtes en cours de traitement appartenant à des clients différents pour des applications différentes. On appelle parfois threads d'exécution, les codes de traitement qui se déroulent en même temps au sein du serveur web.
- les threads d'exécution des clients d'**applications web différentes** n'interfèrent pas entre-eux. Il y a étanchéité.
- les threads d'exécution des clients d'**une même application** peuvent avoir à partager des données :
 - les threads d'exécution des requêtes **de deux clients différents** (pas le même jeton de session) peuvent partager des données au moyen de l'objet [Application].
 - les threads d'exécution des requêtes successives **d'un même client** peuvent partager des données au moyen de l'objet [Session].
 - les threads d'exécution des pages successives traitant une **même requête** d'un client donné peuvent partager des données au moyen de l'objet [Context].

3.1.3.4 Exemple 2

Développons un nouvel exemple mettant en lumière ce qui vient d'être vu. Nous rassemblons dans le même dossier les fichiers suivants :

[global.asax]

```
<%@ Application src="Global.asax.vb" Inherits="global" %>
```

[global.asax.vb]

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque l'application est démarrée
        ' init compteur de clients
        Application.Item("nbRequêtes") = 0
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque la session est démarrée
        ' init compteur de requêtes
        Session.Item("nbRequêtes") = 0
    End Sub
End Class
```

Le principe de l'application va être de compter le nombre total de requêtes faites à l'application et le nombre de requêtes par client. Lorsque l'application démarre [Application_Start], on met à 0 le compteur des requêtes faites à l'application. Ce compteur est placé dans la portée [Application] car il doit être incrémenté par tous les clients. Lorsqu'un client se présente pour la première fois [Session_Start], on met à 0 le compteur des requêtes faites par ce client. Ce compteur est placé dans la portée [Session] car il ne concerne qu'un client donné.

Une fois [global.asax] exécuté, le fichier [main.aspx] suivant sera exécuté :

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<html>
<head>
<title>application-session</title>
</head>
<body>
jeton de session :
<% =jeton %>
<br />
requêtes Application :
<% =nbRequêtesApplication %>
<br />
requêtes Client :
<% =nbRequêtesClient %>
```

```
<br />
</body>
</html>
```

Il affiche trois informations calculées par son contrôleur :

1. l'identité du client via son jeton de session : [jeton]
2. le nombre total de requêtes faites à l'application : [nbRequêtesApplication]
3. le nombre total de requêtes faites par le client identifié en 1 : [nbRequêtesClient]

Les trois informations sont calculées dans [main.aspx.vb] :

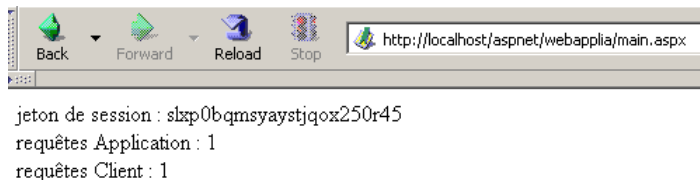
```
Public Class main
    Inherits System.Web.UI.Page

    Protected nbRequêtesApplication As String
    Protected nbRequêtesClient As String
    Protected jeton As String

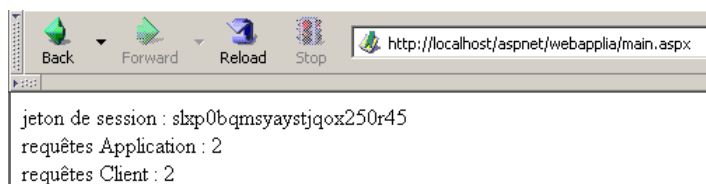
    Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' une requête de plus pour l'application
        Application.Item("nbRequêtes") = CType(Application.Item("nbRequêtes"), Integer) + 1
        ' une requête de plus dans la session
        Session.Item("nbRequêtes") = CType(Session.Item("nbRequêtes"), Integer) + 1
        ' init variables de présentation
        nbRequêtesApplication = Application.Item("nbRequêtes").ToString
        jeton = Session.SessionID
        nbRequêtesClient = Session.Item("nbRequêtes").ToString
    End Sub
End Class
```

Lorsque [main.aspx.vb] est exécutée, nous sommes en cours de traitement d'une requête d'un client donné. Nous utilisons l'objet [Application] pour incrémenter le nombre de requêtes de l'application et l'objet [Session] pour incrémenter le nombre de requêtes du client dont on est en train de traiter la requête. Rappelons que si tous les clients d'une même application partagent le même objet [Application], ils ont chacun un objet [Session] qui leur est propre.

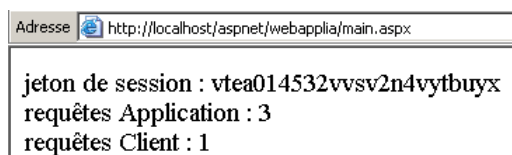
Nous testons l'application en plaçant les quatre fichiers précédents dans un dossier que nous appelons <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/aspnet/webappia). Nous lançons un premier navigateur et demandons l'url [http://localhost/aspnet/webappia/main.aspx] :



Nous faisons une seconde requête avec le bouton [Reload] :



Nous lançons un second navigateur pour demander la même url. Pour le serveur web, c'est un nouveau client :



On peut constater que le jeton de session a changé et qu'on a donc un nouveau client. Cela est reflété dans le nombre de requêtes du client. Revenons maintenant au 1er navigateur et redemandons encore la même url :



jeton de session : slxp0bqmsyastjiox250r45
requêtes Application : 4
requêtes Client : 3

Le nombre de requêtes faites à l'application sont bien toutes comptées.

3.1.3.5 De la nécessité de synchroniser les clients d'une application

Dans l'application précédente, le compteur de requêtes faites à l'application est incrémenté dans la procédure [Form_Load] de la page [main.aspx] de la façon suivante :

```
' une requête de plus pour l'application  
Application.Item("nbRequêtes") = CType(Application.Item("nbRequêtes"), Integer) + 1
```

Cette instruction, quoique simple, nécessite plusieurs instructions du processeur pour être exécutée. Supposons qu'il en faille trois :

1. lecture du compteur
2. incrémentation du compteur
3. réécriture du compteur

Le serveur web s'exécute sur une machine multi-tâches, ce qui entraîne que chaque tâche se voit accordé le processeur pendant quelques millisecondes avant de le perdre puis de le retrouver après que toutes les autres tâches aient eu elles aussi leur quantum de temps. Supposons que deux clients A et B fassent une requête en même temps au serveur web. Admettons que le client A passe le premier, qu'il arrive dans la procédure [Form_Load] de [main.aspx.vb], lise le compteur (=100) puis est interrompu parce que son quantum de temps est épuisé. Supposons maintenant que ce soit le tour du client B et que celui-ci subisse le même sort : il arrive à lire la valeur du compteur (=100) mais n'a pas le temps de l'incrémenter. Les clients A et B sont tous deux en possession d'un compteur égal à 100. Supposons que revient le tour du client A : il incrémente son compteur, le passe à 101 puis se termine. C'est le tour du client B qui a en sa possession l'ancienne valeur du compteur et non la nouvelle. Il passe donc lui aussi la valeur du compteur à 101 et se termine. La valeur du compteur de requêtes de l'application est maintenant erroné.

Pour illustrer ce problème, nous reprenons l'application précédente que nous modifions de la façon suivante :

- les fichiers [global.asax], [global.asax.vb] et [main.aspx] ne changent pas
- le fichier [main.aspx.vb] devient le suivant :

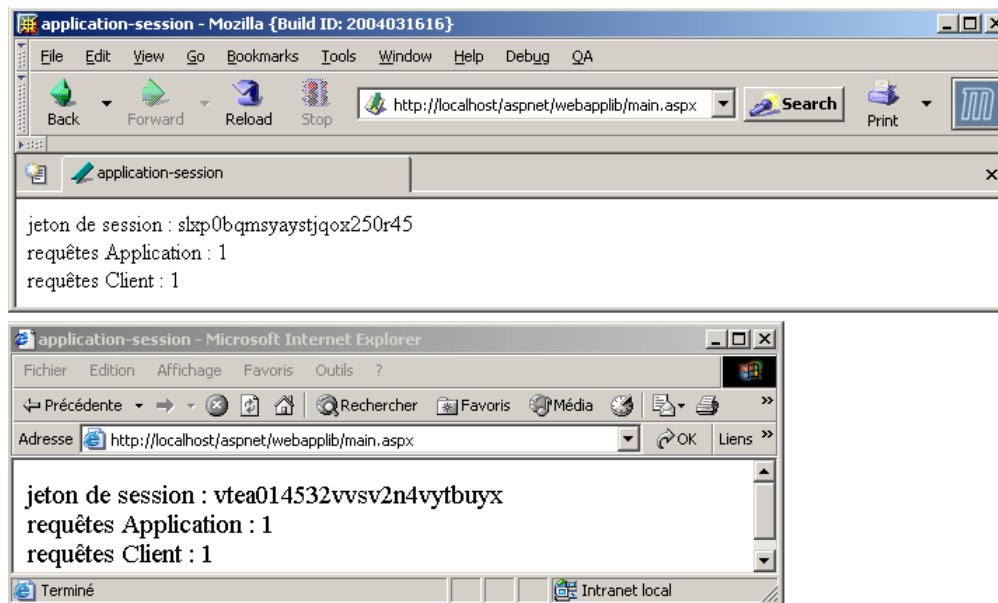
```
Imports System.Threading  
  
Public Class main  
    Inherits System.Web.UI.Page  
  
    Protected nbRequêtesApplication As Integer  
    Protected nbRequêtesClient As Integer  
    Protected jeton As String  
  
    Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load  
        ' une requête de plus pour l'application et la session  
        ' lecture compteurs  
        nbRequêtesApplication = CType(Application.Item("nbRequêtes"), Integer)  
        nbRequêtesClient = CType(Session.Item("nbRequêtes"), Integer)  
        ' attente 5 s  
        Thread.Sleep(5000)  
        ' incrémentation des compteurs  
        nbRequêtesApplication += 1  
        nbRequêtesClient += 1  
        ' enregistrement des compteurs  
        Application.Item("nbRequêtes") = nbRequêtesApplication  
        Session.Item("nbRequêtes") = nbRequêtesClient  
        ' init variables de présentation  
        jeton = Session.SessionID  
    End Sub  
End Class
```

L'incrémentation des compteurs a été divisée en quatre phases :

1. lecture du compteur
2. mise en sommeil du thread d'exécution
3. incrémentation du compteur
4. réécriture du compteur

Considérons de nouveau nos deux clients A et B. Entre la phase de lecture et celle d'incrémentement des compteurs de requêtes, nous forçons le thread d'exécution à s'arrêter pendant 5 secondes. Cela va avoir pour conséquence immédiate qu'il va perdre le processeur qui sera alors donné à une autre tâche. Supposons que le client A passe le premier. Il va lire la valeur N du compteur et être interrompu pendant 5 secondes. Si pendant celles-ci, le client B dispose du processeur, il devrait lire la même valeur N du compteur. Au final, les deux clients devraient afficher la même valeur du compteur, ce qui serait anormal.

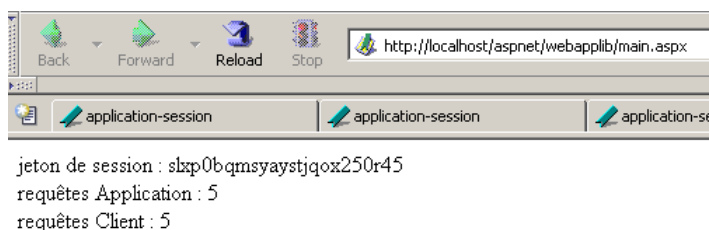
Nous testons l'application en plaçant les quatre fichiers précédents dans un dossier que nous appelons <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/aspnet/webapplib). Nous préparons deux navigateurs différents avec l'url [http://localhost/aspnet/webapplib/main.aspx]. Nous lançons le premier pour qu'il demande l'URL, puis sans attendre la réponse qui arrivera 5 secondes plus tard, on lance le second navigateur. Au bout d'un peu plus de 5 secondes, on obtient le résultat suivant :



On voit:

- qu'on a deux clients différents (pas le même jeton de session)
- que chaque client a fait une requête
- que le compteur de requêtes faites à l'application devrait donc être à 2 dans l'un des deux navigateurs. Ce n'est pas le cas.

Maintenant, faisons une autre expérience. Avec le même navigateur, nous lançons cinq requêtes à l'url [http://localhost/aspnet/webapplib/main.aspx]. Là encore, nous les lançons les unes après les autres sans attendre les résultats. Lorsque toutes les requêtes ont été exécutées, on obtient le résultat suivant pour la dernière :



On peut remarquer :

- que les 5 requêtes ont été considérées comme provenant du même client car le compteur de requêtes client est à 5. Non montré ci-dessus, on constate que le jeton de session est effectivement le même pour les 5 requêtes.
- que le compteur de requêtes faites à l'application est correct.

Qu'en conclure ? Rien de définitif. Peut-être le serveur web ne commence-t-il pas à exécuter une requête d'un client si celui-ci en a déjà une en cours d'exécution ? Il n'y aurait donc jamais simultanéité d'exécution des requêtes d'un même client. Elles seraient exécutées les unes après les autres. Ce point est à vérifier. Il peut en effet dépendre du type de client utilisé.

3.1.3.6 Synchronisation des clients

Le problème mis en évidence dans l'application précédente est un problème classique (mais pas simple à résoudre) d'accès exclusif à une ressource. Dans notre problème particulier, il faut faire en sorte que deux clients A et B ne puissent être en même temps dans la séquence de code :

1. lecture du compteur
2. incrémentation du compteur
3. réécriture du compteur

On appelle une telle séquence de code, une séquence critique. Elle nécessite une synchronisation des threads amenés à l'exécuter de façon simultanée. La palte-forme .NET offre divers outils pour assurer celle-ci. Nous allons ici utiliser la classe [Mutex].

Bibliothèque de classes .NET Framework
Mutex, classe [Visual Basic]

Primitive de synchronisation qui peut également être utilisée pour la synchronisation entre processus.

Pour obtenir une liste de tous les membres de ce type, consultez [Mutex, membres](#).

[System.Object](#)
[System.MarshalByRefObject](#)
[System.Threading.WaitHandle](#)
System.Threading.Mutex

```
NotInheritable Public Class Mutex
    Inherits WaitHandle
```

Nous n'utiliserons ici que les constructeurs et méthodes suivants :

<code>public Mutex()</code>	crée un objet de synchronisation M
<code>public bool WaitOne()</code>	Le thread T1 qui exécute l'opération <i>M.WaitOne()</i> demande la propriété de l'objet de synchronisation M. Si le Mutex M n'est détenu par aucun thread (le cas au départ), il est "donné" au thread T1 qui l'a demandé. Si un peu plus tard, un thread T2 fait la même opération, il sera bloqué. En effet, un Mutex ne peut appartenir qu'à un thread. Il sera débloqué lorsque le thread T1 libérera le mutex M qu'il détient. Plusieurs threads peuvent ainsi être bloqués en attente du Mutex M.
<code>public void ReleaseMutex()</code>	Le thread T1 qui effectue l'opération <i>M.ReleaseMutex()</i> abandonne la propriété du Mutex M. Lorsque le thread T1 perdra le processeur, le système pourra le donner à l'un des threads en attente du Mutex M. Un seul l'obtiendra à son tour, les autres en attente de M restant bloqués

Un Mutex M gère l'accès à une ressource partagée R. Un thread demande la ressource R par *M.WaitOne()* et la rend par *M.ReleaseMutex()*. Une section critique de code qui ne doit être exécutée que par un seul thread à la fois est une ressource partagée. La synchronisation d'exécution de la section critique peut se faire ainsi :

```
M.WaitOne()
' le thread est seul à entrer ici
' section critique
....
M.ReleaseMutex()
```

où M est un objet *Mutex*. Il faut bien sûr ne jamais oublier de libérer un *Mutex* devenu inutile, afin qu'un autre thread puisse entrer dans la section critique à son tour, sinon les threads en attente d'un Mutex jamais libéré n'auront jamais accès au processeur. Par ailleurs, il faut éviter la situation d'interblocage (*deadlock*) dans laquelle deux threads s'attendent mutuellement. Considérons les actions suivantes qui se suivent dans le temps :

- un thread T1 obtient la propriété d'un Mutex M1 pour avoir accès à une ressource partagée R1
- un thread T2 obtient la propriété d'un Mutex M2 pour avoir accès à une ressource partagée R2
- le thread T1 demande le Mutex M2. Il est bloqué.
- le thread T2 demande le Mutex M1. Il est bloqué.

Ici, les threads T1 et T2 s'attendent mutuellement. Ce cas apparaît lorsque des threads ont besoin de deux ressources partagées, la ressource R1 contrôlée par le Mutex M1 et la ressource R2 contrôlée par le Mutex M2. Une solution possible est de demander les deux ressources en même temps à l'aide d'un Mutex unique M. Mais ce n'est pas toujours possible, notamment si cela entraîne une mobilisation longue d'une ressource coûteuse. Une autre solution est qu'un thread ayant M1 et ne pouvant obtenir M2, relâche alors M1 pour éviter l'interblocage.

Si nous mettons en pratique ce que nous venons d'apprendre, notre application devient la suivante :

- les fichiers [global.asax] et [main.aspx] ne changent pas
- le fichier [global.asax.vb] devient le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports System.Threading
```

```

Public Class global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque l'application est démarrée
        ' init compteur de clients
        Application.Item("nbRequêtes") = 0
        ' création d'un verrou de synchronisation
        Application.Item("verrou") = New Mutex
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Se déclenche lorsque la session est démarrée
        ' init compteur de requêtes
        Session.Item("nbRequêtes") = 0
    End Sub
End Class

```

La seule nouveauté est la création d'un [Mutex] qui sera utilisé par les clients pour se synchroniser. Parce qu'il doit être accessible à tous les clients, il est placé dans l'objet [Application].

- le fichier [main.aspx.vb] devient le suivant :

```

Imports System.Threading

Public Class main
    Inherits System.Web.UI.Page

    Protected nbRequêtesApplication As Integer
    Protected nbRequêtesClient As Integer
    Protected jeton As String

    Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' une requête de plus pour l'application et la session
        ' on entre dans une section critique - on récupère le verrou de synchronisation
        Dim verrou As Mutex = CType(Application.Item("verrou"), Mutex)
        ' on demande à entrer seul dans la section critique qui suit
        verrou.WaitOne()

        ' lecture compteurs
        nbRequêtesApplication = CType(Application.Item("nbRequêtes"), Integer)
        nbRequêtesClient = CType(Session.Item("nbRequêtes"), Integer)
        ' attente 5 s
        Thread.Sleep(5000)
        ' incrémentation des compteurs
        nbRequêtesApplication += 1
        nbRequêtesClient += 1
        ' enregistrement des compteurs
        Application.Item("nbRequêtes") = nbRequêtesApplication
        Session.Item("nbRequêtes") = nbRequêtesClient

        ' on permet l'accès à la section critique
        verrou.ReleaseMutex()

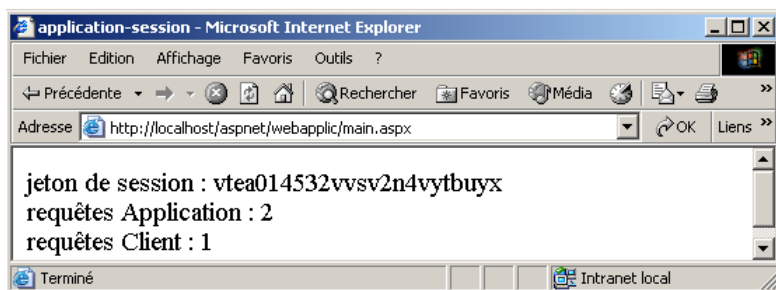
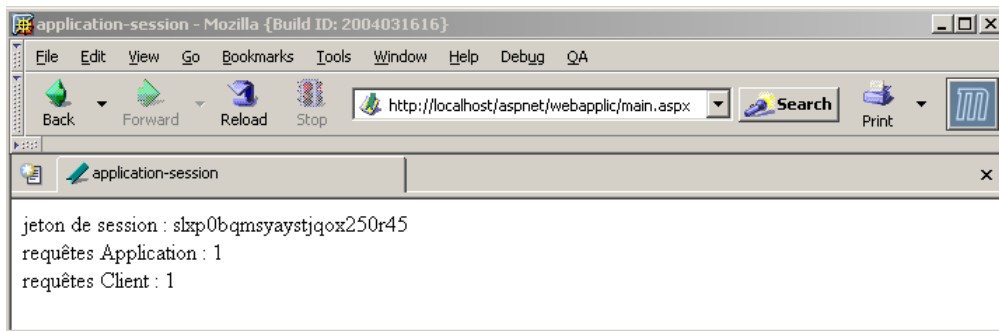
        ' init variables de présentation
        jeton = Session.SessionID
    End Sub
End Class

```

On voit que le client :

- demande à entrer seul dans la section critique. Il demande pour cela la propriété exclusive du Mutex [verrou]
- il libère le Mutex [verrou] à la fin de la section critique afin qu'un autre client puisse entrer à son tour dans la section critique.

Nous testons l'application en plaçant les quatre fichiers précédents dans un dossier que nous appelons <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/aspnet/webapplic). Nous préparons deux navigateurs différents avec l'url [http://localhost/aspnet/webapplic/main.aspx]. Nous lançons le premier pour qu'il demande l'URL puis, sans attendre la réponse qui arrivera 5 secondes plus tard, on lance le second navigateur. Au bout d'un peu plus de 5 secondes, on obtient le résultat suivant :



Cette fois-ci, le compteur de requêtes de l'application est correct.

On retiendra de cette longue démonstration, l'absolue nécessité de synchroniser les clients d'une même application web, s'ils doivent mettre à jour des éléments partagés par tous les clients.

3.1.3.7 Gestion du jeton de session

Nous avons parlé de nombreuses fois du jeton de session que s'échangeait le client et le serveur web. Rappelons son principe :

- le client fait une première requête au serveur. Il n'envoie pas de jeton de session.
- à cause de l'absence du jeton de session dans la requête, le serveur reconnaît un nouveau client et lui affecte un jeton. A ce jeton, est également associé un objet [Session] qui sera utilisé pour stocker des informations propres à ce client. Le jeton va suivre toutes les requêtes de ce client. Il sera inclus dans les entêtes HTTP de la réponse faite à la première requête du client.
- le client connaît maintenant son jeton de session. Il va le renvoyer dans les entêtes HTTP de chacune des requêtes suivantes qu'il va faire au serveur web. Grâce au jeton, le serveur pourra retrouver l'objet [Session] attaché au client.

Pour mettre en évidence ce mécanisme, nous reprenons l'application précédente en ne modifiant que le seul fichier [main.aspx.vb] :

```
Imports System.Threading

Public Class main
  Inherits System.Web.UI.Page

  Protected nbRequêtesApplication As Integer
  Protected nbRequêtesClient As Integer
  Protected jeton As String

  Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' une requête de plus pour l'application et la session
    ' on entre dans une section critique - on récupère le verrou de synchronisation
    Dim verrou As Mutex = CType(Application.Item("verrou"), Mutex)
    ' on demande à entrer seul dans la section qui suit
    verrou.WaitOne()
    ' lecture compteurs
    nbRequêtesApplication = CType(Application.Item("nbRequêtes"), Integer)
    nbRequêtesClient = CType(Session.Item("nbRequêtes"), Integer)
    ' attente 5 s
    Thread.Sleep(5000)
    ' incrémentation des compteurs
    nbRequêtesApplication += 1
    nbRequêtesClient += 1
    ' enregistrement des compteurs
    Application.Item("nbRequêtes") = nbRequêtesApplication
    Session.Item("nbRequêtes") = nbRequêtesClient
    ' on permet l'accès à la section critique
    verrou.ReleaseMutex()
    ' init variables de présentation
    jeton = Session.SessionID
  End Sub
```

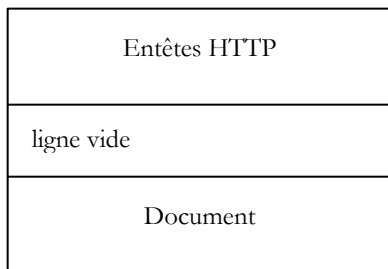
```

Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Init
    ' on mémorise la requête du client dans request.txt du dossier de l'application
    Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
    Me.Request.SaveAs(requestFileName, True)
End Sub
End Class

```

Lorsque se produit l'événement [Page_Init], nous sauvegardons la requête du client dans le dossier de l'application. Rappelons quelques points :

- [TemplateSourceDirectory] représente le chemin virtuel de la page en cours d'exécution,
- MapPath(TemplateSourceDirectory) représente le chemin physique correspondant. Ceci nous permet de construire le chemin physique du fichier à construire,
- [Request] est un objet représentant la requête en cours de traitement. Cet objet a été construit en exploitant la requête brute envoyée par le client, c.a.d. une suite de lignes de texte de la forme :



- Request.Save([FileName]) sauvegarde la totalité de la requête du client (entêtes HTTP et éventuellement le document qui suit) dans un fichier dont le chemin est passé en paramètre.

Nous pouvons donc savoir exactement quelle a été la requête du client. Nous testons l'application en plaçant les quatre fichiers précédents dans un dossier que nous appelons <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/aspnet/session1). Puis avec un navigateur, nous demandons l'URL [http://localhost/aspnet/session1/main.aspx]. Nous obtenons le résultat suivant :



```

jeton de session : y153tk45sise0lrhdzrf22m3
requêtes Application : 1
requêtes Client : 1

```

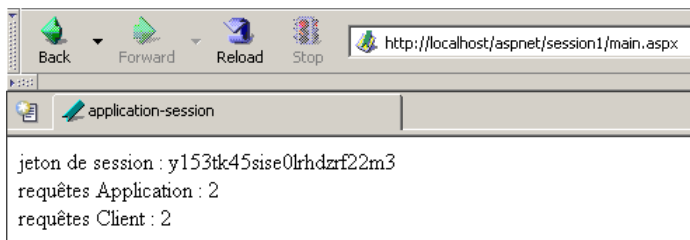
Nous utilisons le fichier [request.txt] sauvegardé par [main.aspx.vb] pour avoir accès à la requête du navigateur :

```

GET /aspnet/session1/main.aspx HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Keep-Alive: 300
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7b) Gecko/20040316

```

Nous constatons que le navigateur a fait la demande de l'URL [/aspnet/session1/main.aspx], envoyé d'autres informations dont nous avons déjà parlé dans le précédent chapitre. On n'y voit pas de jeton de session. La page reçue en réponse, montre elle que le serveur a créé un jeton de session. On ne sait pas encore si le navigateur l'a reçu. Faisons maintenant une seconde requête avec le même navigateur (Reload). Nous obtenons la nouvelle réponse suivante :

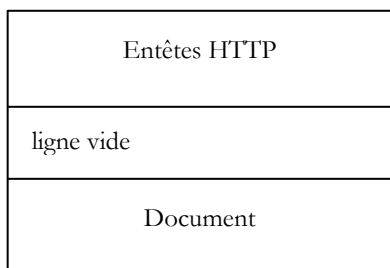


Il y a bien un suivi de session puisque le nombre de requêtes de la session a été correctement incrémenté. Voyons maintenant le contenu du fichier [request.txt] :

```
GET /aspnet/session1/main.aspx HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Keep-Alive: 300
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Cookie: ASP.NET_SessionId=y153tk45sise0lrhdzrf22m3
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7b) Gecko/20040316
```

On constate que, pour cette deuxième requête, le navigateur a envoyé au serveur un nouvel entête HTTP [Cookie:] définissant une information appelée [ASP.NET_SessionId] et ayant pour valeur le jeton de session qu'on a vu apparaître dans la réponse à la première requête. Grâce à ce jeton, le serveur web va connecter cette nouvelle requête à l'objet [Session] identifié par le jeton [y153tk45sise0lrhdzrf22m3] et retrouver le compteur de requêtes associé.

On ne sait toujours pas par quel mécanisme, le serveur a envoyé le jeton au client car nous n'avons pas accès à la réponse HTTP du serveur. Rappelons que celle-ci a la même structure que la demande du client, à savoir un ensemble de lignes de texte de la forme :



Nous avons eu l'occasion d'utiliser un client web qui nous donnait accès à la réponse HTTP du serveur web, le client curl. Nous l'utilisons de nouveau, dans une fenêtre dos, pour interroger la même url que le navigateur précédent :

```
E:\curl>curl --include http://localhost/aspnet/session1/main.aspx
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 07:31:42 GMT
X-AspNet-Version: 1.1.4322
Set-Cookie: ASP.NET_SessionId=qxnxmqqmvhde3al55kzsmx445; path=/
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 228
Connection: Close

<HTML>
  <HEAD>
    <title>application-session</title>
  </HEAD>
  <body>
    jeton de session :
    qxnxmqqmvhde3al55kzsmx445
    <br>
    requêtes Application :
    3
    <br>
    requêtes Client :
    1
```

```
<br>
</body>
</HTML>
```

Nous avons la réponse à notre question. Le serveur web envoie le jeton de session sous la forme d'un entête HTTP [Set-Cookie:] :

```
Set-Cookie: ASP.NET_SessionId=qxnxmqqmvhde3al55kzsmx445; path=/
```

Faisons la même demande sans renvoyer le jeton de session. On obtient la réponse suivante :

```
E:\curl>curl --include http://localhost/aspnet/session1/main.aspx
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 07:36:06 GMT
X-AspNet-Version: 1.1.4322
```

```
Set-Cookie: ASP.NET_SessionId=cs2p12mehdiz5v55ihevlkaz; path=/
```

```
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 228
Connection: Close
```

```
<HTML>
  <HEAD>
    <title>application-session</title>
  </HEAD>
  <body>
```

```
    jeton de session :
    cs2p12mehdiz5v55ihevlkaz
```

```
    <br>
    requêtes Application :
    4
    <br>
    requêtes Client :
    1
    <br>
```

```
  </body>
</HTML>
```

Parce que nous n'avons pas renvoyé le jeton de session, le serveur n'a pas pu nous identifier et nous a redonné un nouveau jeton. Pour poursuivre une session commencée, le client doit renvoyer au serveur le jeton de session qu'il a reçu. Nous allons le faire ici en utilisant l'option [--cookie clé=valeur] de curl qui va générer l'entête HTTP [Cookie: clé=valeur]. Nous avons vu que le navigateur avait envoyé cet entête HTTP lors de sa seconde requête.

```
E:\curl>curl --include --cookie ASP.NET_SessionId=cs2p12mehdiz5v55ihevlkaz http://localhost/aspnet/session1/main.aspx
```

```
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 07:40:20 GMT
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 228
Connection: Close
```

```
<HTML>
  <HEAD>
    <title>application-session</title>
  </HEAD>
  <body>
```

```
    jeton de session :
    cs2p12mehdiz5v55ihevlkaz
```

```
    <br>
    requêtes Application :
    5
    <br>
```

```
    requêtes Client :
    2
```

```
    <br>
```

```
  </body>
</HTML>
```

On remarquera plusieurs choses :

- le compteur de requêtes client a bien été incrémenté, montrant par là que le serveur a bien reconnu notre jeton.
- le jeton de session affiché par la page est bien celui qu'on a envoyé

- le jeton de session n'est plus dans les entêtes HTTP envoyés par le serveur web. En effet, celui-ci ne l'envoie qu'une fois : lors de la génération du jeton au démarrage d'une nouvelle session. Une fois que le client a obtenu son jeton, c'est à lui de l'utiliser quand il le veut pour se faire reconnaître.

Rien n'empêche un client de jouer avec plusieurs jetons de session, comme le montre l'exemple suivant avec [curl] où nous utilisons le jeton obtenu lors de notre première requête (requête n° 1) :

```
E:\curl>curl --include --cookie ASP.NET_SessionId=qxnxmvmhde3al55kzsmx445 http://localhost/aspnet/session1/main.aspx
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 07:48:47 GMT
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 228
Connection: Close

<HTML>
  <HEAD>
    <title>application-session</title>
  </HEAD>
  <body>
    jeton de session :
    qxnxmvmhde3al55kzsmx445
  <br>
  requêtes Application :
  6
  <br>
  requêtes Client :
  2
  <br>
  </body>
</HTML>
```

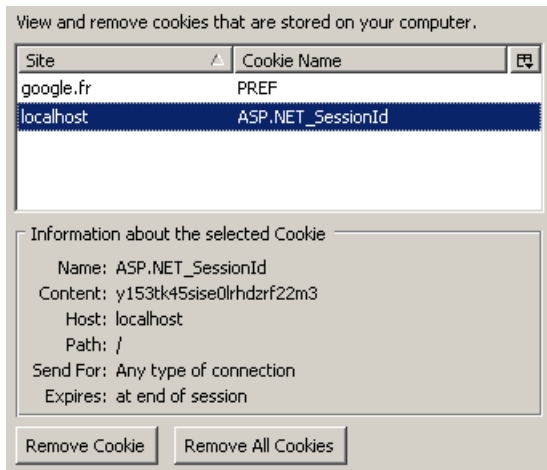
Que signifie cet exemple ? Nous avons envoyé un jeton obtenu un peu plus tôt. Lorsque le serveur web crée un jeton, il le garde tant que le client associé à ce jeton continue à lui envoyer des requêtes. Après un certain temps d'inactivité (20 mn par défaut avec IIS), le jeton est supprimé. L'exemple précédent montre que nous avons utilisé un jeton encore actif.

On peut avoir la curiosité de voir quelles ont été les requêtes HTTP du client [curl] pendant toutes ces manipulations. Nous savons qu'elles ont été enregistrées dans le fichier [request.txt]. Voici la dernière :

```
GET /aspnet/session1/main.aspx HTTP/1.1
Pragma: no-cache
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, */*
Cookie: ASP.NET_SessionId=qxnxmvmhde3al55kzsmx445
Host: localhost
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4
```

On y trouve bien l'entête HTTP envoyant le jeton de session.

Les informations transmises par le serveur via l'entête HTTP [Set-Cookie:] sont appelées des **cookies**. Le serveur peut utiliser ce mécanisme pour transmettre d'autres informations que le jeton de session. Lorsque le serveur S transmet un cookie à un client, il indique également la durée de vie D de celui-ci et l'URL U associée. Cela signifie pour le client que lorsqu'il demande au serveur S une url de la forme /U/chemin, il peut renvoyer le cookie s'il n'a pas reçu celui-ci depuis un temps supérieur à D. Rien n'empêche un client de ne pas observer ce code de déontologie. Les navigateurs eux le respectent. Certains navigateurs donnent accès au contenu des cookies qu'ils reçoivent. C'est le cas du navigateur Mozilla. Voici par exemple les informations liées au cookie envoyé par le serveur dans un exemple précédent :

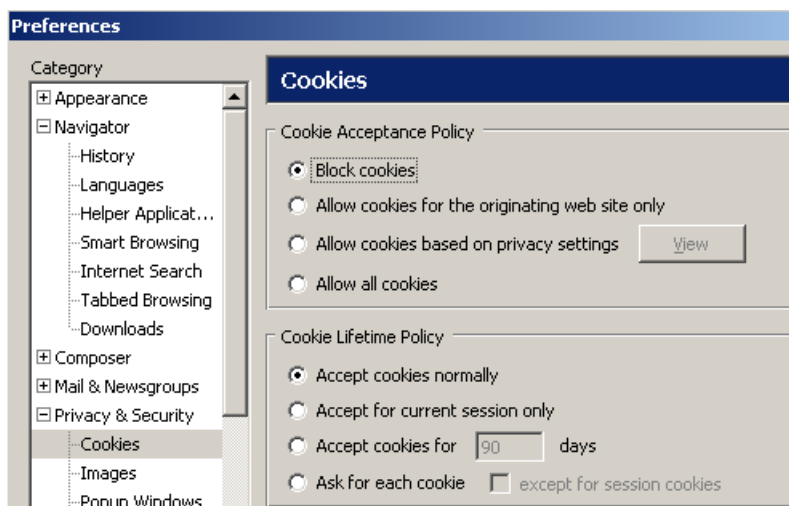


On y trouve :

- le nom du cookie [ASP.NET_SessionId]
- sa valeur [y153...m3]
- la machine à laquelle il est associé [localhost]
- l'url à laquelle il est associé [/]
- sa durée de vie [at end of session]

Le navigateur enverra donc le jeton de session à chaque fois qu'il demandera une URL de la forme [http://localhost/...], c.a.d. à chaque fois qu'il demandera une url au serveur web de la machine [localhost]. La durée de vie du cookie est celle de la session. Pour le navigateur, cela entraîne que le cookie n'expire jamais. Il l'enverra à chaque fois qu'il demandera une url de la machine [localhost]. Ainsi si le navigateur reçoit le jeton de session le jour J, qu'on le ferme et qu'on le réutilise le lendemain, il renverra alors le jeton de session (qui a été conservé dans un fichier). Le serveur recevra ce jeton que lui n'a plus, car un jeton de session a une durée de vie limitée sur le serveur (20 mn sur IIS). Aussi démarrera-t-il une nouvelle session.

Il est possible d'inhiber l'utilisation des cookies sur un navigateur. Dans ce cas, le client reçoit bien le jeton de session mais ne le renvoie pas ce qui empêche le suivi de session. Pour le montrer, nous inhibons l'utilisation des cookies sur notre navigateur (Mozilla ici) :



Par ailleurs, nous supprimons tous les cookies existant :



Ceci fait, nous relançons le serveur Cassini pour repartir de zéro et avec le navigateur, nous demandons de nouveau l'url [http://localhost/aspnet/session1/main.aspx] :

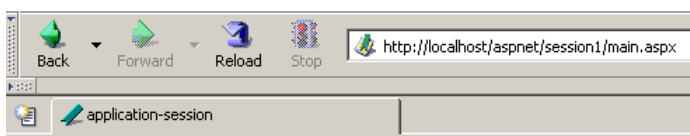


jeton de session : nqf12zlmrvh4h55euzrcji
requêtes Application : 1
requêtes Client : 1

Regardons si notre navigateur a stocké un cookie :



Nous constatons que le navigateur n'a pas stocké le cookie du jeton de session que le serveur lui a envoyé. On peut donc s'attendre à ce qu'il n'y ait pas de suivi de session. Nous redemandons la même url (Reload) :



jeton de session : bksblq550d2hpcfy35nh355
requêtes Application : 2
requêtes Client : 1

On a bien ce qui était attendu. Le navigateur n'a pas renvoyé le jeton de session, qu'il avait pourtant reçu mais pas stocké. Le serveur a donc commencé une nouvelle session avec un nouveau jeton. On retiendra de cet exemple que notre politique de suivi de session est mise à mal si l'utilisateur a inhibé l'utilisation des cookies sur son navigateur. Il y a cependant une autre façon que les cookies, d'échanger le jeton de session entre serveur et client. Il est en effet possible de signaler au serveur web que l'application travaille sans cookie. Cela se fait au moyen du fichier de configuration [web.config] :

```
<?xml version="1.0" encoding="UTF-8" ?>  
<configuration>  
  <system.web>  
    <sessionState cookieless="true" timeout="10" />  
  </system.web>  
</configuration>
```

Le fichier de configuration ci-dessus indique que l'application va travailler sans cookies (cookieless="true") et que la durée d'inactivité maximale d'un jeton de session est de 10 mn (timeout="10"). Après ce délai, la session associée au jeton est détruite. Le processus d'échange du jeton de session entre le serveur et le client va être le suivant :

1. le client demande l'url [http://machine:port/V/chemin] où V est un dossier virtuel du serveur web
2. le serveur génère un jeton J et répond au client de se rediriger vers l'url [http://machine:port/V/(J)/chemin]. Il a donc placé le jeton dans l'url à interroger, immédiatement derrière le dossier virtuel V
3. le client obéit à cette redirection et demande la nouvelle URL [[http://machine:port/V/(J)/chemin].
4. le serveur répond à cette demande et envoie une page de réponse.

Illustrons ces différents points. Nous mettons la totalité de l'application précédente dans un nouveau dossier <application-path>. Nous plaçons dans ce même dossier le fichier [web.config] précédent. Par ailleurs, nous modifions le code de présentation [main.aspx] pour y inclure un lien :

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>  
<HTML>  
  <HEAD>  
    <title>application-session</title>  
  </HEAD>  
  <body>  
    jeton de session :  
    <% =jeton %>  
    <br>  
    requêtes Application :  
    <% =nbRequêtesApplication %>  
    <br>  
    requêtes Client :  
    <% =nbRequêtesClient %>
```

```

<br>
<a href="main.aspx">Recharger l'application</a>
</body>
</HTML>

```

Ce lien pointe sur la page [main.aspx] et est donc équivalent au bouton (Reload) du navigateur. Le serveur Cassini est lancé avec les paramètres (<application-path>, /session2). Nous dérogeons à notre habitude qui consistait à noter le dossier virtuel [/aspnet/XX]. En effet, à cause de l'insertion du jeton de session dans l'url, le dossier virtuel ne doit avoir qu'un élément /XX. Nous utilisons tout d'abord le client [curl] pour demander l'url [http://localhost/session2/main.aspx] :

```

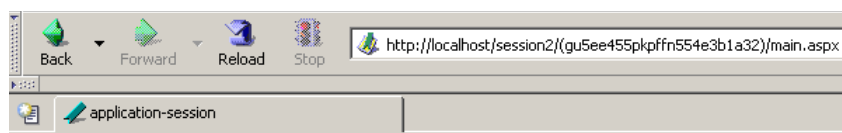
E:\curl>curl --include http://localhost/session2/main.aspx
HTTP/1.1 302 Found
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 13:52:36 GMT
X-AspNet-Version: 1.1.4322
Location: /session2/(hinadjag3bt0u155g5hqe245)/main.aspx
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 163
Connection: Close

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href='/session2/(hinadjag3bt0u155g5hqe245)/main.aspx'>here
</body></html>

```

Nous voyons que le serveur répond par l'entête HTTP [HTTP/1.1 302 Found] au lieu de [HTTP/1.1 200 OK]. C'est une entête qui demande au client de se rediriger vers l'url indiquée par l'entête HTTP Location [Location: /session2/(hinadjag3bt0u155g5hqe245)/main.aspx]. On voit le jeton de session qui a été inséré dans l'url de redirection. Un navigateur recevant cette réponse, demande la nouvelle url de façon transparente pour l'utilisateur qui ne voit pas la nouvelle requête. Au cas où le navigateur ne gèrerait pas seul la redirection, un document HTML est envoyé derrière le code HTTP ci-dessus. On y trouve un lien sur l'url de redirection, lien sur lequel l'utilisateur pourra cliquer.

Maintenant, faisons la même chose avec un navigateur où les cookies ont été inhibés. Nous demandons là encore, l'url [http://localhost/session2/main.aspx]. Nous obtenons la réponse suivante du serveur :



jeton de session : gu5ee455pkpffn554e3b1a32
 requêtes Application : 1
 requêtes Client : 1
[Recharger l'application](#)

Tout d'abord, constatons que l'url affichée par le navigateur n'est pas celle que nous avons demandée. C'est le signe qu'une redirection a eu lieu. En effet, le navigateur affiche toujours l'URL du dernier document reçu. Si donc, il n'affiche pas l'url [http://localhost/session2/main.aspx], c'est qu'on lui a demandé de se rediriger vers une autre url. Il peut y avoir plusieurs redirections. L'url affichée par le navigateur est l'url de la dernière redirection. Nous pouvons constater que le jeton de session est présent dans l'url affichée par le navigateur. On peut le voir car ce jeton est également affiché par notre programme dans la page.

Rappelons le code du lien qui a été placé dans la page :

```

<a href="main.aspx">Recharger l'application</a>

```

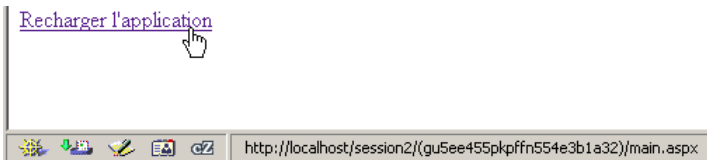
C'est un lien relatif puisqu'il ne commence pas par le signe / qui en ferait un lien absolu. Relatif à quoi ? Pour comprendre ce point, il faut revenir à l'url du document actuellement affiché : [http://localhost/session2/(gu5ee455pkpffn554e3b1a32)/main.aspx]. Les liens relatifs qui seront trouvés dans ce document seront relatifs au chemin [http://localhost/session2/(gu5ee455pkpffn554e3b1a32)]. Ainsi notre lien ci-dessus est-il équivalent au lien :

```

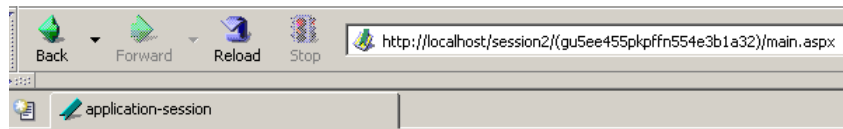
<a href=" http://localhost/session2/(gu5ee455pkpffn554e3b1a32)/main.aspx">Recharger l'application</a>

```

C'est ce que nous montre le navigateur si on passe la souris sur le lien :



Si nous cliquons sur le lien [Recharger l'application], c'est donc l'url [http://localhost/session2/(gu5ee455pkpffn554e3b1a32)/main.aspx] qui est appelée. Le serveur va donc recevoir le jeton de session et pouvoir retrouver les informations qui lui sont liées. C'est ce que nous montre sa réponse rveur :



jeton de session : gu5ee455pkpffn554e3b1a32
 requêtes Application : 2
 requêtes Client : 2
[Recharger l'application](#)

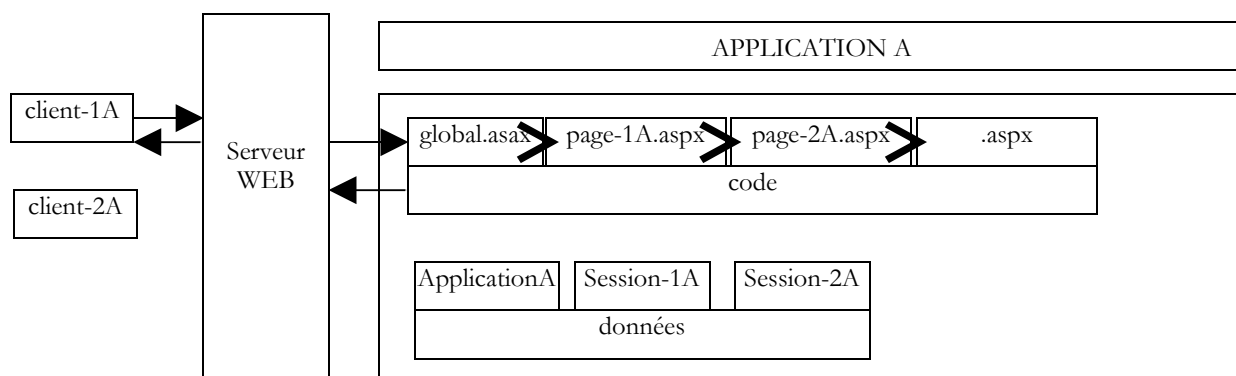
Nous retiendrons que si nous avons besoin de faire un suivi de session dans une application web et que nous ne sommes pas sûrs que les navigateurs clients de cette application vont autoriser l'utilisation des cookies, alors

- on doit configurer l'application pour qu'elle travaille sans cookies
- les pages de l'application doivent comporter des liens relatifs et non absolus

3.2 Récupérer les informations d'une requête client

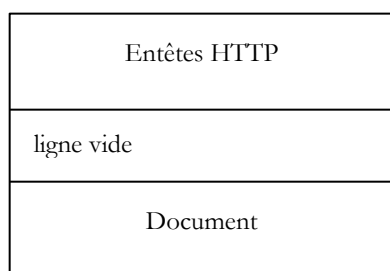
3.2.1 Le cycle requête-réponse du client-serveur web

Rappelons ici le contexte client-serveur d'une application web :



La requête d'un client pour une application web est traitée de fa façon suivante :

1. le client ouvre une connexion tcp-ip vers un port P du service web de la machine M abritant l'application web
2. il envoie sur cette connexion une suite de lignes de texte selon le protocole HTTP. Cet ensemble de lignes forme ce qu'on appelle la requête du client. Elle a la forme suivante :



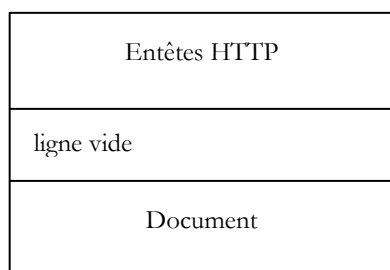
Une fois la demande envoyée, le client va attendre la réponse.

3. la première ligne des entêtes HTTP précise l'action demandée au serveur web. Elle peut avoir plusieurs formes :

- GET url HTTP/<version>, avec <version> égal actuellement à 1.0 ou 1.1. Dans ce cas, la requête ne comprend pas la partie [Document]
- POST url HTTP/<version>. Dans ce cas, la requête comprend une partie [Document], le plus souvent une liste d'informations à destination de l'application web
- PUT url HTTP/<version>. Le client envoie un document dans la partie [Document] et veut le stocker sur le serveur à l'adresse url

Lorsque le client souhaite transmettre des informations à l'application web à laquelle il s'est connecté, il dispose principalement de deux moyens :

- sa demande est [GET url_enrichie HTTP/<version>] où url_enrichie est de la forme [url?param1=val1¶m2=val2&...]. Le client transmet outre l'url, une série d'informations sous la forme [clé=valeur].
 - sa demande est [POST url HTTP/<version>]. Dans la partie [Document], il transmet des informations sous la même forme que précédemment : [param1=val1¶m2=val2&...].
4. sur le serveur, l'ensemble de la chaîne de traitement de la requête du client a accès à celle-ci via un objet global appelé **Request**. Le serveur web a placé dans cet objet la totalité de la requête du client sous une forme que nous allons découvrir. L'application sollicitée va traiter cet objet et construire une réponse au client. Celle-ci est disponible dans un objet global appelé **Response**. Le rôle de l'application web est de construire un objet [**Response**] à partir de l'objet [**Request**] reçu. La chaîne de traitement dispose également des objets globaux [**Application**] et [**Session**] dont nous avons déjà parlé et qui vont lui permettre de partager des données entre clients différents (Application) ou entre requêtes successives d'un même client (Session).
 5. l'application va envoyer sa réponse au serveur au moyen de l'objet [Response]. Celle-ci, une fois sur le réseau aura la forme HTTP suivante :



Une fois cette réponse envoyée, le serveur va fermer la connexion réseau en réception (sauf si le client lui a dit de ne pas le faire).

6. le client va recevoir la réponse et va fermer à son tour la connexion (en émission). Ce qui sera fait de cette réponse dépend du type du client. Si celui-ci est un navigateur, et que le document reçu est un document HTML, celui-ci sera affiché. Si le client est un programme, la réponse va être analysée et exploitée.
7. Le fait qu'après le cycle requête-réponse, la connexion qui liait le client au serveur soit fermée fait du protocole HTTP un protocole sans état. Lors de la requête suivante, le client établira une nouvelle connexion réseau au même serveur. Du fait que ce n'est plus la même connexion réseau, le serveur n'a aucune possibilité (au niveau tcp-ip et HTTP) de lier cette nouvelle connexion à une précédente. C'est le système du jeton de session qui permettra ce lien.

3.2.2 Récupérer les informations transmises par le client

Nous examinons maintenant certaines propriétés et méthodes de l'objet [Request] qui permet au code de l'application d'avoir accès à la requête du client et donc aux informations qu'il a transmises. L'objet [Request] est de type [HttpRequest] :

[System.Object](#)
System.Web.HttpRequest

```
NotInheritable Public Class HttpRequest
```

Cette classe a de nombreuses propriétés et méthodes. Nous nous intéressons aux propriétés **HttpMethod**, **QueryString**, **Form** et **Params** qui vont nous permettre d'avoir accès aux éléments de la chaîne d'informations [param1=val1¶m2=val2&...].

HttpMethod as String	méthode de requête du client : GET, POST, HEAD, ...
QueryString as NameValueCollection	collection des éléments de la chaîne de requête param1=val1¶m2=val2&.. de la 1ère ligne HTTP [méthode]?param1=val1¶m2=val2&... où [méthode] peut être GET, POST, HEAD.
Form as NameValueCollection	collection des éléments de la chaîne de requête param1=val1¶m2=val2&.. se trouvant dans la partie [Document] de la requête (méthode POST).
Params as NameValueCollection	rassemble plusieurs collections : QueryString, Form, ServerVariables, Cookies au sein d'une unique collection.

3.2.3 Exemple 1

Mettons en oeuvre ces éléments sur un premier exemple. L'application n'aura qu'un élément [main.aspx]. Le code de présentation [main.aspx] sera le suivant :

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<html>
<head>
<title>Requête client</title>
</head>
<body>
Requête :
<% = méthode %>
<br />
nom :
<% = nom %>
<br />
âge :
<% = age %>
<br />
</body>
</html>
```

La page affiche trois informations [méthode, nom, age] calculées par sa partie contrôleur [main.aspx.vb] :

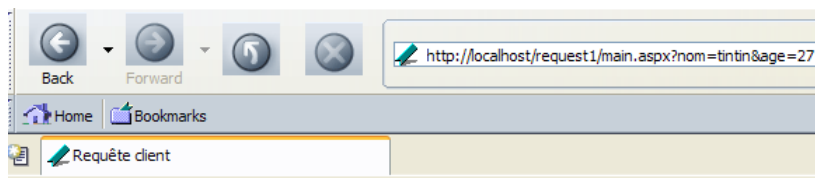
```
Public Class main
    Inherits System.Web.UI.Page

    Protected nom As String = "xx"
    Protected age As String = "yy"
    Protected méthode As String

    Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Init
        ' on mémorise la requête du client dans request.txt du dossier de l'application
        Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
        Me.Request.SaveAs(requestFileName, True)
    End Sub

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère les paramètres de la requête
        méthode = Request.HttpMethod.ToLower
        If Not Request.QueryString("nom") Is Nothing Then nom = Request.QueryString("nom").ToString
        If Not Request.QueryString("age") Is Nothing Then age = Request.QueryString("age").ToString
        If Not Request.Form("nom") Is Nothing Then nom = Request.Form("nom").ToString
        If Not Request.Form("age") Is Nothing Then age = Request.Form("age").ToString
    End Sub
End Class
```

Lorsque la page est chargée (Form_Load), les informations [nom, age] sont récupérées dans la requête du client. On les recherche dans les deux collections [QueryString] et [Form]. . Par ailleurs, dans [Page_Init], nous mémorisons la requête du client afin de pouvoir vérifier ce qu'il a envoyé. Nous plaçons ces deux fichiers dans un dossier <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/request1), puis avec un navigateur nous demandons l'url [http://localhost/request1/main.aspx?nom=tintin&age=27] . Nous obtenons la réponse suivante :



```
Requête : get
nom : tintin
âge : 27
```

Les informations transmises par le client ont été récupérées correctement. La requête du navigateur mémorisée dans le fichier [request.txt] est la suivante :

```
GET /request1/main.aspx?nom=tintin&age=27 HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Keep-Alive: 300
```

```
Accept: application/x-shockwave-  
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp  
eg,image/gif;q=0.2,*/*;q=0.1  
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7  
Accept-Encoding: gzip,deflate  
Accept-Language: en-us,en;q=0.5  
Host: localhost  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

On voit que le navigateur a fait une requête GET. Pour faire une requête POST, nous allons utiliser le client [curl]. Dans une fenêtre Dos, nous tapons la commande suivante :

```
C:\curl>curl --include --data nom=tintin --data age=27 http://localhost/request1/main.aspx
```

--include pour afficher les entêtes HTTP de la réponse
--data param=valeur pour envoyer l'information param=valeur au moyen d'un POST

La réponse du serveur est la suivante :

```
HTTP/1.1 200 OK  
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0  
Date: Fri, 02 Apr 2004 09:27:25 GMT  
Cache-Control: private  
Content-Type: text/html; charset=utf-8  
Content-Length: 178  
Connection: Close
```

```
<html>  
  <head>  
    <title>Requête client</title>  
  </head>  
  <body>  
    Requête :  
    post  
    <br />  
    nom :  
    tintin  
    <br />  
    âge :  
    27  
    <br />  
  </body>  
</html>
```

Le serveur a bien, là encore, récupéré les paramètres envoyés cette fois-ci par un POST. Pour s'assurer de ce dernier point, on peut vérifier le contenu du fichier [request.txt] :

```
POST /request1/main.aspx HTTP/1.1  
Pragma: no-cache  
Content-Length: 17  
Content-Type: application/x-www-form-urlencoded  
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*  
Host: localhost  
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4  
  
nom=tintin&age=27
```

Le client [curl] a bien fait un POST. Maintenant, mélangeons les deux méthodes de passage d'information. Nous mettons [age] dans l'url demandée et [nom] dans le document posté :

```
E:\curl>curl --include --data nom="tintin" http://localhost/request1/main.aspx?age=27
```

La requête envoyée par [curl] est la suivante (request.txt) :

```
POST /request1/main.aspx?age=27 HTTP/1.1  
Pragma: no-cache  
Content-Length: 10  
Content-Type: application/x-www-form-urlencoded  
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*  
Host: localhost  
User-Agent: curl/7.10.8 (win32) libcurl/7.10.8 OpenSSL/0.9.7a zlib/1.1.4  
  
nom=tintin
```

On voit que l'âge est passé dans l'url demandée. On l'obtiendra dans la collection [QueryString]. Le nom est lui passé dans le document envoyé à cette url. On l'obtiendra dans la collection [Form]. La réponse obtenue par le client [curl] :

```
<html>
  <head>
    <title>Requête client</title>
  </head>
  <body>
    Requête :
    post
    <br />
    nom :
    tintin
    <br />
    âge :
    27
    <br />
  </body>
</html>
```

Enfin n'envoyons aucune information au serveur :

```
E:\curl>curl --include http://localhost/request1/main.aspx
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Fri, 02 Apr 2004 12:43:14 GMT
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 173
Connection: Close
```

```
<html>
  <head>
    <title>Requête client</title>
  </head>
  <body>
    Requête :
    get
    <br />
    nom :
    xx
    <br />
    âge :
    yy
    <br />
  </body>
</html>
```

Le lecteur est invité à relire le code du contrôleur [main.aspx.vb] pour comprendre cette réponse.

3.2.4 Exemple 2

Il est possible pour le client d'envoyer plusieurs valeurs pour une même clé. Ainsi qu'arrive-t-il si dans l'exemple précédent on demande l'url [http://localhost/request1/main.aspx?nom=tintin&age=27&nom=milou] où il y a deux fois la clé [nom] ? Essayons avec un navigateur :



```
Requête : get
nom : tintin,milou
âge : 27
```

Notre application a bien récupéré les deux valeurs associées à la clé [nom]. L'affichage est un peu trompeur. Il a été obtenu par l'instruction

```
| If Not Request.QueryString("nom") Is Nothing Then nom = Request.QueryString("nom").ToString |
```


La méthode [ToString] a produit la chaîne [tintin,milou] qui a été affichée. Elle cache le fait qu'en réalité l'objet [Request.QueryString("nom")] est un tableau de chaînes de caractères {"tintin","milou"}. L'exemple suivant met ce point en évidence. La page [main.aspx] de présentation sera la suivante :

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<HTML>
<HEAD>
<title>Requête client</title>
</HEAD>
<body>
<P>Informations passées par le client :</P>
<form runat="server">
<P>QueryString :</P>
<P><asp:listbox id="lstQueryString" runat="server" EnableViewState="False" Rows="6"></asp:listbox></P>
<P>Form :</P>
<P><asp:listbox id="lstForm" runat="server" EnableViewState="False" Rows="2"></asp:listbox></P>
</form>
</body>
</HTML>
```

Il y a des nouveautés dans cette page qui utilise ce qu'on appelle des contrôles serveur. Ils sont caractérisés par l'attribut [runat="server"]. Il est trop tôt pour introduire la notion de contrôle serveur. Il suffit de savoir qu'ici :

- que la page a deux listes (balises <asp:listbox>)
- que ces listes sont des objets (lstQueryString, lstForm) de type [ListBox] qui seront construits par le contrôleur de la page
- que ces objets n'ont d'existence qu'au sein du serveur web. Au moment de la réponse, ils seront transformés en balises HTML classiques que le client pourra comprendre. Un objet [listbox] sera ainsi transformé (on dit aussi rendu) en balises HTML <select> et <option>.
- que l'intérêt principal de ces objets est de débarrasser le code de présentation de tout code VB, celui-ci restant confiné au contrôleur.

Le contrôleur [main.aspx.vb] chargé de construire les deux objets [lstQueryString] et [lstForm] est le suivant :

```
Imports System.Collections
Imports System
Imports System.Collections.Specialized

Public Class main
    Inherits System.Web.UI.Page

    Protected WithEvents lstQueryString As System.Web.UI.WebControls.ListBox
    Protected WithEvents lstForm As System.Web.UI.WebControls.ListBox

    Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Init
        ' on mémorise la requête du client dans request.txt du dossier de l'application
        Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
        Me.Request.SaveAs(requestFileName, True)
    End Sub

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' variables locales
        Protected infosQueryString As ArrayList
        Protected infosForm As ArrayList

        ' on récupère toute la collection des informations du QueryString
        infosQueryString = getValeurs(Request.QueryString)
        ' on associe les données au ListBox
        lstQueryString.DataSource = infosQueryString
        lstQueryString.DataBind()
        ' on fait la même chose avec les données du POST
        infosForm = getValeurs(Request.Form)
        lstForm.DataSource = infosForm
        lstForm.DataBind()
    End Sub

    Private Function getValeurs(ByRef data As NameValueCollection) As ArrayList
        ' au départ une liste d'infos vide
        Dim infos As New ArrayList
        ' on récupère les clés de la collection
        Dim clés() As String = data.AllKeys
        ' on parcourt le tableau des clés
        Dim valeurs() As String
        For Each clé As String In clés
            ' valeurs associées à la clé
            valeurs = data.GetValues(clé)
            ' une seule valeur ?
            If valeurs.Length = 1 Then
                infos.Add(clé + "=" + valeurs(0))
            Else

```

```

' plusieurs valeurs
For ivalue As Integer = 0 To valeurs.Length - 1
    infos.Add(clé + "(" + ivalue.ToString + ")=" + valeurs(ivalue))
Next
End If
Next
' on rend le résultat
Return infos
End Function
End Class

```

Les points importants de ce code sont les suivants :

- dans [Form_Load] la page récupère les deux collection [QueryString] et [Form]. Elle utilise une fonction [getValeurs] pour mettre le contenu de ces deux collections dans deux objets de type [ArrayList] qui contiendront des chaînes de caractères du type [clé=valeur] si la clé de la collection est associée à une unique valeur ou [clé(i)=valeur] si la clé est associée à plusieurs valeurs.
- chacun des objets [ArrayList] est attaché ensuite à l'un des objets [ListBox] de la page de présentation au moyen de deux instructions :
 - [ListBox.DataSource=ArrayList] et [ListBox.DataBind]. Cette dernière instruction transfère les éléments de [DataSource] dans la collection [Items] de l'objet [ListBox]
 on remarquera qu'aucun des deux objets [ListBox] n'est créé explicitement par une opération [New]. On en déduira qu'en présence de la balise <asp:listbox id="xx">...<asp:listbox/>, le serveur web crée lui-même l'objet [ListBox] référencé par l'attribut [id] de la balise.
- la fonction [getValeurs] exploite l'objet de type [NameValueCollection] qu'on lui passe en paramètre pour produire un résultat de type [ArrayList].

Nous plaçons les deux fichiers précédents dans un dossier <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>/request2), puis nous demandons l'url [http://localhost/request2/main.aspx?nom=tintin&age=27]. Nous obtenons la réponse suivante :



Informations passées par le client :

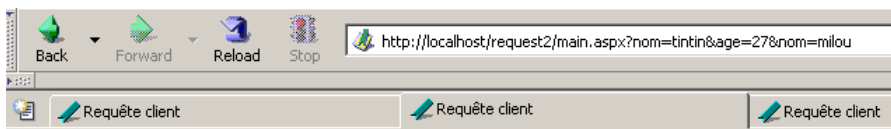
QueryString :



Form :

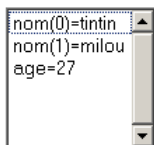


Nous demandons maintenant une url où la clé [nom] est présente deux fois :



Informations passées par le client :

QueryString :



Nous constatons que l'objet [Request.QueryString("nom")] était bien un tableau. Ici, les requêtes étaient faites par une méthode GET. Nous utilisons le client [curl] pour faire une requête POST :

```
E:\curl>curl --data nom=milou --data nom=tintin --data age=14 --data age=27 http://localhost/request2/main.aspx
```

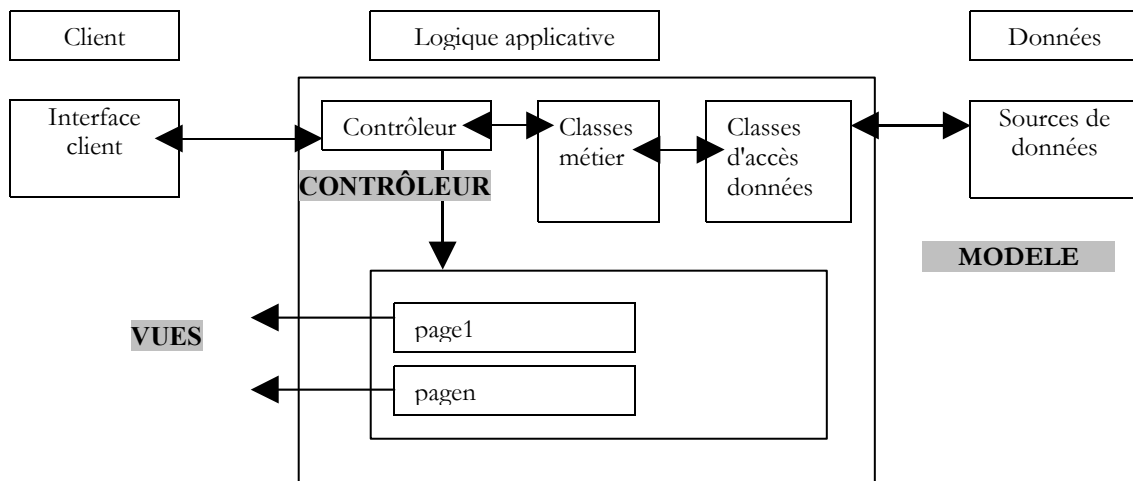
```
<HTML>
  <HEAD>
    <title>Requête client</title>
  </HEAD>
  <body>
    <P>Informations passées par le client :</P>
    <form name="_ctl0" method="post" action="main.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwtMTI3MjA1MzUzMTs7PtCDC7NG4riDYIB4YjyGFpVAaAvid" />
      <P>QueryString :</P>
      <P><select name="lstQueryString" size="6" id="lstQueryString">
</select></P>
      <P>Form :</P>
      <P><select name="lstForm" size="2" id="lstForm">
        <option value="nom(0)=milou">nom(0)=milou</option>
        <option value="nom(1)=tintin">nom(1)=tintin</option>
        <option value="age(0)=14">age(0)=14</option>
        <option value="age(1)=27">age(1)=27</option>
      </select></P>
    </form>
  </body>
</HTML>
```

On peut voir que le client reçoit bien du code HTML classique pour les deux listes de la page. On voit apparaître des informations qu'on n'a pas mises nous-mêmes telles le champ caché [_VIEWSTATE]. Ces informations ont été générées par les balises <asp:xx runat="server">. Nous devons apprendre à maîtriser celles-ci.

3.3 Mise en oeuvre d'une architecture MVC

3.3.1 Le concept

Terminons ce long chapitre par la mise en oeuvre d'une application construite selon le modèle MVC (Model-View-Controller). Une application web architecturée selon ce modèle ressemble à ceci :



- le client adresse ses requêtes à une entité particulière de l'application appelée le contrôleur
- le contrôleur analyse la requête du client et la fait exécuter. Pour cela il se fait aider par des classes regroupant la logique métier de l'application et des classes d'accès aux données.
- selon le résultat de l'exécution de la requête, le contrôleur choisit d'envoyer une certaine page en réponse au client

Dans ce modèle, toutes les requêtes passent par un unique contrôleur qui est le chef d'orchestre de toute l'application web. L'intérêt de ce modèle est qu'on peut regrouper dans le contrôleur tout ce qui doit être fait avant chaque requête. Supposons par exemple que l'application nécessite une authentification. Celle-ci est faite une unique fois. Une fois réussie, l'application va mettre dans la session, des informations liées à l'utilisateur qui vient de s'authentifier. Comme un client peut appeler directement une page de l'application sans s'authentifier, chaque page devra donc vérifier dans la session que l'authentification a bien été faite. Si toutes les requêtes passent par un unique contrôleur, c'est lui qui peut faire ce travail. Les pages à qui la requête sera éventuellement passée n'auront pas à le faire.

3.3.2 Contrôler une application MVC sans session

De ce que nous avons vu jusqu'ici, on peut penser que le fichier [global.asax] pourrait jouer le rôle du contrôleur. En effet, on sait que toutes les requêtes passent par lui. Il est donc bien placé pour tout contrôler. L'application qui suit l'utilise à cette fin. Son chemin virtuel sera [http://localhost/mvc1]. Pour indiquer ce qu'il veut, le client passera derrière l'url un paramètre action=valeur. Selon la valeur du paramètre [action], le contrôleur [global.asax] dirigera la requête vers une page particulière :

1. [main.aspx] si le paramètre action n'est pas défini ou si action=main
2. [action1.aspx] si action=action1
3. [inconnu.aspx] si action ne tombe pas dans les cas 1 et 2

Les pages [main.aspx, action1.aspx, inconnu.aspx] se contentent d'afficher la valeur de [action] qui a provoqué leur affichage. Nous listons ci-dessous les huit fichiers de cette application et nous les commentons lorsque c'est nécessaire :

```
[global.asax]
|<%@ Application src="Global.asax.vb" Inherits="Global" %>
```

```
[global.asax.vb]
```

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
        ' on récupère l'action à faire
        Dim action As String
        If Request.QueryString("action") Is Nothing Then
            action = "main"
        Else
            action = Request.QueryString("action").ToString.ToLower
        End If
        ' on met l'action dans le contexte de la requête
        Context.Items("action") = action
        ' on exécute l'action
        Select Case action
            Case "main"
                Server.Transfer("main.aspx", True)
            Case "action1"
                Server.Transfer("action1.aspx", True)
            Case Else
                Server.Transfer("inconnu.aspx", True)
        End Select
    End Sub
End Class
```

Les points à noter :

- nous interceptons toutes les requêtes du client dans la procédure [Application_BeginRequest] qui est automatiquement exécutée au démarrage de chaque nouvelle requête faite à l'application.
- dans cette procédure, nous avons accès à l'objet [Request] qui est l'image de la requête HTTP du client. Comme nous attendons une url de la forme [http://localhost/mvc1/main.aspx?action=xx], nous cherchons une clé [action] dans la collection [Request.QueryString]. Si elle n'y est pas, on fixe par défaut [action] égal à [main].
- la valeur du paramètre [action] est placée dans l'objet [Context]. Comme les objets [Application, Session, Request, Response, Server], cet objet est global et accessible dans tout code. Cet objet est passé de page en page si la requête est traitée par plusieurs pages comme cela va être le cas ici. Il est supprimé dès que la réponse a été envoyée au client. Sa durée de vie est donc celle du traitement de la requête.
- selon la valeur du paramètre [action], on passe la requête à la page appropriée. Pour cela on utilise l'objet global [Server] qui grâce à sa méthode permet de transférer la requête courante à une autre page. Son premier paramètre est le nom de la page cible, le second un booléen indiquant si on doit ou non transférer à la page cible les collections [QueryString] et [Form]. Ici, c'est oui.

Les fichiers [main.aspx] et [main.aspx.vb] :

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<HTML>
<head>
<title>main</title></head>
<body>
<h3>Page [main]</h3>
Action : <% =action %>
</body>
```

Les fondamentaux du développement asp.net

```
</HTML>
```

```
Public Class main
    Inherits System.Web.UI.Page

    Protected action As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère l'action en cours
        action = Me.Context.Items("action").ToString
    End Sub
End Class
```

Le contrôleur [main.aspx.vb] se contente de récupérer la valeur de la clé [action] dans le contexte, cette valeur étant affichée par le code de présentation. On cherche ici à montrer le passage de l'objet [Context] entre différentes pages traitant une même requête client. Les pages [action1.aspx] et [inconnu.aspx] ont un fonctionnement analogue :

[action1.aspx]

```
<%@ Page src="action1.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="action1" %>
<HTML>
<head>
    <title>action1</title></head>
<body>
    <h3>Page [action1]</h3>
    Action : <% =action %>
</body>
</HTML>
```

[action1.aspx.vb]

```
Public Class action1
    Inherits System.Web.UI.Page

    Protected action As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère l'action en cours
        action = Me.Context.Items("action").ToString
    End Sub
End Class
```

[inconnu.aspx]

```
<%@ Page src="inconnu.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="inconnu" %>
<HTML>
<head>
    <title>inconnu</title></head>
<body>
    <h3>Page [inconnu]</h3>
    Action : <% =action %>
</body>
</HTML>
```

[inconnu.aspx.vb]

```
Public Class inconnu
    Inherits System.Web.UI.Page

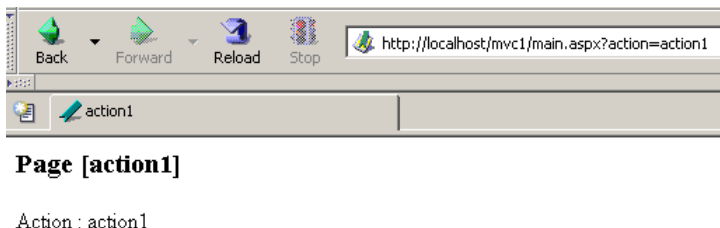
    Protected action As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère l'action en cours
        action = Me.Context.Items("action").ToString
    End Sub
End Class
```

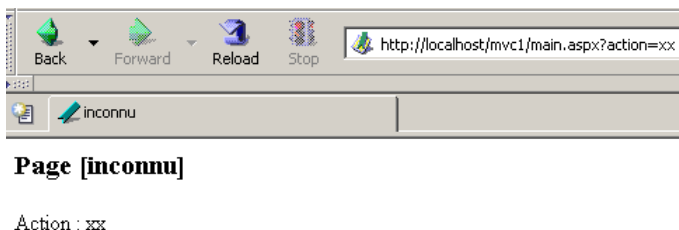
Pour tester, les documents précédents sont placés dans un dossier <application-path> et Cassini lancé avec les paramètres (<application-path>,/mvc1). Nous demandons l'url [http://localhost/mvc1/main.aspx] :



La requête n'a envoyé aucun paramètre [action]. Le code du contrôleur de l'application [global.asax.vb] a fait délivrer la page [main.aspx]. Maintenant nous demandons l'url [http://localhost/mvc1/main.aspx?action=action1] :



Le code du contrôleur de l'application [global.asax.vb] a fait délivrer la page [action1.aspx]. Maintenant nous demandons l'url [http://localhost/mvc1/main.aspx?action=xx] :



L'action n'a pas été reconnue et le contrôleur [global.asax.vb] a fait délivrer la page [inconnu.aspx].

3.3.3 Contrôler une application MVC avec session

La plupart du temps, les différentes requêtes d'un client pour une application doivent se partager des informations. On a vu une solution possible à ce problème : stocker les informations à partager dans l'objet [Session] de la requête. Cet objet est en effet partagé par toutes les requêtes et est capable de mémoriser des informations sous la forme (clé,valeur) ou clé est de type [String] et valeur est tout type dérivé de [Object].

Dans l'exemple précédent, les différentes pages associées aux différentes actions étaient appelées dans la procédure [Application_BeginRequest] du fichier [global.asax.vb] :

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' on récupère l'action à faire
    Dim action As String
    If Request.QueryString("action") Is Nothing Then
        action = "main"
    Else
        action = Request.QueryString("action").ToString.ToLower
    End If
    ' on met l'action dans le contexte de la requête
    Context.Items("action") = action
    ' on exécute l'action
    Select Case action
        Case "main"
            Server.Transfer("main.aspx", True)
        Case "action1"
            Server.Transfer("action1.aspx", True)
        Case Else
            Server.Transfer("inconnu.aspx", True)
    End Select
End Sub
```

Il se trouve que dans la procédure [Application_BeginRequest] l'objet [Session] n'est pas accessible. Il en est de même dans la page à laquelle l'exécution est transférée. Aussi ce modèle n'est-il pas utilisable pour une application avec session. Nous pouvons faire

jouer le rôle du contrôleur à toute page, par exemple [default.aspx]. Les fichiers [global.asax, global.asax.vb] disparaissent alors pour être remplacés par les fichiers [default.aspx, default.aspx.vb] :

[default.aspx]

```
<%@ Page src="default.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="controleur" %>
```

[default.aspx.vb]

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class controleur
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère l'action à faire
        Dim action As String
        If Request.QueryString("action") Is Nothing Then
            action = "main"
        Else
            action = Request.QueryString("action").ToString.ToLower
        End If

        ' on met l'action dans le contexte de la requête
        Context.Items("action") = action
        ' on récupère l'action précédente si elle existe
        Context.Items("actionPrec") = Session.Item("actionPrec")
        If Context.Items("actionPrec") Is Nothing Then Context.Items("actionPrec") = ""
        ' on mémorise l'action courante dans la session
        Session.Item("actionPrec") = action

        ' on exécute l'action
        Select Case action
            Case "main"
                Server.Transfer("main.aspx", True)
            Case "action1"
                Server.Transfer("action1.aspx", True)
            Case Else
                Server.Transfer("inconnu.aspx", True)
        End Select
    End Sub
End Class
```

Afin de mettre en évidence, le mécanisme de session, les différentes pages vont afficher outre l'action courante, l'action qui a précédé. Pour une suite d'actions A_1, A_2, \dots, A_n , lorsque l'action A_i se produit, le contrôleur ci-dessus :

- met l'action courante A_i dans le contexte
- retrouve dans la session l'action A_{i-1} qui a précédé. Dans le cas où il n'y en a pas (cas de l'action A_1), la chaîne vide est affectée à l'action précédente.
- met l'action courante A_i dans la session en remplacement de A_{i-1}
- transfère l'exécution à la page adéquate

Les trois pages de l'application sont les suivantes :

[main.aspx]

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<HTML>
<HEAD>
<title>main</title>
</HEAD>
<body>
<h3>Page [main]</h3>
Action courante :
<% =action %>
<br>
Action précédente :
<% =actionPrec %>
</body>
</HTML>
```

[action1.aspx]

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
```

Les fondamentaux du développement asp.net

```

<HTML>
<head>
  <title>action1</title></head>
<body>
  <h3>Page [action1]</h3>
  Action courante :
  <% =action %>
  <br>
  Action précédente :
  <% =actionPrec %>
</body>
</HTML>

```

[inconnu.aspx]

```

<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
<HTML>
<head>
  <title>inconnu</title>
</head>
<body>
  <h3>Page [inconnu]</h3>
  Action courante :
  <% =action %>
  <br>
  Action précédente :
  <% =actionPrec %>
</body>
</HTML>

```

Parce que les trois pages affichent les mêmes informations [action, actionPrec], elles peuvent avoir toutes les trois le même contrôleur de page. On les a donc toutes fait dériver de la classe [main] du fichier [main.aspx.vb] :

```

Public Class main
  Inherits System.Web.UI.Page

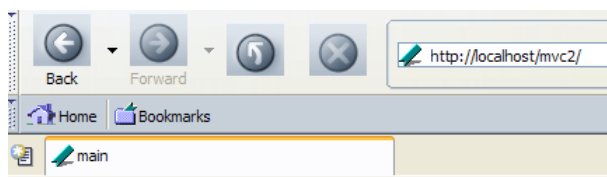
  Protected action As String
  Protected actionPrec As String

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' on récupère l'action en cours
    action = Me.Context.Items("action").ToString
    ' et l'action précédente
    actionPrec = Me.Context.Items("actionPrec").ToString
  End Sub
End Class

```

Le code ci-dessus se contente de récupérer les informations mises dans le contexte par le contrôleur de l'application [default.aspx.vb].

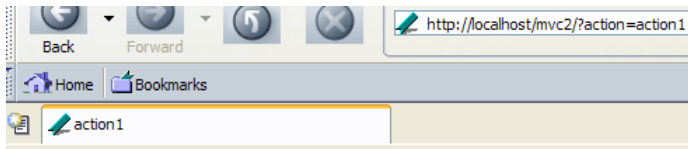
Tous ces fichiers sont placés dans <application-path> et Cassini est lancé avec les paramètres (<application-path>/mvc2). On demande d'abord l'url [http://localhost/mvc2] :



Page [main]

Action courante : main
Action précédente :

L'url [http://localhost/mvc2] désigne un dossier. Nous savons que dans ce cas, c'est le document [default.aspx] de ce dossier qui est renvoyé par le serveur, s'il existe. Ici, aucune action n'était précisée. C'est donc l'action [main] qui s'est exécutée. Passons à l'action [action1] :

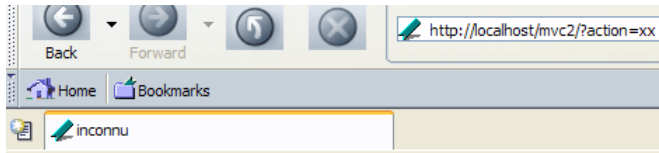


Page [action1]

Action courante : action1

Action précédente : main

L'action courante et l'action précédente ont été correctement identifiées. Passons à une action [xx] :



Page [inconnu]

Action courante : xx

Action précédente : action1

3.4 Conclusion

Nous avons maintenant les éléments de base à partir desquels toute application ASP.NET est construite. Il nous reste cependant une notion importante à introduire : celle de formulaire. C'est l'objet du chapitre qui suit.

4 Gestion de l'interface utilisateur

4.1 Introduction

Dans la relation client-serveur du web, le client transmet des informations au serveur sous la forme de chaîne de paramètres [param1=val1¶m2=val2&...]. Nous avons dans nos exemples précédents, construit cette chaîne le plus souvent à la main en demandant des url de la forme [http://localhost/appli? param1=val1¶m2=val2&...]. Dans la réalité, les informations que le transmet le client au serveur proviennent de formulaires que l'utilisateur a remplis. Nous explorons la construction de ceux-ci dans ce chapitre. Nous introduisons également l'outil WebMatrix qui va nous permettre de dessiner les interfaces utilisateur. L'installation de cet outil est présentée dans les annexes.

4.2 Le langage HTML

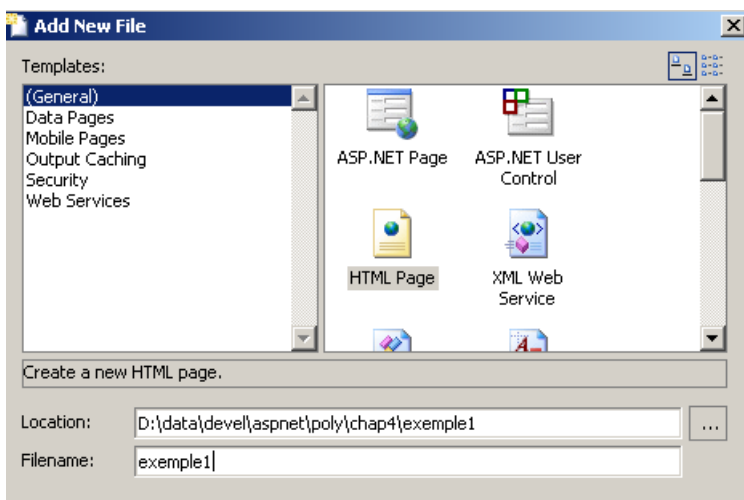
4.2.1 Un exemple

Considérons l'exemple suivant, créé avec [Web Matrix] qui présente :

- un tableau
- une image
- un lien



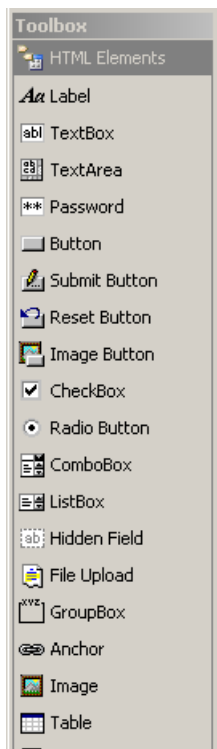
Lançons [WebMatrix] puis prenons l'option [File/New File] :



Nous choisissons de créer une page HTML. Les informations ci-dessus vont créer un fichier [d:\data\devel\aspnet\chap4\exemple1\exemple1.htm]. [WebMatrix] offre deux possibilités d'éditer ce fichier : le mode [Design] et le mode [HTML] :



Le mode [Design] permet d'utiliser la palette de composants HTML proposée par [WebMatrix] :



Pour insérer un élément de cette palette, il suffit de double-cliquer dessus et ensuite de le positionner sur la fenêtre [Design]. Le mode [HTML] permet de construire le document HTML à l'aide d'un éditeur de texte. Il faut pour cela connaître la syntaxe du langage HTML. Dans l'onglet [HTML], un squelette de document a été généré :

```
D:\data\devel\aspnet\poly\chap4\exemple1\exemple1.htm *
<html>
<head>
</head>
<body>
  <!-- Insert content here -->
</body>
</html>
```

La fenêtre [HTML] est très utile à celui qui ne connaît pas le langage HTML. Il construit alors dans la fenêtre [Design] son document et vérifie le code HTML généré dans la fenêtre [HTML]. Il acquiert ainsi progressivement la maîtrise de HTML et peut assez rapidement travailler uniquement avec l'éditeur de texte sans l'aide du mode [Design]. Nous allons montrer maintenant comment construire le document HTML présenté au début de cette section. Nous travaillons dans la fenêtre [Design]. Tout d'abord, nous introduisons directement la première ligne de texte :

```
D:\data\devel\aspnet\poly\chap4\exemple1\exemple1.htm *
Le langage HTML - 1
```

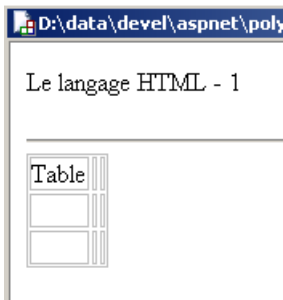
- nous ajoutons le composant [Horizontal Rule] dans la palette des composants :

```
D:\data\devel\aspnet\poly\chap4\exemple1\exemple1.htm *
Le langage HTML - 1

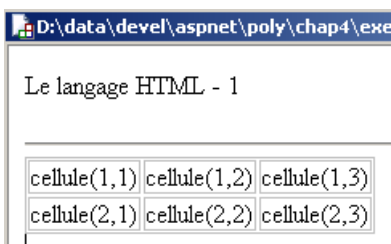

---


```

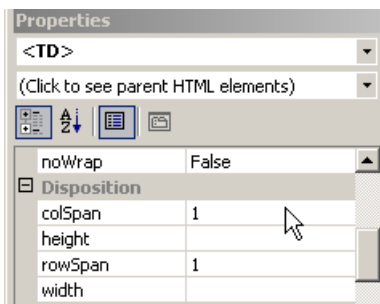
- nous ajoutons le composant [Table] :



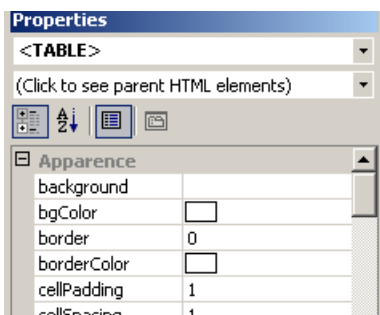
- nous plaçons le curseur dans la troisième ligne de la table pour la supprimer avec l'option [HTML/Edit Table/Delete Table Row]. Puis nous inscrivons le texte attendu dans chaque cellule :



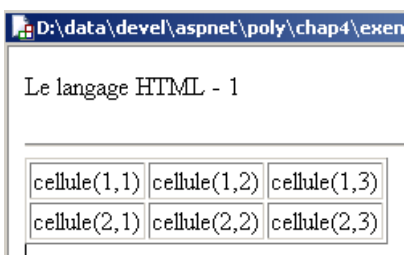
- nous plaçons le curseur dans l'une des cellules du tableau et nous nous intéressons aux propriétés de celle-ci. La fenêtre de propriétés est présentée en bas à droite de l'environnement de travail :



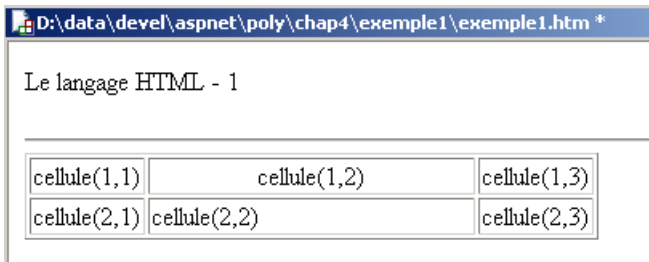
- la cellule est représentée par la balise HTML <TD>. On nous présente donc les propriétés de la balise <TD>. Nous nous intéressons au tableau qui est un objet englobant la cellule. Nous cliquons sur la liste déroulante (click to see parent HTML elements) ci-dessus pour sélectionner l'objet <TABLE> :



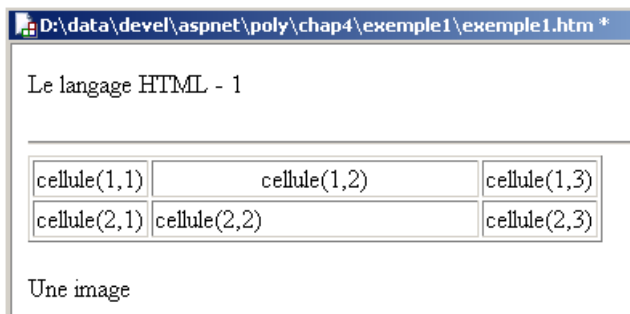
- l'objet <TABLE> a une propriété [border] fixant la largeur de la bordure entourant les cellules du tableau. Ici, nous prenons border=1.



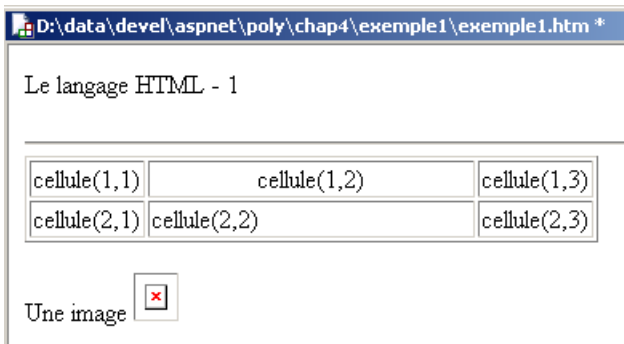
- nous éditons maintenant les attributs de l'objet <TD> de la cellule (1,2) pour fixer align=Center et width=200 (pixels). Le texte sera centré dans la cellule (align=center) et celle-ci sera large de 200 pixels. Pour voir la modification, vous pouvez parfois être obligé de sélectionner l'onglet [HTML] puis revenir sur l'onglet [Design] :



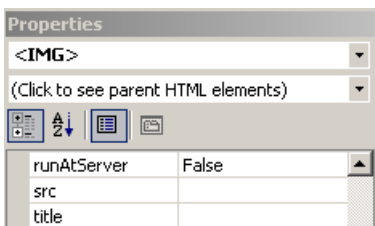
- maintenant nous plaçons le texte qui précède l'image :



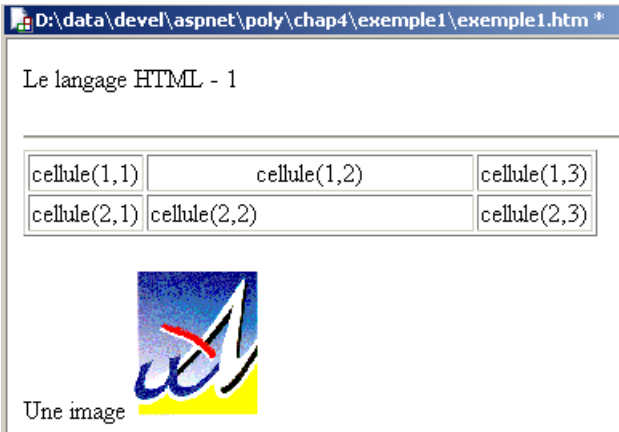
- puis nous plaçons l'image en double-cliquant sur le composant [image] de la palette :



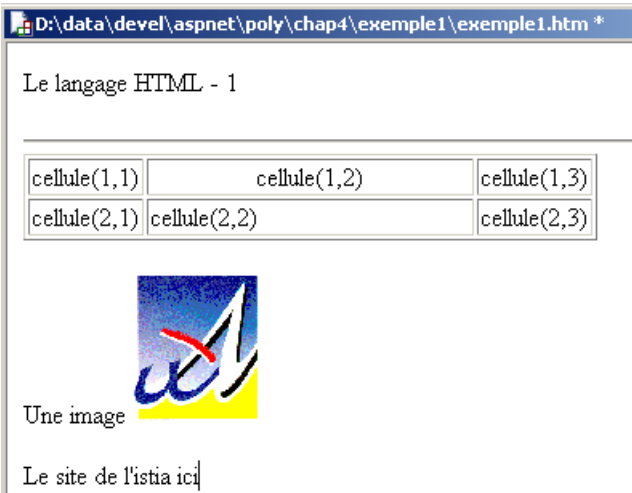
- nous sélectionnons l'image pour éditer ses propriétés :



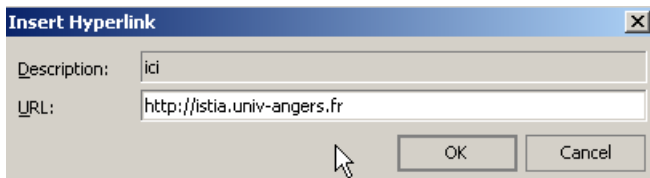
- dans l'attribut [src], nous mettons le nom du fichier contenant l'image, ici [univ01.gif] :



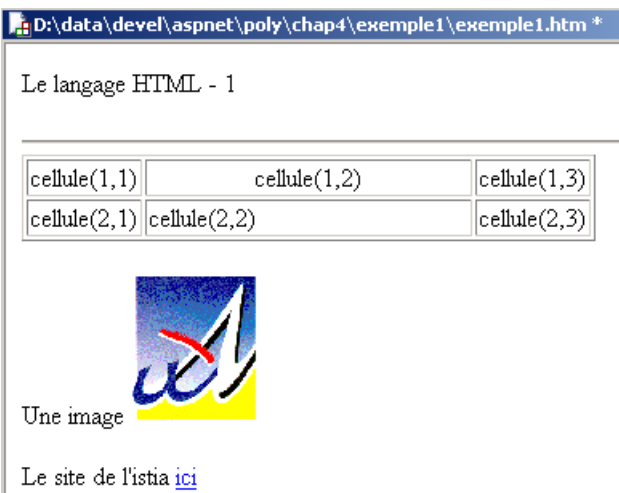
- nous plaçons le texte précédant le lien :



- nous faisons du texte [ici] un lien vers l'url [http://istia.univ-angers.fr]. Pour cela, nous le sélectionnons puis prenons l'option [HTML/Insert Hyperlink] :



- nous obtenons le résultat suivant :



- nous y sommes presque. Regardons le code HTML généré dans l'onglet [HTML] :

```

<html>
<head>
</head>
<body>
  <p>
    Le langage HTML - 1
  </p>
  <hr />
  <table border="1">
    <tbody>
      <tr>
        <td>
          cellule(1,1)</td>
        <td align="middle" width="200">
          cellule(1,2)</td>
        <td>
          cellule(1,3)</td>
      </tr>
      <tr>
        <td>
          cellule(2,1)</td>
        <td>
          cellule(2,2)</td>
        <td>
          cellule(2,3)</td>
      </tr>
    </tbody>
  </table>
  <p>
    Une image 
  </p>
  <p>
    Le site de l'istia <a href="http://istia.univ-angers.fr">ici</a>
  </p>
</body>
</html>

```

Il nous reste quelques détails à régler. Tout d'abord nous aimerions donner un titre à notre document. [WebMatrix] ne permet pas de le faire en mode [Design]. Dans l'onglet [HTML], nous remplaçons la séquence <head>..</head>, par la séquence suivante :

```

<head>
  <title>HTML1</title>
</head>

```

Par ailleurs, nous souhaiterions avoir le texte [Le langage HTML - 1] en plus grands caractères. La séquence <Hi>texte</Hi> permet de fixer la taille d'un texte avec i variant de 1 à 6 du plus grand au plus petit. Ici, nous prendrons H2. La séquence

```

  <p>
    Le langage HTML - 1
  </p>

```

devient :

```

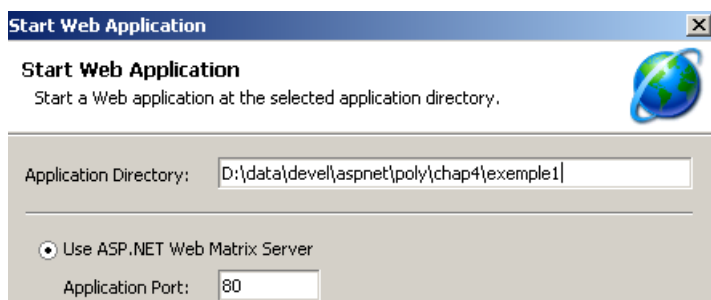
  <h2>Le langage HTML - 1</h2>

```

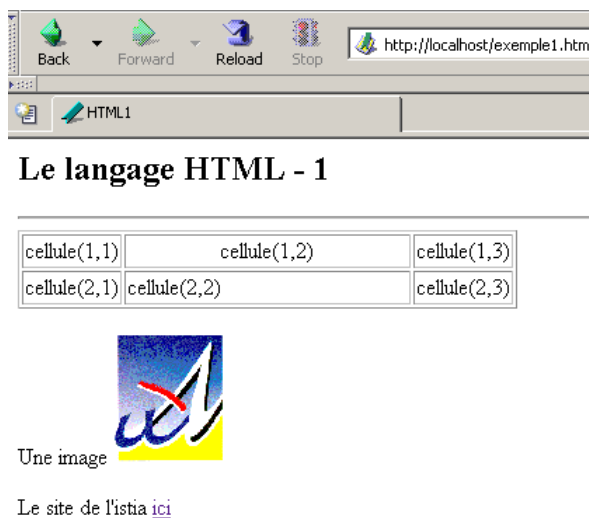
La fenêtre [Design] reflète nos changements :



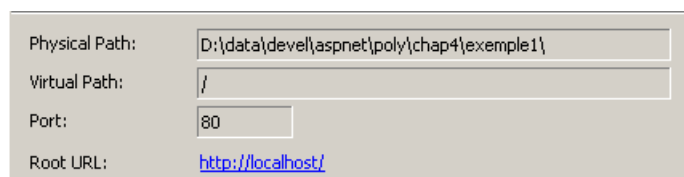
Il ne nous reste plus qu'à tester à l'aide de l'option [View/Start] ou [F5]. [WebMatrix] demande quelques renseignements pour lancer le serveur Web [Cassini] :



Nous pouvons accepter les valeurs proposées par défaut. Le serveur [Cassini] est lancé et notre page affichée dans un navigateur :



Par curiosité, nous pouvons vérifier avec quels paramètres a été lancé [Cassini] :



Nous avons présenté les bases de la construction d'une page HTML avec [WebMatrix]. Le lecteur est invité à construire d'autres pages HTML et à vérifier à chaque fois le code HTML généré. Peu à peu, il sera capable de construire une page sans l'aide du mode [Design]. Un document HTML a la forme générale suivante :

| <html>


```

<head>
  <title>Un titre</title>
  ...
</head>
<body attributs>
  ...
</body>
</html>

```

L'ensemble du document est encadré par les balises **<html>...</html>**. Il est formé de deux parties :

<head>...</head> : c'est la partie non affichable du document. Elle donne des renseignements au navigateur qui va afficher le document. On y trouve souvent la balise **<title>...</title>** qui fixe le texte qui sera affiché dans la barre de titre du navigateur. On peut y trouver d'autres balises notamment des balises définissant les mots clés du document, mots clés utilisés ensuite par les moteurs de recherche. On peut trouver également dans cette partie des scripts, écrits le plus souvent en javascript ou vbscript et qui seront exécutés par le navigateur.

<body attributs>...</body> : c'est la partie qui sera affichée par le navigateur. Les balises HTML contenues dans cette partie indiquent au navigateur la forme visuelle "souhaitée" pour le document. Chaque navigateur va interpréter ces balises à sa façon. Deux navigateurs peuvent alors visualiser différemment un même document web. C'est généralement l'un des casse-têtes des concepteurs web.

Le code HTML du document précédent était le suivant :

```

<html>
<head>
  <title>HTML1</title>
</head>
<body>
  <h2>Le langage HTML - 1
  </h2>
  <hr />
  <table border="1">
    <tbody>
      <tr>
        <td>
          cellule(1,1)</td>
        <td align="middle" width="200">
          cellule(1,2)</td>
        <td>
          cellule(1,3)</td>
      </tr>
      <tr>
        <td>
          cellule(2,1)</td>
        <td>
          cellule(2,2)</td>
        <td>
          cellule(2,3)</td>
      </tr>
    </tbody>
  </table>
  <p>
    Une image 
  </p>
  <p>
    Le site de l'istia <a href="http://istia.univ-angers.fr">ici</a>
  </p>
</body>
</html>

```

Elément	balises et exemples HTML
titre du document	<title>HTML1</title> HTML1 apparaîtra dans la barre de titre du navigateur qui affichera le document
barre horizontale	
 : affiche un trait horizontal
tableau	<table attributs>...</table> : pour définir le tableau <tr attributs>...</tr> : pour définir une ligne <td attributs>...</td> : pour définir une cellule
exemples :	
	<table border="1">...</table> : l'attribut border définit l'épaisseur de la bordure du tableau
	<td align="center" width="200">cellule(1,2)</td> : définit une cellule dont le contenu sera <i>cellule(1,2)</i> . Ce contenu

sera centré horizontalement (`align="center"`). La cellule aura une largeur de 200 pixels (`width="200"`)

image `` : définit une image dont le fichier source est `univ01.gif` sur le serveur web (`src="univ01.gif"`). Ce lien se trouve sur un document web obtenu avec l'URL `http://localhost/exemple1.htm`. Aussi, le navigateur demandera-t-il l'URL `http://localhost/univ01.gif` pour avoir l'image référencée ici.

lien `ici` : fait que le texte `ici` sert de lien vers l'URL `http://istia.univ-angers.fr`.

On voit dans ce simple exemple que pour construire l'intégralité du document, le navigateur doit faire deux requêtes au serveur :

`http://localhost/exemple1.htm` pour avoir le source HTML du document

`http://localhost/univ01.gif` pour avoir l'image `univ01.gif`

4.2.2 Construction d'un formulaire

Un formulaire HTML a pour but de présenter à l'utilisateur une page de saisie d'informations analogue aux formulaires de saisie rencontrés sous windows. Le formulaire de saisie est envoyé sous la forme d'un document HTML au navigateur. Celui-ci va présenter le formulaire à l'utilisateur qui va le remplir et le valider avec un bouton ayant cette fonction. Le navigateur va alors transmettre les valeurs saisies au serveur pour traitement. On verra qu'il ne renvoie pas au serveur la totalité du formulaire mais seulement les valeurs saisies. L'exemple suivant présente un formulaire Web créé lui aussi avec WebMatrix :

Gestion d'un formulaire

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="*****"/>
Boîte de saisie	<input type="text" value="ligne1"/> <input type="text" value="ligne2"/>
ComboBox	<input type="text" value="choix2"/> ▼
Liste à choix simple	liste1 ▲ liste2 ▢ liste3 ▼
Liste à choix multiple	multiple1 ▲ multiple2 ▢ multiple3 ▼
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

Le code HTML généré par [WebMatrix] est le suivant :

```
<html>
<head>
  <title>Formulaire</title>
  <script language="javascript">
    function effacer(){
      alert("Vous avez cliqué sur le bouton [Effacer]");
    }
  </script>
</head>
<body>
  <p>
    Gestion d'un formulaire
  </p>
  <hr />
  <form name="formulaire" method="post">
    <table border="1">
      <tr>
        <td>
          Etes-vous marié(e)</td>
        <td>
          <p align="center">
```

```

        <input type="radio" value="oui" name="rdMarie" />Oui
        <input type="radio" checked value="non" name="rdMarie" />Non
    </p>
</td>
</tr>
<tr>
    <td>
        Cases à cocher
    </td>
    <td>
        <p align="center">
            <input type="checkbox" value="un" name="C1" />1
            <input type="checkbox" checked value="deux" name="C2" />2
            <input type="checkbox" value="trois" name="C3" />3
        </p>
    </td>
</tr>
<tr>
    <td>
        Champ de saisie</td>
    <td>
        <p align="center">
            <input type="text" maxlength="30" value="qqs mots" name="txtSaisie" />
        </p>
    </td>
</tr>
<tr>
    <td>
        Mot de passe</td>
    <td>
        <p align="center">
            <input type="password" maxlength="12" size="12" value="unMotDePasse" name="txtMdp" />
        </p>
    </td>
</tr>
<tr>
    <td>
        Boîte de saisie</td>
    <td>
        <p align="center">
            <textarea name="areaSaisie">ligne1
ligne2</textarea>
        </td>
</tr>
<tr>
    <td>
        ComboBox</td>
    <td>
        <p align="center">
            <select name="cmbValeurs">
                <option value="1">choix1</option>
                <option value="2" selected>choix2</option>
                <option value="3">choix3</option>
            </select>
        </td>
</tr>
<tr>
    <td>
        Liste à choix simple</td>
    <td>
        <p align="center">
            <select size="3" name="lstSimple">
                <option value="1" selected>liste1</option>
                <option value="2">liste2</option>
                <option value="3">liste3</option>
                <option value="4">liste4</option>
                <option value="5">liste5</option>
            </select>
        </td>
</tr>
<tr>
    <td>
        Liste à choix multiple</td>
    <td>
        <p align="center">
            <select multiple size="3" name="lstMultiple">
                <option value="1" selected>multiple1</option>

```

```

        <option value="2">multiple2</option>
        <option value="3" selected>multiple3</option>
        <option value="4">multiple4</option>
        <option value="5">multiple5</option>
    </select>
</td>
</tr>
<tr>
<td>
    Bouton simple</td>
<td>
    <p align="center">
        <input onclick="effacer()" type="button" value="Effacer" name="btnEffacer" />
    </p>
</td>
</tr>
<tr>
<td>
    Bouton submit</td>
<td>
    <p align="center">
        <input type="submit" value="Envoyer" name="btnEnvoyer" />
    </p>
</td>
</tr>
<tr>
<td>
    Bouton reset</td>
<td>
    <p align="center">
        <input type="reset" value="Rétablir" name="btnRetablir" runat="server" />
    </p>
</td>
</tr>
</table>
<input type="hidden" name="secret" value="uneValeur" />
</form>
</body>
</html>

```

L'association contrôle visuel <--> balise HTML est le suivant :

Contrôle	balise HTML
formulaire	<form name="formulaire" method="post">
champ de saisie	<input type="text" maxlength="30" value="qqs mots" name="txtSaisie" />
champ de saisie cachée	<input type="password" maxlength="12" size="12" value="unMotDePasse" name="txtMdp" />
champ de saisie multilignes	<textarea name="areaSaisie">ligne1 ligne2</textarea>
boutons radio	<input type="radio" value="oui" name="rdMarie" />Oui <input type="radio" checked value="non" name="rdMarie" />Non
cases à cocher	<input type="checkbox" value="un" name="C1" />1 <input type="checkbox" checked value="deux" name="C2" />2 <input type="checkbox" value="trois" name="C3" />3
Combo	<select name="cmbValeurs"> <option value="1">choix1</option> <option value="2" selected>choix2</option> <option value="3">choix3</option> </select>
liste à sélection unique	<select size="3" name="lstSimple"> <option value="1" selected>liste1</option> <option value="2">liste2</option> <option value="3">liste3</option> <option value="4">liste4</option> <option value="5">liste5</option> </select>
liste à sélection multiple	<select multiple size="3" name="lstMultiple"> <option value="1" selected>multiple1</option> <option value="2">multiple2</option> <option value="3" selected>multiple3</option> <option value="4">multiple4</option> <option value="5">multiple5</option> </select>
champ caché	<input type="hidden" name="secret" value="uneValeur" />
bouton de type submit	<input type="submit" value="Envoyer" name="btnEnvoyer" />

bouton de type reset		<code><input type="reset" value="Rétablir" name="btnRetablir" runat="server" /></code>	
bouton de type button		<code><input onclick="effacer()" type="button" value="Effacer" name="btnEffacer" /></code>	

Passons en revue ces différents contrôles.

4.2.2.1 Le formulaire

formulaire		<code><form name="formulaire" method="post"></code>	
------------	--	---	--

balise HTML `<form name="..." method="..." action="...">...</form>`

attributs

- name="frmexemple"** : nom du formulaire
- method="..."** : méthode utilisée par le navigateur pour envoyer au serveur web les valeurs récoltées dans le formulaire
- action="..."** : URL à laquelle seront envoyées les valeurs récoltées dans le formulaire.

Un formulaire web est entouré des balises `<form>...</form>`. Le formulaire peut avoir un nom (*name="xx"*). C'est le cas pour tous les contrôles qu'on peut trouver dans un formulaire. Ce nom est utile si le document web contient des scripts qui doivent référencer des éléments du formulaire. Le but d'un formulaire est de rassembler des informations données par l'utilisateur au clavier/souris et d'envoyer celles-ci à une URL de serveur web. Laquelle ? Celle référencée dans l'attribut *action="URL"*. Si cet attribut est absent, les informations seront envoyées à l'URL du document dans lequel se trouve le formulaire. Comment un client web fait-il pour donner des informations (celles contenues dans le formulaire) à un serveur web ? Nous l'avons vu en détail. Il peut utiliser deux méthodes différentes appelées POST et GET. L'attribut *method="méthode"*, avec méthode égale à GET ou POST, de la balise `<form>` indique au navigateur la méthode à utiliser pour envoyer les informations recueillies dans le formulaire à l'URL précisée par l'attribut *action="URL"*. Lorsque l'attribut *method* n'est pas précisé, c'est la méthode GET qui est prise par défaut.

4.2.2.2 Champ de saisie

champ de saisie		<code><input type="text" maxlength="30" value="qqs mots" name="txtSaisie" /></code> <code><input type="password" maxlength="12" size="12" value="unMotDePasse" name="txtMdp" /></code>	
-----------------	--	---	--

Champ de saisie	<input type="text" value="qqs mots"/>
Mot de passe	<input type="password" value="*****"/>

balise HTML `<input type="..." name="..." size=".." value="..">`

La balise input existe pour divers contrôles. C'est l'attribut *type* qui permet de différencier ces différents contrôles entre eux.

attributs

- type="text"** : précise que c'est un champ de saisie
- type="password"** : les caractères présents dans le champ de saisie sont remplacés par des caractères *. C'est la seule différence avec le champ de saisie normal. Ce type de contrôle convient pour la saisie des mots de passe.
- size="12"** : nombre de caractères visibles dans le champ - n'empêche pas la saisie de davantage de caractères
- maxlength="30"** : fixe à 30 le nombre maximal de caractères - le navigateur est chargé de faire respecter cet attribut
- name="txtSaisie"** : nom du contrôle
- value="qqs mots"** : texte qui sera affiché dans le champ de saisie.

4.2.2.3 Champ de saisie multilignes

champ de saisie multilignes		<code><textarea name="areaSaisie">ligne1 ligne2</textarea></code>	
-----------------------------	--	---	--

Boîte de saisie	<pre>ligne1 ligne2</pre>
-----------------	--------------------------

balise HTML `<textarea ...>texte</textarea>`

affiche une zone de saisie multilignes avec au départ texte dedans

attributs **rows="2"** : nombre de lignes

`cols="20"` : nombre de colonnes
`name="areaSaisie"` : nom du contrôle

4.2.2.4 Boutons radio

```
boutons radio | <input type="radio" value="oui" name="rdMarie" />Oui  
               | <input type="radio" checked value="non" name="rdMarie" />Non
```

Etes-vous marié(e)	<input type="radio"/> Oui <input checked="" type="radio"/> Non
--------------------	--

balise HTML `<input type="radio" attribut2="valeur2"/>texte`
affiche un bouton radio avec **texte** à côté.

attributs **name="rdMarie"** : nom du contrôle. Les boutons radio portant le même nom forment un groupe de boutons exclusifs les uns des autres : on ne peut cocher que l'un d'eux.
value="valeur" : valeur affectée au bouton radio. Il ne faut pas confondre cette valeur avec le texte affiché à côté du bouton radio. Celui-ci n'est destiné qu'à l'affichage.
checked : si ce mot clé est présent, le bouton radio est coché, sinon il ne l'est pas.

4.2.2.5 Cases à cocher

```
cases à cocher | <input type="checkbox" value="un" name="C1" />1  
               | <input type="checkbox" checked value="deux" name="C2" />2  
               | <input type="checkbox" value="trois" name="C3" />3
```

Cases à cocher	<input type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
----------------	---

balise HTML `<input type="checkbox" attribut2="valeur2"/>texte`
affiche une case à cocher avec texte à côté.

attributs **name="C1"** : nom du contrôle. Les cases à cocher peuvent porter ou non le même nom. Les cases portant le même nom forment un groupe de cases associées.
value="valeur" : valeur affectée à la case à cocher. Il ne faut pas confondre cette valeur avec le texte affiché à côté de la case à cocher. Celui-ci n'est destiné qu'à l'affichage.
checked : si ce mot clé est présent, le bouton radio est coché, sinon il ne l'est pas.

4.2.2.6 Liste déroulante (combo)

```
Combo | <select name="cmbValeurs">  
       | <option value="1">choix1</option>  
       | <option value="2" selected>choix2</option>  
       | <option value="3">choix3</option>  
       | </select>
```

ComboBox	choix2 ▼
----------	----------

balise HTML `<select size=".." name="..">
 <option [selected="selected"] [value="valeur"]>texte</option>
 ...
</select>`

affiche dans une liste les textes compris entre les balises `<option>...</option>`

attributs **name="cmbValeurs"** : nom du contrôle.
size="1" : nombre d'éléments de liste visibles. `size="1"` fait de la liste l'équivalent d'un combobox. C'est la valeur par défaut si l'attribut `[size]` est absent.
selected="selected" : si ce mot clé est présent pour un élément de liste, ce dernier apparaît sélectionné dans la liste. Dans notre exemple ci-dessus, l'élément de liste `choix2` apparaît comme l'élément sélectionné du combo lorsque celui-ci est affiché pour la première fois.
value="valeur" : fixe la valeur à envoyer au serveur si l'élément est sélectionné. Si cet attribut est absent, c'est alors le texte associé à l'option qui est envoyé au serveur.

4.2.2.7 Liste à sélection unique

liste à sélection unique

```
<select size="3" name="lstSimple">
  <option value="1" selected>liste1</option>
  <option value="2">liste2</option>
  <option value="3">liste3</option>
  <option value="4">liste4</option>
  <option value="5">liste5</option>
</select>
```



balise HTML

```
<select size=".." name="..">
  <option [selected] [value="valeur"]>...</option>
  ...
</select>
```

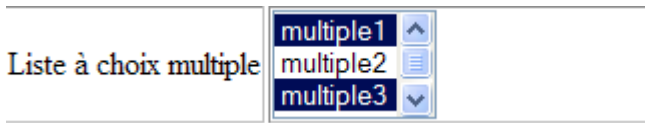
affiche dans une liste les textes compris entre les balises `<option>...</option>`

attributs les mêmes que pour la liste déroulante n'affichant qu'un élément. Ce contrôle ne diffère de la liste déroulante précédente que par son attribut `size>1`.

4.2.2.8 Liste à sélection multiple

liste à sélection multiple

```
<select multiple size="3" name="lstMultiple">
  <option value="1" selected>multiple1</option>
  <option value="2">multiple2</option>
  <option value="3" selected>multiple3</option>
  <option value="4">multiple4</option>
  <option value="5">multiple5</option>
</select>
```



balise HTML

```
<select size=".." name=".." multiple>
  <option [selected] [value="valeur"]>...</option>
  ...
</select>
```

affiche dans une liste les textes compris entre les balises `<option>...</option>`

attributs **multiple** : permet la sélection de plusieurs éléments dans la liste. Dans l'exemple ci-dessus, les éléments *liste1* et *liste3* sont tous deux sélectionnés.

4.2.2.9 Bouton de type button

bouton de type button

```
<input onclick="effacer()" type="button" value="Effacer" name="btnEffacer" />
```



balise HTML

```
<input type="button" value="..." name="..." onclick="effacer()" ...>
```

attributs **type="button"** : définit un contrôle bouton. Il existe deux autres types de bouton, les types *submit* et *reset*.

value="Effacer" : le texte affiché sur le bouton

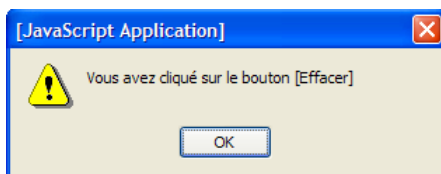
onclick="fonction()" : permet de définir une fonction à exécuter lorsque l'utilisateur clique sur le bouton. Cette fonction fait partie des scripts définis dans le document web affiché. La syntaxe précédente est une syntaxe *javascript*. Si les scripts sont écrits en *vbscript*, il faudrait écrire **onclick="fonction"** sans les parenthèses. La syntaxe devient identique s'il faut passer des paramètres à la fonction : **onclick="fonction(val1, val2,...)"**

Dans notre exemple, un clic sur le bouton *Effacer* appelle la fonction javascript *effacer* suivante :

```
<script language="javascript">
```

```
function effacer(){
    alert("Vous avez cliqué sur le bouton [Effacer]");
}
</script>
```

La fonction *effacer* affiche un message :



4.2.2.10 Bouton de type submit

bouton de type submit | `<input type="submit" value="Envoyer" name="btnEnvoyer" />`



balise HTML `<input type="submit" value="Envoyer" name="btnEnvoyer">`

attributs **type="submit"** : définit le bouton comme un bouton d'envoi des données du formulaire au serveur web. Lorsque le client va cliquer sur ce bouton, le navigateur va envoyer les données du formulaire à l'URL définie dans l'attribut **action** de la balise `<form>` selon la méthode définie par l'attribut **method** de cette même balise.
value="Envoyer" : le texte affiché sur le bouton

4.2.2.11 Bouton de type reset

bouton de type reset | `<input type="reset" value="Rétablir" name="btnRetablir" runat="server" />`



balise HTML `<input type="reset" value="Rétablir" name=" btnRetablir ">`

attributs **type="reset"** : définit le bouton comme un bouton de réinitialisation du formulaire. Lorsque le client va cliquer sur ce bouton, le navigateur va remettre le formulaire dans l'état où il l'a reçu.
value="Rétablir" : le texte affiché sur le bouton

4.2.2.12 Champ caché

champ caché | `<input type="hidden" name="secret" value="uneValeur" />`

balise HTML `<input type="hidden" name="..." value="...">`

attributs **type="hidden"** : précise que c'est un champ caché. Un champ caché fait partie du formulaire mais n'est pas présenté à l'utilisateur. Cependant, si celui-ci demandait à son navigateur l'affichage du code source, il verrait la présence de la balise `<input type="hidden" value="...">` et donc la valeur du champ caché.

value="uneValeur" : valeur du champ caché.

Quel est l'intérêt du champ caché ? Cela peut permettre au serveur web de garder des informations au fil des requêtes d'un client. Considérons une application d'achats sur le web. Le client achète un premier article *art1* en quantité *q1* sur une première page d'un catalogue puis passe à une nouvelle page du catalogue. Pour se souvenir que le client a acheté *q1* articles *art1*, le serveur peut mettre ces deux informations dans un champ caché du formulaire web de la nouvelle page. Sur cette nouvelle page, le client achète *q2* articles *art2*. Lorsque les données de ce second formulaire vont être envoyées au serveur (submit), celui-ci va non seulement recevoir l'information (*q2,art2*) mais aussi (*q1,art1*) qui fait partie également partie du formulaire en tant que champ caché non modifiable par l'utilisateur. Le serveur web va alors mettre dans un nouveau champ caché les informations (*q1,art1*) et (*q2,art2*)

et envoyer une nouvelle page de catalogue. Et ainsi de suite.

4.3 Envoi à un serveur web par un navigateur des valeurs d'un formulaire

Nous savons déjà, grâce au chapitre précédent, comment un client transmet des informations au serveur Il le fait soit par :

- une requête HTTP GET url?param1=val1¶m2=val2&....
- une requête HTTP POST url suivie d'un document contenant la chaîne de paramètres param1=val1¶m2=val2&....

Le navigateur va utiliser l'une de ces deux méthodes selon que l'attribut [method] de la balise [form] est GET ou POST. C'est ce que nous allons montrer maintenant. La page étudiée précédemment est une page statique. Afin d'avoir accès aux entêtes HTTP envoyés par le navigateur qui va demander ce document, nous la transformons en page dynamique pour un serveur web .NET (IIS ou Cassini). Le code statique précédent est placé dans un fichier [formulaire_get.aspx] et son contenu est le suivant :

```
<%@ Page src="formulaire_get.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="formulaire_get" %>
<html>
<head>
  <title>Formulaire</title>
  <script language="javascript">
    function effacer(){
      alert("Vous avez cliqué sur le bouton [Effacer]");
    }
  </script>
</head>
<body>
.....
</body>
</html>
```

La page de présentation ci-dessus est associée au contrôleur [formulaire_get.aspx.vb] :

```
Public Class formulaire_get
  Inherits System.Web.UI.Page

  Private Sub Page_Init(ByVal Sender As Object, ByVal e As System.EventArgs) Handles MyBase.Init
    ' sauve la requête courante dans request.txt du dossier de la page
    Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
    Me.Request.SaveAs(requestFileName, True)
  End Sub
End Class
```

A chaque appel du document [formulaire_get.aspx], la requête du client sera sauvegardée dans un fichier [request.txt] par la procédure Page_Init. Nous avons déjà rencontré ce mode de fonctionnement, aussi ne commenterons-nous pas plus le contrôleur.

4.3.1 Méthode GET

Faisons un premier test, où dans le code HTML du document, la balise FORM est définie de la façon suivante :

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

Nous plaçons les fichiers [formulaire_get.aspx] et [formulaire_get.aspx.vb] dans un dossier <application-path> et nous lançons le serveur Cassini avec les paramètres (<application-path>, /form2). Nous demandons l'url http://localhost/form2/formulaire_get.aspx :



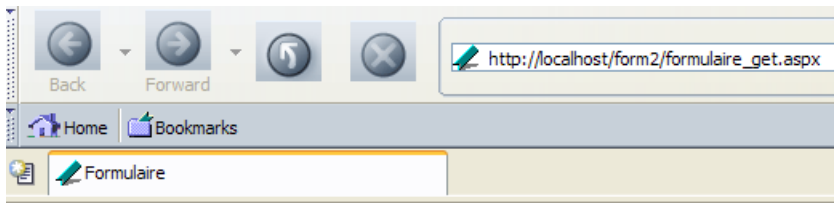
Gestion d'un formulaire

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="*****"/>
Boîte de saisie	<input type="text" value="ligne1"/> <input type="text" value="ligne2"/>
ComboBox	<input type="text" value="choix2"/>
Liste à choix simple	<input type="text" value="liste1"/> <input type="text" value="liste2"/> <input type="text" value="liste3"/>
Liste à choix multiple	<input type="text" value="multiple1"/> <input type="text" value="multiple2"/> <input type="text" value="multiple3"/>
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

Le navigateur vient de faire une requête et nous savons que celle-ci a été enregistrée dans le fichier [request.txt]. Regardons le contenu de celui-ci :

```
GET /form2/formulaire_get.aspx HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-flash, text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png, image/jpeg, image/gif;q=0.2, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.7, *;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-us, en;q=0.5
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

Le navigateur a utilisé une requête HTTP GET pour demander l'url [http://localhost/form2/formulaire_get.aspx]. C'est toujours le cas lorsque l'url est donnée par l'utilisateur. Nous remplissons le formulaire est rempli de la façon suivante :



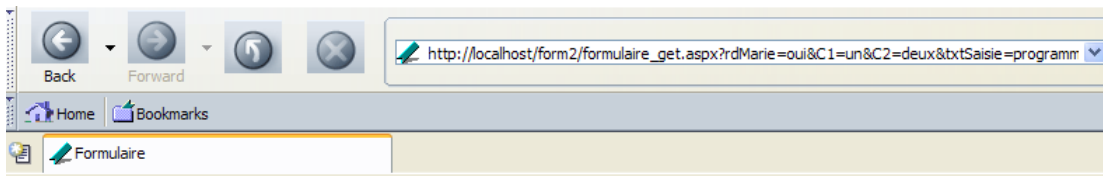
Gestion d'un formulaire

Etes-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	<input type="text" value="programmation asp.net"/>
Mot de passe	<input type="password" value="*****"/>
Boîte de saisie	<input type="text" value="les bases de la programmation web"/>
ComboBox	<input type="text" value="choix3"/>
Liste à choix simple	<input type="list" value="liste1"/> <input type="list" value="liste2"/> <input type="list" value="liste3"/>
Liste à choix multiple	<input type="list" value="multiple2"/> <input type="list" value="multiple3"/> <input type="list" value="multiple4"/>
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

Nous utilisons le bouton [Envoyer] ci-dessus. Son code HTML est le suivant :

```
<form method="get">  
...  
<input type="submit" value="Envoyer">  
...  
</form>
```

Sur l'activation d'un bouton de type [Submit], le navigateur envoie les paramètres du formulaire (balise <form>) à l'URL indiquée dans l'attribut [action] de la balise <form action="URL"> s'il existe. Si cet attribut n'existe pas, les paramètres du formulaire sont envoyés à l'URL qui a délivré le formulaire. C'est le cas ici. Le bouton [Envoyer] devrait donc entraîner une requête du navigateur à l'URL [http://localhost/form2/formulaire_get.aspx] avec un transfert des paramètres du formulaire. Comme la page [formulaire_get.aspx] mémorise la requête reçue, nous devrions savoir comment le client a transféré ces paramètres. Essayons. Nous cliquons sur le bouton [Envoyer]. Nous recevons la réponse suivante du navigateur :



Gestion d'un formulaire

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	qqs mots
Mot de passe	*****
Boîte de saisie	ligne1 ligne2
ComboBox	choix2
Liste à choix simple	liste1 liste2 liste3
Liste à choix multiple	multiple1 multiple2 multiple3
Bouton simple	Effacer
Bouton submit	Envoyer
Bouton reset	Rétablir

C'est la page initiale mais on peut remarquer que l'URL a changé dans le champ [Adresse] du navigateur. Elle est devenue la suivante :

```
http://localhost/form2/formulaire_get.aspx?rdMarie=oui&C1=un&C2=deux&txtSaisie=programmation+asp.net&txtMdp=unMotDePasse&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web%0D%0A&cmbValeurs=3&lstSimple=1&lstMultiple=2&lstMultiple=4&btnEnvoyer=Envoyer&secret=uneValeur
```

On constate que les choix faits dans le formulaire se retrouvent dans l'URL. Regardons le contenu du fichier [request.txt] qui a mémorisé la requête du client :

```
GET
/form2/formulaire_get.aspx?rdMarie=oui&C1=un&C2=deux&txtSaisie=programmation+asp.net&txtMdp=ceciestsecre
&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web%0D%0A&cmbValeurs=3&lstSimple=1&lstMultiple=2&lstMulti
ple=4&btnEnvoyer=Envoyer&secret=uneValeur HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp
eg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/form2/formulaire_get.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

On retrouve une requête HTTP assez semblable à celle qui avait été faite initialement par le navigateur lorsqu'il avait demandé le document sans transmettre de paramètres. Il y a deux différences :

GET URL HTTP/1.1 les paramètres du formulaire ont été ajoutés derrière l'URL du document sous la forme ? param1=val1¶m2=val2&...

Referer le client indique par cet entête HTTP l'URL du document qu'il affichait lorsqu'il a fait la requête

Examinons de plus près comment les paramètres ont été passés dans la commande *GET URL?param1=valeur1¶m2=valeur2&... HTTP/1.1* où les *parami* sont les noms des contrôles du formulaire web et *valeuri* les valeurs qui leur sont associées. Nous présentons ci-dessous un tableau à trois colonnes :

- colonne 1 : reprend la définition d'un contrôle HTML de l'exemple
- colonne 2 : donne l'affichage de ce contrôle dans un navigateur

- colonne 3 : donne la valeur envoyée au serveur par le navigateur pour le contrôle de la colonne 1 sous la forme qu'elle a dans la requête GET de l'exemple

contrôle HTML	visuel avant validation	valeur(s) renvoyée(s)
<pre><input type="radio" value="oui" name="rdMarie" />Oui <input type="radio" checked value="non" name="rdMarie" />Non</pre>		rdMarie=oui - la valeur de l'attribut <i>value</i> du bouton radio coché par l'utilisateur.
<pre><input type="checkbox" value="un" name="C1" />1 <input type="checkbox" checked value="deux" name="C2" />2 <input type="checkbox" value="trois" name="C3" />3</pre>		C1=un C2=deux - valeurs des attributs <i>value</i> des cases cochées par l'utilisateur
<pre><input type="text" maxlength="30" value="qqs mots" name="txtSaisie" /></pre>		txtSaisie=programmation+asp.net - texte tapé par l'utilisateur dans le champ de saisie. Les espaces ont été remplacés par le signe +
<pre><input type="password" maxlength="12" size="12" value="unMotDePasse" name="txtMdp" /></pre>		txtMdp=ceciestsecre - texte tapé par l'utilisateur dans le champ de saisie. Le texte réellement tapé était "ceciestsecre!". Le dernier caractère a été perdu parce que l'attribut <i>maxlength="12"</i> limitait le nombre de caractères à 12.
<pre><textarea name="areaSaisie">ligne1 ligne2</textarea></pre>		areaSaisie=les+bases+de+la%0D%0A programmation+web - texte tapé par l'utilisateur dans le champ de saisie. %OD%OA est la marque de fin de ligne. Les espaces ont été remplacés par le signe +
<pre><select name="cmbValeurs"> <option value="1">choix1</option> <option value="2" selected>choix2</option> <option value="3">choix3</option> </select></pre>		cmbValeurs=3 - valeur (attribut <i>value</i> de la balise <option>) choisie par l'utilisateur dans la liste à sélection unique
<pre><select size="3" name="lstSimple"> <option value="1" selected>liste1</option> <option value="2">liste2</option> <option value="3">liste3</option> <option value="4">liste4</option> <option value="5">liste5</option> </select></pre>		lstSimple=3 - valeur (attribut <i>value</i> de la balise <option>) choisie par l'utilisateur dans la liste à sélection unique
<pre><select multiple size="3" name="lstMultiple"> <option value="1" selected>multiple1</option> <option value="2">multiple2</option> <option value="3" selected>multiple3</option> <option value="4">multiple4</option> </select></pre>		lstMultiple=2 lstMultiple=4 - valeurs (attribut <i>value</i> de la balise <option>) choisies par l'utilisateur dans la liste à sélection multiple

```
value="4">multiple4</option>
<option
value="5">multiple5</option>
</select>
```

```
<input type="submit"
value="Envoyer"
name="btnEnvoyer" />
```

Bouton submit

Envoyer

btnEnvoyer=Envoyer

- nom et attribut *value* du bouton qui a servi à envoyer les données du formulaire au serveur

```
<input type="hidden"
name="secret"
value="uneValeur" />
```

secret=uneValeur

- attribut *value* du champ caché

On peut se demander ce que le serveur a fait des paramètres qu'on lui a passés. En réalité rien. A la réception de la commande

```
GET
/form2/formulaire_get.aspx?rdMarie=oui&C1=un&C2=deux&txtSaisie=programmation+asp.net&txtMdp=ceciestsecre
&areaSaisie=les+bases+de+la%0D%0Aprogrammation+web%0D%0A&cmbValeurs=3&lstSimple=1&lstMultiple=2&lstMulti
ple=4&btnEnvoyer=Envoyer&secret=uneValeur HTTP/1.1
```

le serveur web a transmis les paramètres à l'URL au document [http://localhost/form2/formulaire_get.aspx], c.a.d. au document que nous avons construit initialement. Nous n'avons écrit aucun code pour récupérer et traiter les paramètres que le client nous envoie. Aussi tout se passe comme si la requête du client était simplement :

```
GET http://localhost/form2/formulaire_get.aspx
```

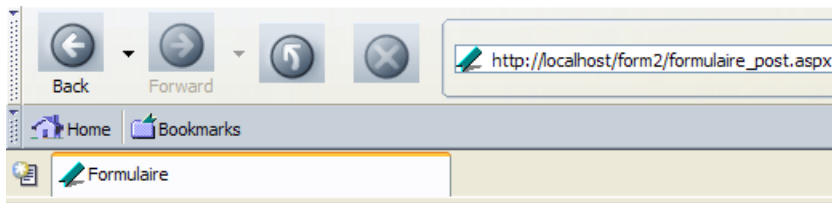
C'est pour cette raison, qu'en réponse à notre bouton [Envoyer], nous avons obtenu la même page que celle obtenue initialement en demandant l'URL [http://localhost/form2/formulaire_get.aspx] sans paramètres.

4.3.2 Méthode POST

Le document HTML est maintenant programmé pour que le navigateur utilise la méthode POST pour envoyer les valeurs du formulaire au serveur web. Pour cela, nous copions le fichier [formulaire_get.aspx] dans [formulaire_post.aspx] et modifions uniquement la balise <form> dans [formulaire_post.aspx]:

```
<%@ Page src="formulaire_get.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="formulaire_get" %>
<html>
<head>
...
</head>
<body>
<p>
Gestion d'un formulaire
</p>
<hr />
<form name="formulaire" method="post">
<table border="1">
```

Il n'y a pas nécessité de changer le contrôleur [formulaire_get.aspx.vb], aussi le gardons-nous en l'état. Nous demandons le nouveau document via l'URL [http://localhost/form2/formulaire_post.aspx], nous remplissons le formulaire tel que pour la méthode GET et nous transmettons les paramètres au serveur avec le bouton [Envoyer]. Nous obtenons du serveur la page réponse suivante :



Gestion d'un formulaire

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="*****"/>
Boîte de saisie	<input type="text" value="ligne1"/> <input type="text" value="ligne2"/>
ComboBox	<input type="text" value="choix2"/>
Liste à choix simple	<input type="text" value="liste1"/> <input type="text" value="liste2"/> <input type="text" value="liste3"/>
Liste à choix multiple	<input type="text" value="multiple1"/> <input type="text" value="multiple2"/> <input type="text" value="multiple3"/>
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

Nous obtenons donc le même résultat qu'avec la méthode GET, c.a.d. la page initiale. On remarquera une différence : dans le champ [Adresse] du navigateur, les paramètres transmis n'apparaissent pas. Maintenant, regardons la requête envoyée par le client et qui a été mémorisée dans le fichier [request.txt] :

```
POST /form2/formulaire_post.aspx HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Content-Length: 222
Content-Type: application/x-www-form-urlencoded
Accept: application/x-shockwave-flash, text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png, image/jpeg, image/gif;q=0.2, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.7, *;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/form2/formulaire_post.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316

rdMarie=oui&C1=un&C2=deux&txtSaisie=programmation+asp.net&txtMdp=ceciestsecre&areaSaisie=les+bases+de+la
%0D%0Aprogrammation+web%0D%0A&cmbValeurs=3&lstSimple=3&lstMultiple=2&lstMultiple=4&btnEnvoyer=Envoyer&se
cret=uneValeur
```

Des nouveautés apparaissent dans la requête HTTP du client :

- POST URL HTTP/1.1** la requête GET a laissé place à une requête POST. Les paramètres ne sont plus présents dans cette première ligne de la requête. On peut constater qu'ils sont maintenant placés derrière la requête HTTP après une ligne vide. Leur encodage est identique à celui qu'ils avaient dans la requête GET.
- Content-Length** nombre de caractères "postés", c.a.d. le nombre de caractères que devra lire le serveur web après avoir reçu les entêtes HTTP pour récupérer le document que lui envoie le client. Le document en question est ici la liste des valeurs du formulaire.
- Content-type** précise le type du document que le client enverra après les entêtes HTTP. Le type [application/x-www-form-urlencoded] indique que c'est un document contenant des valeurs de formulaire.

Il y a deux méthodes pour transmettre des données à un serveur web : GET et POST. Y-a-t-il une méthode meilleure que l'autre ? Nous avons vu que si les valeurs d'un formulaire étaient envoyées par le navigateur avec la méthode GET, le navigateur affichait dans son champ *Adresse* l'URL demandée sous la forme `URL?param1=val1¶m2=val2&...`. On peut voir cela comme un avantage ou un inconvénient :

- un avantage si on veut permettre à l'utilisateur de placer cette URL paramétrée dans ses liens favoris
- un inconvénient si on ne souhaite pas que l'utilisateur ait accès à certaines informations du formulaire tels, par exemple, les champs cachés

Par la suite, nous utiliserons quasi exclusivement la méthode POST dans nos formulaires.

4.4 Traitement côté serveur des valeurs d'un formulaire

4.4.1 Présentation de l'exemple

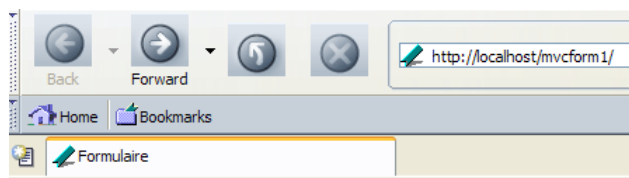
Maintenant que nous avons fait le lien entre la balise HTML `<form method="GET/POST" ...>` et la façon utilisée par le navigateur pour envoyer les valeurs du formulaire, nous savons comment côté serveur récupérer celles-ci. C'est le chapitre précédent qui nous a donné la réponse :

- si la méthode de la balise `<form>` est GET, les valeurs des paramètres seront récupérées dans la collection [Request.QueryString]
- si la méthode de la balise `<form>` est POST, les valeurs des paramètres "postés" seront récupérées dans la collection [Request.Form]. Il faut ici apporter une précision. La balise `<form>` peut préciser à l'aide de l'attribut [action] l'url cible du GET ou du POST. Cette url peut très bien être paramétrée qu'on ait un GET ou un POST, sous la forme `action="url?param1=val1¶m2=val2&..."`. Ces paramètres s'ajoutent alors à ceux compris entre les balises `<form>` et `</form>` et qui vont faire l'objet d'un transfert vers le serveur au moyen d'un GET ou d'un POST. Parce qu'ils font partie de l'url cible, ils seront obtenus dans la collection [Request.QueryString] qu'on ait affaire à un GET ou à un POST.

Nous écrivons maintenant une application MVC à deux vues :

- la première vue est le formulaire précédent. Nous l'appellerons [vue_formulaire].
- la seconde vue est une page d'informations listant les valeurs saisies dans la première page. Un lien permet le retour au formulaire. Nous appellerons cette vue [vue_confirmation].

La vue [vue_formulaire] est envoyée à l'utilisateur qui la remplit et la valide. Elle pourrait ressembler à ceci juste avant la validation :



Gestion d'un formulaire

Etés-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	<input type="text" value="programmation asp.net"/>
Mot de passe	<input type="password" value="***"/>
Boîte de saisie	<input type="text" value="les bases de la programmation web"/>
ComboBox	<input type="text" value="choix3"/>
Liste à choix simple	<input type="text" value="liste1"/> <input type="text" value="liste2"/> <input type="text" value="liste3"/> <input type="button" value="Raz"/>
Liste à choix multiple	<input type="text" value="multiple2"/> <input type="text" value="multiple3"/> <input type="text" value="multiple4"/> <input type="button" value="Raz"/>
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

L'utilisateur utilise le bouton [Envoyer] pour valider ses saisies. Il reçoit en retour, la vue [vue_validation] suivante :



Valeurs saisies

rdMarie	[oui]
C1	[un]
C2	[deux]
C3	
txtSaisie	[programmation asp.net]
txtMdp	[mdp]
areaSaisie	[les bases de la programmation web.]
cmbValeurs	[3]
lstSimple	[3]
lstMultiple	[2][4]
secret	[uneValeur]

4.4.2 Le contrôleur de l'application

Nous avons vu dans le chapitre précédent, que nous pouvions confier le rôle du contrôleur d'une application MVC au fichier [global.asax] par qui transitent toutes les requêtes des clients. Nous avons déjà présenté une application MVC construite ainsi et nous suivons ici le modèle de développement présenté alors. Le fichier [global.asax] sera le suivant :

```
<%@ Application src="Global.asax.vb" Inherits="Global" %>
```

Il est constitué d'une unique ligne référençant le contrôleur placé dans le fichier [global.asax.vb] :

```
Imports System
Imports System.Web
Imports System.Web.SessionState

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_BeginRequest (ByVal sender As Object, ByVal e As EventArgs)
        ' on récupère l'action à faire
        Dim action As String
        If Request.QueryString("action") Is Nothing Then
            action = "init"
        Else
            action = Request.QueryString("action").ToString.ToLower
        End If
        ' on exécute l'action
        Select Case action
            Case "init"
                Server.Transfer("formulaire.aspx", False)
            Case "validation"
                Server.Transfer("validation.aspx", True)
            Case Else
                Server.Transfer("formulaire.aspx", True)
        End Select
    End Sub
End Class
```

Le principe du contrôleur est le suivant :

- il attend dans l'url cible, une chaîne de paramètres contenant le paramètre [action]. Si celui-ci est absent, on fait comme si on avait eu [action=init] dans la chaîne de paramètres.
- seules deux actions sont reconnues :
 - **init** : on délivre au client le formulaire pré-rempli
 - **validation** : on délivre au client la page confirmant les saisies qu'il a faites
- si l'action n'est pas l'une des précédentes, on fait comme si on avait eu [action=init]. On pourrait aussi délivrer une page d'erreur.

4.4.3 Traitement de l'action init

Lorsque le contrôleur traite l'action "init", il doit générer un formulaire pré-rempli. C'est le rôle de la page [formulaire.aspx] de faire ce travail. Son code est le suivant :

```
<HTML>
<HEAD>
  <title>Formulaire</title>
  <script language="javascript">
    function effacer(){
      alert("Vous avez cliqué sur le bouton [Effacer]");
    }
  }
  function raz(liste){
    liste.selectedIndex=-1
  }
</script>
</HEAD>
<body>
  <p>
    Gestion d'un formulaire
  </p>
  <hr>
  <form name="formulaire" method="post" action="?action=validation">
    <table border="1">
      <tr>
        <td>
          Etes-vous marié(e)</td>
        <td>
          <p align="center">
            <input type="radio" value="oui" name="rdMarie">Oui <input type="radio" checked value="non"
name="rdMarie">Non
          </p>
        </td>
      </tr>
      ....
      <td>
        <select size="3" name="lstSimple">
          <option value="1" selected>liste1</option>
          <option value="2">liste2</option>
          <option value="3">liste3</option>
          <option value="4">liste4</option>
          <option value="5">liste5</option>
        </select>
        <INPUT type="button" value="Raz" name="btnRazSimple" onclick="raz(lstSimple)">
      </td>
      ....
      <td>
        <select multiple size="3" name="lstMultiple">
          <option value="1" selected>multiple1</option>
          <option value="2">multiple2</option>
          <option value="3" selected>multiple3</option>
          <option value="4">multiple4</option>
          <option value="5">multiple5</option>
        </select>
        <INPUT type="button" value="Raz" name="btnRazMultiple" onclick="raz(lstMultiple)">
      </td>
      ...
    </table>
    <input type="hidden" name="secret" value="uneValeur">
  </form>
</body>
</HTML>
```

Nous retrouvons le code HTML du formulaire étudié précédemment à quelques différences près. Le code de la balise <form> a été modifié :

```
<form name="formulaire" method="post" action="?action=validation">
```

- **post** : les valeurs saisies par l'utilisateur puis envoyées au serveur lorsque le bouton de type [submit] sera utilisé le seront par la méthode HTTP POST
- **action** : la syntaxe action="url" sert à désigner l'url à laquelle doivent être envoyées les valeurs du formulaire. Cette url peut comporter une chaîne de paramètres de la forme param1=val1¶m2=val2&... C'est ce qui est fait ici où nous passons le paramètre [action=validation] pour indiquer au contrôleur l'action qu'il doit entreprendre. On peut noter que cette chaîne de paramètres n'est pas précédée d'une adresse web. Le navigateur enverra alors les paramètres du formulaire à l'adresse qui lui a fourni ce formulaire. Dans notre exemple présenté plus haut, cette adresse est

[http://localhost/mvcform1]. Le navigateur fera donc une requête [POST /mvcform1?action=validation] à la machine [localhost].

Des boutons ont été ajoutés afin de permettre à l'utilisateur de désélectionner les éléments des listes [lstSimple] et [lstMultiple] :

Liste à choix simple	<input type="text" value="liste1"/> <input type="text" value="liste2"/> <input type="text" value="liste3"/> <input type="button" value="Raz"/>
Liste à choix multiple	<input type="text" value="multiple1"/> <input type="text" value="multiple2"/> <input type="text" value="multiple3"/> <input type="button" value="Raz"/>

Cela se traduit par le code HTML suivant :

```
<td>
  <select size="3" name="lstSimple">
    <option value="1" selected>liste1</option>
    <option value="2">liste2</option>
    <option value="3">liste3</option>
    <option value="4">liste4</option>
    <option value="5">liste5</option>
  </select>
  <INPUT type="button" value="Raz" name="btnRazSimple" onclick="raz (lstSimple) ">
</td>
....
<td>
  <select multiple size="3" name="lstMultiple">
    <option value="1" selected>multiple1</option>
    <option value="2">multiple2</option>
    <option value="3" selected>multiple3</option>
    <option value="4">multiple4</option>
    <option value="5">multiple5</option>
  </select>
  <INPUT type="button" value="Raz" name="btnRazMultiple" onclick="raz (lstMultiple) ">
</td>
```

Un clic sur un bouton [Raz] provoque la désélection de tout élément de la liste à laquelle il est associé. Prenons l'exemple de la liste [lstMultiple]. Un clic sur le bouton [Raz] correspondant, provoque l'exécution de la fonction Javascript [raz(lstMultiple)]. Rappelons que le code Javascript d'un document HTML affiché par un navigateur est exécuté par ce même navigateur et non pas par le serveur. Javascript est un langage très complet qui permet de donner un aspect dynamique à des pages sans intervention du serveur. La fonction [raz] est la suivante :

```
function raz(liste){
  liste.selectedIndex=-1
}
```

Cette fonction reçoit un paramètre qui est un objet Javascript représentant une liste HTML. Cet objet a des propriétés et des méthodes. L'une de ses propriétés est [selectedIndex] dont la valeur est le numéro de la première option sélectionnée dans la liste HTML. S'il n'y en a pas, cette propriété vaut -1. Inversement fixer une valeur à cette propriété revient à sélectionner un nouvel élément dans la liste HTML et lui donner la valeur -1 revient à n'avoir aucun élément sélectionné. C'est ce qui est fait ici.

On remarquera enfin que le code de présentation [formulaire.aspx] n'est pas accompagné d'un contrôleur [formulaire.aspx.vb]. En effet, il n'y a aucune partie dynamique générée par le serveur dans le document HTML donc aucune nécessité d'avoir un contrôleur.

4.4.4 Traitement de l'action validation

Lorsque le contrôleur traite l'action "validation", il doit générer une page listant les valeurs saisies par l'utilisateur. C'est le rôle de la page [validation.aspx]. Visuellement la page se présente de la façon suivante :



Valeurs saisies

rdMarie	[oui]
C1	[un]
C2	[deux]
C3	
txtSaisie	[programmation asp.net]
txtMdp	[mdp]
areaSaisie	[les bases de la programmation web.]
cmbValeurs	[3]
lstSimple	[3]
lstMultiple	[2][4]
secret	[uneValeur]

Son code HTML est le suivant :

```
<%@ Page src="validation.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="validation" %>
<HTML>
  <HEAD>
    <title>validation</title>
  </HEAD>
  <body>
    <P>Valeurs saisies</P>
    <HR width="100%" SIZE="1">
    <TABLE id="Table1" cellSpacing="1" cellPadding="1" width="300" border="1">
      <TR>
        <TD width="84">rdMarie</TD>
        <TD><% =rdMarie%></TD>
      </TR>
      <TR>
        <TD width="84">C1</TD>
        <TD><% =C1%></TD>
      </TR>
      <TR>
        <TD width="84">C2</TD>
        <TD><% =C2%></TD>
      </TR>
      <TR>
        <TD width="84">C3</TD>
        <TD><% =C3%></TD>
      </TR>
      <TR>
        <TD width="84">txtSaisie</TD>
        <TD><% =txtSaisie%></TD>
      </TR>
      <TR>
        <TD width="84">txtMdp</TD>
        <TD><% =txtMdp%></TD>
      </TR>
      <TR>
        <TD width="84">areaSaisie</TD>
        <TD><% =areaSaisie%></TD>
      </TR>
      <TR>
        <TD width="84">cmbValeurs</TD>
        <TD><% =cmbValeurs%></TD>
      </TR>
      <TR>
        <TD width="84">lstSimple</TD>
        <TD><% =lstSimple%></TD>
      </TR>
      <TR>
        <TD width="84">lstMultiple</TD>
        <TD><% =lstMultiple%></TD>
      </TR>
    </TABLE>
  </body>
</HTML>
```

```

    <TR>
      <TD width="84">secret</TD>
      <TD><%=secret%></TD>
    </TR>
  </TABLE>
</body>
</HTML>

```

Les parties dynamiques <%=variable%> du document sont calculées par le contrôleur [validation.aspx.vb] associé :

```

Imports System.Text.RegularExpressions

Public Class validation
  Inherits System.Web.UI.Page

  Protected rdMarie As String
  Protected C1 As String
  Protected C2 As String
  Protected C3 As String
  Protected txtSaisie As String
  Protected txtMdp As String
  Protected areaSaisie As String
  Protected cmbValeurs As String
  Protected lstSimple As String
  Protected lstMultiple As String
  Protected secret As String
  Protected delimitateur As New Regex("\r\n")

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'on récupère les paramètres postés
    rdMarie = getValue("rdMarie")
    C1 = getValue("C1")
    C2 = getValue("C2")
    C3 = getValue("C3")
    txtSaisie = getValue("txtSaisie")
    txtMdp = getValue("txtMdp")
    areaSaisie = String.Join(",", delimitateur.Split(getValue("areaSaisie")))
    cmbValeurs = getValue("cmbValeurs")
    lstSimple = getValue("lstSimple")
    lstMultiple = getValue("lstMultiple")
    secret = getValue("secret")
  End Sub

  Private Function getValue(ByVal champ As String) As String
    ' récupère la valeur du champ [champ] de la requête postée
    ' qq chose ?
    If Request.Form(champ) Is Nothing Then Return ""
    ' on récupère la ou les valeurs du champ
    Dim valeurs() As String = Request.Form.GetValues(champ)
    Dim valeur As String = ""
    Dim i As Integer
    For i = 0 To valeurs.Length - 1
      valeur += "[" + valeurs(i) + "]"
    Next
    Return valeur
  End Function
End Class

```

Le calcul des valeurs à afficher est fait dans la procédure [Form_Load]. La valeur d'un champ "posté" est obtenue par la fonction getValue(C) où C est le nom du champ. Les points clés de cette fonction sont les suivants :

- si C n'est pas dans la chaîne de paramètres postée alors on rend la chaîne vide pour valeur de C
- sinon on obtient le tableau des valeurs du champ C par [Request.Form.GetValues(C)]. Celles-ci sont concaténées dans une chaîne de caractères sous la forme [val1][val2]...[valn] où [val_i] est la valeur n° i du champ C

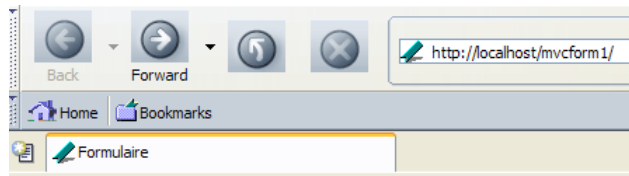
Le champ [areaSaisie] est traité de façon particulière. Sa valeur est envoyée par le navigateur sous la forme areaSaisie=ligne1\r\nligne2\r\n... où \r est le caractère de code ASCII 13 (retour chariot) et \n le caractère de code ASCII 10 (saut de ligne). La fonction [getValue(areaSaisie)] rend donc la chaîne "ligne1\r\nligne2\r\n...". Cette chaîne est découpée en lignes par la méthode [Regex.Split]. On obtient alors un tableau de chaînes {ligne1,ligne2,...}. Ce tableau est transformé en chaîne "ligne1,ligne2,..." par la méthode [String.Join]. C'est cette dernière chaîne qui sera affichée pour valeur du champ [areaSaisie]. Le but était ici de montrer comment on pouvait obtenir les différentes lignes d'un champ HTML de type [TextArea].

4.4.5 Tests

Nous plaçons tous les fichiers (global.asax, global.asax.vb, formulaire.aspx, validation.aspx, validation.aspx.vb) dans un dossier <application-path>. Nous lançons le serveur Cassini avec les paramètres (<application-path>,/mvcform1). Nous créons dans le

dossier <application-path> un fichier [default.aspx] vide puis nous demandons l'url [http://localhost/mvcform1]. Comme [/mvcform1] est le chemin virtuel d'un dossier et non d'un document, le serveur web va afficher le document [default.aspx] s'il est présent. C'est pourquoi nous avons créé celui-ci. Avant de l'afficher, le script [global.asax] va être exécuté et va finalement faire afficher la page [formulaire.aspx]. C'est pourquoi le fichier [default.aspx] peut être vide.

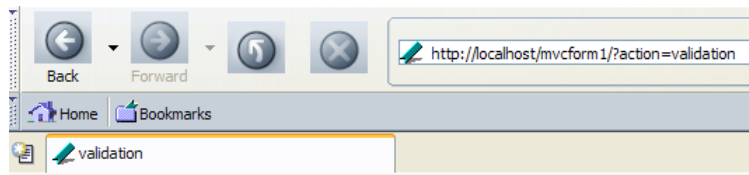
Supposons que le formulaire validé soit le suivant :



Gestion d'un formulaire

Etes-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	programmation asp.net
Mot de passe	***
Boîte de saisie	les bases de la programmation web
ComboBox	choix3
Liste à choix simple	liste1 liste2 liste3 <input type="button" value="Raz"/>
Liste à choix multiple	multiple2 multiple3 multiple4 <input type="button" value="Raz"/>
Bouton simple	<input type="button" value="Effacer"/>
Bouton submit	<input type="button" value="Envoyer"/>
Bouton reset	<input type="button" value="Rétablir"/>

L'utilisation du bouton [Envoyer] renvoie la page suivante :



Valeurs saisies

rdMarie	[oui]
C1	[un]
C2	[deux]
C3	
txtSaisie	[programmation asp.net]
txtMdp	[mdp]
areaSaisie	[les bases de la programmation web,]
cmbValeurs	[3]
lstSimple	[3]
lstMultiple	[2][4]
secret	[uneValeur]

Remarquons tout d'abord l'url de la réponse : [http://localhost/mvcform1/?action=validation]. C'est l'URL de l'attribut [action] de la balise <form> du formulaire [formulaire.aspx] :

```
<form name="formulaire" method="post" action="?action=validation">
```

Examinons une à une les valeurs obtenues pour les différents champs du formulaire :

champ	valeur	HTML	commentaires
rdMarie	oui	<pre><input type="radio" value="oui" name="rdMarie">Oui <input type="radio" checked value="non" name="rdMarie">Non</pre>	la valeur obtenue est l'attribut [value] du bouton coché
C1	un	<pre><input type="checkbox" value="un" name="C1">1</pre>	idem
C2	deux	<pre><input type="checkbox" checked value="deux" name="C2">2</pre>	idem
C3		<pre><input type="checkbox" value="trois" name="C3">3</pre>	ce bouton n'a pas été coché par l'utilisateur. Sa valeur n'a alors pas été envoyée par le navigateur. Dans le code, la condition [Request.Form("C3") is Nothing] a donc été vraie.
txtSaisie	programmation asp.net	<pre><input type="text" ... name="txtSaisie"></pre>	la valeur obtenue est le texte présent dans le champ de saisie au moment de la validation
txtMdp	mdp	<pre><input type="password" ...name="txtMdp"></pre>	idem
areaSaisie	les bases de la programmation web	<pre><textarea name="areaSaisie">...</textarea></pre>	idem
cmbValeurs	3	<pre><select name="cmbValeurs"> ... <option value="3">choix3</option> </select></pre>	la valeur obtenue est l'attribut [value] de l'option sélectionnée
lstSimple	3	<pre><select size="3" name="lstSimple"> ... <option value="3">liste3</option> ... </select></pre>	idem
lstMultiple	2 4	<pre><select multiple size="3" name="lstMultiple"> ... <option value="2">multiple2</option> ... <option value="4">multiple4</option> ... </select></pre>	les valeurs obtenues sont celles des attributs [value] des options sélectionnées
secret	uneValeur	<pre><input type="hidden" name="secret" value="uneValeur"></pre>	la valeur obtenue est l'attribut [value] du champ caché

Maintenant intéressons-nous aux listes. Supposons qu'au moment de la validation l'état du formulaire soit le suivant :

ComboBox	choix1
Liste à choix simple	liste1 liste2 liste3 Raz
Liste à choix multiple	multiple1 multiple2 multiple3 Raz

- l'option [choix1] est sélectionnée dans le comboBox
- aucune option n'est sélectionnée dans les deux autres listes. On a utilisé les boutons [Raz] pour obtenir cela.

Voici la page retournée après validation :

cmbValeurs	[1]
lstSimple	
lstMultiple	

Lorsqu'aucune valeur n'est sélectionnée dans une liste, le navigateur n'envoie pas de paramètre pour celle-ci. Dans le code de [validation.aspx.vb], les expressions [Request.Form("lstSimple")] et [Request.Form("lstMultiple")] sont donc égales à la constante [Nothing] d'où le résultat affiché ci-dessus.

4.5 Maintenir l'état d'une page

4.5.1 Maintenir l'état d'une page avec une session

Nous dupliquons la totalité de l'application précédente dans un nouveau dossier. Nous ajoutons un lien dans la page [validation.aspx] permettant à l'utilisateur de revenir au formulaire :

lstMultiple	[1][3]
secret	[uneValeur]

[Retour au formulaire](#)

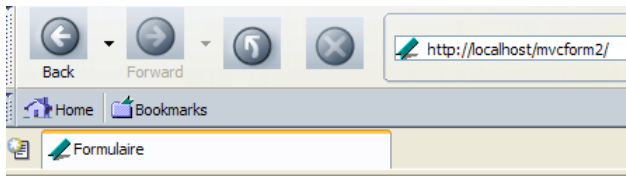
La modification à opérer sur la page [validation.aspx] est l'ajout du lien :

```
.....
    <TR>
        <TD width="84">secret</TD>
        <TD><%=secret%></TD>
    </TR>
</TABLE>
<P>
    <a href="?action=formulaire">Retour au formulaire</a>
</P>
</body>
</HTML>
```

L'attribut [href] du lien a pour valeur une url paramétrée avec pour seul paramètre [action=formulaire], permettant au serveur de savoir qu'il doit afficher le formulaire tel qu'il était lorsqu'il a été validé. Il y a donc une différence avec l'action [init] qui affiche un formulaire prédéfini. L'url paramétrée [?action=formulaire] n'a en fait pas d'url. Celle-ci sera donc la même que celle qui a permis l'affichage de la page de validation. Si on se souvient de l'étude précédente, celle-ci est l'url du dossier de l'application. La demande passera donc par le contrôleur. Celui-ci doit maintenant traiter l'action [formulaire]. Le code de [global.asax.vb] est modifié comme suit :

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' on récupère l'action à faire
    Dim action As String
    If Request.QueryString("action") Is Nothing Then
        action = "init"
    Else
        action = Request.QueryString("action").ToString.ToLower
    End If
    ' on exécute l'action
    Select Case action
        Case "init"
            Server.Transfer("formulaire.aspx", False)
        Case "validation"
            Server.Transfer("validation.aspx", True)
        Case "formulaire"
            Server.Transfer("formulaire.aspx", True)
        Case Else
            Server.Transfer("formulaire.aspx", True)
    End Select
End Sub
```

Dans le cas où l'action est "formulaire", on se contente de passer la requête à la page [formulaire.aspx]. On sait que celle-ci affiche un formulaire prédéfini dans lequel il n'y a aucune partie variable. Cette page ne peut donc afficher le formulaire avec les valeurs qu'il avait lorsqu'il a été validé. On cherche ici à mettre en évidence ce point. Si l'ensemble des fichiers de l'application ont été placés dans <application-path>, nous lançons Cassini avec les paramètres (<application-path>./mvcform2) puis nous demandons l'url [http://localhost/mvcform2]. Nous obtenons la page suivante (vue partielle) :



Gestion d'un formulaire

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

Nous remplissons le formulaire de la façon suivante puis le validons :

Etes-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input checked="" type="checkbox"/> 3

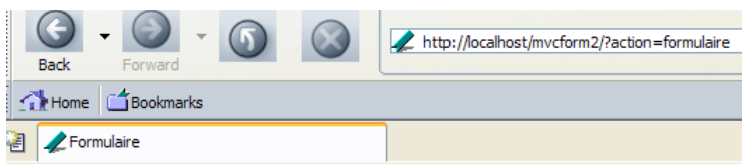
Nous obtenons la page de confirmation suivante :



Valeurs saisies

rdMarie	[oui]
C1	[un]
C2	[deux]
C3	[trois]

Nous utilisons le lien [Retour au formulaire] présent (mais non représenté) sur la page ci-dessus. Nous obtenons la réponse suivante du serveur :



Gestion d'un formulaire

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

Nous constatons que :

- l'url demandée a bien le paramètre [action=formulaire] comme il était souhaité
- le formulaire affiché a perdu les valeurs saisies.

Nous savons pourquoi nous ne retrouvons pas les valeurs saisies. Si nous regardons le code du document [formulaire.aspx], nous constatons que tout est statique et qu'il est obligé ainsi d'afficher à chaque fois la même page. On voit bien qu'il nous faut rendre ce code dynamique. Il faudra qu'il affiche les valeurs validées par l'utilisateur. Mais où mémoriser celles-ci ?

Rappelons la loi d'airain du protocole sans état HTTP :

- l'url [http://localhost/mvcform2] est demandée - une réponse est reçue. Une liaison TCP-IP a été ouverte entre le client et le serveur au début de la demande et fermée à la fin de la réponse.
- l'utilisateur saisit puis valide des données. L'url [http://localhost/mvcform2/?action=validation] est demandée - une réponse est reçue. Une nouvelle connexion TCP-IP a été ouverte puis fermée entre les deux partenaires.
- l'utilisateur utilise le lien [Retour au formulaire]. L'url [http://localhost/mvcform2/?action=formulaire] est demandée - une réponse est reçue. Une nouvelle connexion TCP-IP a été de nouveau ouverte puis fermée entre les deux partenaires.

Les cycles demande-réponse sont indépendants les uns des autres parce que chacun d'eux utilise une nouvelle connexion TCP-IP. Lorsqu'une application client-serveur utilise une unique connexion TCP-IP pour une suite d'échanges, le serveur peut identifier un client via sa connexion. Il peut donc mémoriser des informations qu'il va lier à une connexion précise et ainsi avoir un suivi des échanges. Lorsque les échanges se font sur des connexions différentes, le serveur ne peut identifier un client à une connexion. Il faut un autre moyen. Nous avons présenté l'un d'eux dans le chapitre précédent : le mécanisme de la session. Il permet aux cycles demande-réponse de mémoriser des informations dans un objet [Session] accessible à tous les cycles successifs.

Maintenant que nous avons une application MVC avec session, nous ne pouvons plus utiliser le fichier [global.asax] comme contrôleur comme il a été montré au chapitre précédent. Le rôle de contrôleur d'application doit être joué par une page particulière dédiée à cela. Ici, ce sera la page [main.aspx]. Nous procéderons ainsi :

- lorsque l'utilisateur validera ses données (action=validation), le contrôleur [main.aspx.vb] les mémorisera dans la session courante avant de faire afficher la page de validation.
- lorsqu'il faudra faire afficher le formulaire avec les valeurs saisies (action=formulaire), le contrôleur [main.aspx.vb] les récupérera dans la session et les mettra dans le contexte avant de faire afficher le formulaire.

4.5.2 Le nouveau contrôleur d'application

Le contrôleur d'application est formé des deux fichiers [main.aspx, main.aspx.vb] :

[main.aspx]

```
<%@ Page src="main.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="main" %>
```

[main.aspx.vb]

```
Imports System.Collections.Specialized
Imports Microsoft.VisualBasic

Public Class main
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère l'action à faire
        Dim action As String
        If Request.QueryString("action") Is Nothing Then
            action = "init"
        Else
            action = Request.QueryString("action").ToString.ToLower
        End If
        ' on exécute l'action
        Select Case action
            Case "init"
                ' on affiche le formulaire pré-rempli
                Context.Items("formulaire") = initForm()
                Server.Transfer("formulaire.aspx", True)
            Case "validation"
                ' on affiche la page de confirmation après avoir enregistré les valeurs postées dans la session
                Session.Item("formulaire") = Request.Form
                Server.Transfer("validation.aspx", True)
            Case "formulaire"
                ' on affiche le formulaire avec des valeurs prises dans la session
                Context.Items("formulaire") = Session.Item("formulaire")
                Server.Transfer("formulaire.aspx", True)
            Case Else
                ' on affiche le formulaire pré-rempli
                Context.Items("formulaire") = initForm()
                Server.Transfer("formulaire.aspx", True)
        End Select
    End Sub

    Private Function initForm() As NameValueCollection
        ' on initialise le formulaire
        Dim form As New NameValueCollection
        form.Set("rdMarie", "non")
        form.Set("C2", "deux")
        form.Set("txtSaisie", "qqs mots")
        form.Set("txtMdp", "ceciestsecret")
        form.Set("areasaisie", "ligne1" + ControlChars.CrLf + "ligne2" + ControlChars.CrLf)
        form.Set("cmbValeurs", "2")
        form.Set("lstSimple", "1")
        form.Set("lstMultiple", "1")
        form.Add("lstMultiple", "3")
        Return form
    End Function
End Class
```

On retrouve l'essence de la structure du contrôleur d'application étudié précédemment avec les différences suivantes :

- le travail du contrôleur se fait dans la procédure [Page_Load]
- dans le cas de l'action [validation], les valeurs du formulaire présentes dans [Request.Form] sont stockées dans la session associées à la clé "formulaire". Ensuite l'exécution est transférée à la page [validation.aspx] qui affichera ces valeurs.
- pour les autres actions, l'exécution est transférée dans tous les cas à la page [formulaire.aspx]. Celle-ci attend dans son contexte, une clé "formulaire" qui sera associée à un objet du type de [Request.form] c.a.d. de type [NameValueCollection]. Cet objet doit contenir la collection des valeurs des champs du formulaire.
- dans le cas où l'action est [init] ou une action non reconnue, la collection des valeurs est construite arbitrairement par la fonction [initForm].
- dans le cas où l'action est [formulaire], cette collection est la collection [Request.Form] qui avait été placée dans la session par l'action [validation]

4.5.3 Le nouveau formulaire

Parce que l'application n'affiche pas toujours le même contenu dans le formulaire, celui-ci doit être généré dynamiquement. La nouvelle page [formulaire.aspx] devient la suivante :

```
<%@ Page src="formulaire.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="formulaire" %>
<HTML>
<HEAD>
<title>Formulaire</title>
<script language="javascript">
function effacer(){
    alert("Vous avez cliqué sur le bouton [Effacer]");
}
function raz(liste){
    liste.selectedIndex=-1
}
</script>
</HEAD>
<body>
<p>
    Gestion d'un formulaire
</p>
<hr>
<form name="formulaire" method="post" action="main.aspx?action=validation">
<table border="1">
<tr>
<td>
        Etes-vous marié(e)</td>
<td>
<p align="center">
                <INPUT type="radio" value="oui" name="rdMarie" <%=rdouchecked%>>Oui
                <INPUT type="radio" value="non" name="rdMarie" <%=rdnonchecked%>>Non
            </p>
</td>
</tr>
<TR>
<TD>Cases à cocher
</TD>
<TD>
<p align="center">
                <INPUT type="checkbox" value="un" name="C1" <%=c1checked%>>1
                <INPUT type="checkbox" value="deux" name="C2" <%=c2checked%>>2
                <INPUT type="checkbox" value="trois" name="C3" <%=c3checked%>>3
            </p>
</TD>
</TR>
<TR>
<TD>Champ de saisie</TD>
<TD>
<p align="center">
                <INPUT type="text" maxLength="30" value="<%=txtSaisie%>" name="txtSaisie">
            </p>
</TD>
</TR>
<tr>
<td>
        Mot de passe</td>
<td>
<p align="center">
                <input type="password" maxLength="12" size="12" value="<%=txtMdp%>" name="txtMdp">
            </p>
</td>
</tr>
</table>
</body>
</HTML>
```

```

        </p>
    </td>
</tr>
<tr>
    <td>
        Boîte de saisie</td>
    <td>
        <textarea name="areaSaisie"><%=areaSaisie%></textarea>
    </td>
</tr>
<tr>
    <td>
        ComboBox</td>
    <td>
        <select name="cmbValeurs">
            <%=cmbValeursOptions%>
        </select>
    </td>
</tr>
<tr>
    <td>
        Liste à choix simple</td>
    <td>
        <select size="3" name="lstSimple">
            <%=lstSimpleOptions%>
        </select>
        <INPUT type="button" value="Raz" name="btnRazSimple" onclick="raz(lstSimple)">
    </td>
</tr>
<tr>
    <td>
        Liste à choix multiple</td>
    <td>
        <select multiple size="3" name="lstMultiple">
            <%=lstMultipleOptions%>
        </select>
        <INPUT type="button" value="Raz" name="btnRazMultiple" onclick="raz(lstMultiple)">
    </td>
</tr>
<tr>
    <td>
        Bouton simple</td>
    <td>
        <p align="center">
            <input onclick="effacer()" type="button" value="Effacer" name="btnEffacer">
        </p>
    </td>
</tr>
<tr>
    <td>
        Bouton submit</td>
    <td>
        <p align="center">
            <input type="submit" value="Envoyer" name="btnEnvoyer">
        </p>
    </td>
</tr>
<tr>
    <td>
        Bouton reset</td>
    <td>
        <p align="center">
            <input type="reset" value="Rétablir" name="btnRetablir" runat="server">
        </p>
    </td>
</tr>
</table>
<input type="hidden" name="secret" value="uneValeur">
</form>
</body>
</HTML>

```

Commentons les variables dynamiques qui apparaissent dans le code HTML :

variable	rôle
rdouisChecked	aura la valeur "checked" si le bouton radio [oui] doit être coché, la valeur "" sinon
rdnonchecked	idem pour le bouton radio [non]
c1checked	idem pour la case à cocher [C1]
c2checked	idem pour la case à cocher [C2]

c3checked	idem pour la case à cocher [C3]
txtSaisie	le texte à placer dans le champ [txtSaisie]
txtMdp	le texte à placer dans le champ [txtMdp]
areaSaisie	le texte à placer dans le champ [areaSaisie]
cmbValeursOptions	le texte HTML des options de la liste select [cmbValeurs]
lstSimpleOptions	le texte HTML des options de la liste select [lstSimple]
lstMultipleOptions	le texte HTML des options de la liste select [lstMultiple]

Les valeurs de ces variables sont calculées par le contrôleur de la page [formulaire.aspx.vb] :

```
Imports Microsoft.VisualBasic
Imports System.Collections.Specialized

Public Class formulaire
    Inherits System.Web.UI.Page

    ' champs constants du formulaire
    Private libellésCmbValeurs() As String = {"choix1", "choix2", "choix3"}
    Private valeursCmbValeurs() As String = {"1", "2", "3"}
    Private libellésLstSimple() As String = {"liste1", "liste2", "liste3"}
    Private valeursLstSimple() As String = {"1", "2", "3"}
    Private libellésLstMultiple() As String = {"multiple1", "multiple2", "multiple3", "multiple4",
"multiple5"}
    Private valeursLstMultiple() As String = {"1", "2", "3", "4", "5"}

    ' champs dynamiques du formulaire
    Protected rdouichecked As String
    Protected rdnonchecked As String
    Protected clchecked As String
    Protected c2checked As String
    Protected c3checked As String
    Protected txtSaisie As String
    Protected txtMdp As String
    Protected areaSaisie As String
    Protected cmbValeursOptions As String
    Protected lstSimpleOptions As String
    Protected lstMultipleOptions As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère la précédente requête dans la session
        Dim form As NameValueCollection
        If Not Context.Items("formulaire") Is Nothing Then
            form = Context.Items("formulaire")
        Else
            form = New NameValueCollection
        End If
        ' on prépare la page à afficher
        ' boutons radio
        rdouichecked = ""
        rdnonchecked = "checked"
        If IsEqual(form("rdMarie"), "oui") Then
            rdouichecked = "checked"
            rdnonchecked = ""
        End If
        ' cases à cocher
        clchecked = ""
        If IsEqual(form("C1"), "un") Then clchecked = "checked"
        c2checked = ""
        If IsEqual(form("C2"), "deux") Then c2checked = "checked"
        c3checked = ""
        If IsEqual(form("C3"), "trois") Then c3checked = "checked"
        ' champs de saisie
        txtSaisie = ""
        If Not form("txtSaisie") Is Nothing Then txtSaisie = form("txtSaisie").ToString
        txtMdp = ""
        If Not form("txtMdp") Is Nothing Then txtMdp = form("txtMdp").ToString
        areaSaisie = ""
        If Not form("areaSaisie") Is Nothing Then areaSaisie = form("areaSaisie").ToString
        ' listes
        Dim sélections() As String = {}
        If Not form("cmbValeurs") Is Nothing Then sélections = form.GetValues("cmbValeurs")
        cmbValeursOptions = getOptions(valeursCmbValeurs, libellésCmbValeurs, sélections)
        sélections = New String() {}
        If Not form("lstSimple") Is Nothing Then sélections = form.GetValues("lstSimple")
        lstSimpleOptions = getOptions(valeursLstSimple, libellésLstSimple, sélections)
        sélections = New String() {}
        If Not form("lstMultiple") Is Nothing Then sélections = form.GetValues("lstMultiple")
        lstMultipleOptions = getOptions(valeursLstMultiple, libellésLstMultiple, sélections)
    End Sub
End Class
```

```

Private Function getOptions(ByRef valeurs() As String, ByRef libelles() As String, ByRef sélections()
As String) As String
' rend le code HTML des options d'une balise <select>
' valeurs : tableau des valeurs des options de la balise
' libelles : tableau des libellés des options de la balise
' sélections : options à sélectionner
Dim iValeur As Integer
Dim iSelection As Integer
Dim selected As String
Dim toString As String = ""
' on parcourt la liste des valeurs des options
For iValeur = 0 To valeurs.Length - 1
' on regarde si la valeur courante doit être sélectionnée
selected = "" : iSelection = 0
Do While iSelection < sélections.Length And selected = ""
If valeurs(iValeur) = sélections(iSelection) Then selected = "selected"
iSelection += 1
Loop
' on intègre le code HTML de l'option
toString += "<option " + selected + " value='" + valeurs(iValeur) + "'> " _
+ libelles(iValeur) + "</option>" + ControlChars.CrLf
Next
' on rend le résultat
Return toString
End Function

Private Function isEqual(ByVal champ As Object, ByVal valeur As String) As Boolean
' rend vrai si le champ est égal à valeur
If champ Is Nothing OrElse champ.ToString <> valeur Then
Return false
Else
Return true
End If
end function
End Class

```

Lorsque la page [formulaire.aspx] commence à s'exécuter, elle va trouver dans son contexte une clé "formulaire" associée à la collection de valeurs des champs qu'elle doit afficher.

```

' on récupère la précédente requête dans la session
Dim form As NameValueCollection
If Not Context.Items("formulaire") Is Nothing Then
form = Context.Items("formulaire")
Else
form = New NameValueCollection
End If

```

On peut se demander pourquoi on vérifie que [Context.Items("formulaire")] existe. En effet, le contrôleur affecte une valeur à cet objet dans tous les cas. Cependant, rien n'empêche le client de demander directement la page [formulaire.aspx] sans passer par le contrôleur. Si c'était le cas, le code précédent travaillera avec une collection de valeurs vide mais il n'y aura pas de "plantage".

Le code explore la collection de valeurs qu'il a reçues pour calculer toutes les variables dynamiques de la page HTML qui lui est associée. Même s'il est long, ce code n'est pas spécialement compliqué et nous laissons le lecteur s'y plonger afin de ne pas alourdir cette explication. Nous nous attarderons cependant à la façon de générer le code HTML des trois listes de type [select]. Ce code est généré par la fonction suivante :

```

Private Function getOptions(ByRef valeurs() As String, ByRef libelles() As String, ByRef sélections()
As String) As String
' rend le code HTML des options d'une balise <select>
' valeurs : tableau des valeurs des options de la balise
' libelles : tableau des libellés des options de la balise
' sélections : options à sélectionner
Dim iValeur As Integer
Dim iSelection As Integer
Dim selected As String
Dim toString As String = ""
' on parcourt la liste des valeurs des options
For iValeur = 0 To valeurs.Length - 1
' on regarde si la valeur courante doit être sélectionnée
selected = "" : iSelection = 0
Do While iSelection < sélections.Length And selected = ""
If valeurs(iValeur) = sélections(iSelection) Then selected = "selected"
iSelection += 1
Loop
' on intègre le code HTML de l'option
toString += "<option " + selected + " value='" + valeurs(iValeur) + "'> " _
+ libelles(iValeur) + "</option>" + ControlChars.CrLf

```

```

Next
' on rend le resultat
Return toString
End Function

```

Rappelons que les options d'une balise <select> correspondent au code HTML suivant :

```
<option [selected] value="...">texte</option>
```

Pour chaque option, il y a donc trois informations à générer :

- la valeur de l'option dans l'attribut [value]
- le texte de l'option entre les balises <option> et </option>
- le mot clé [selected] si l'option doit être sélectionnée dans la liste

Pour générer ces trois informations pour chacune des options, la fonction [getOptions] reçoit trois valeurs :

1. le tableau des valeurs des options dans [valeurs]
2. le tableau des textes des options dans [libelles]
3. le tableau des valeurs à sélectionner dans [sélections]

4.5.4 La page de validation

La page de validation reste inchangée :

[validation.aspx]

```

<%@ Page src="validation.aspx.vb" Language="vb" AutoEventWireup="false" Inherits="validation" %>
<HTML>
<HEAD>
<title>validation</title>
</HEAD>
<body>
<P>Valeurs saisies</P>
<HR width="100%" SIZE="1">
<TABLE id="Table1" cellSpacing="1" cellPadding="1" width="300" border="1">
<TR>
<TD width="84">rdMarie</TD>
<TD><% =rdMarie%></TD>
</TR>
<TR>
<TD width="84">C1</TD>
<TD><% =C1%></TD>
</TR>
<TR>
<TD width="84">C2</TD>
<TD><% =C2%></TD>
</TR>
<TR>
<TD width="84">C3</TD>
<TD><% =C3%></TD>
</TR>
<TR>
<TD width="84">txtSaisie</TD>
<TD><% =txtSaisie%></TD>
</TR>
<TR>
<TD width="84">txtMdp</TD>
<TD><% =txtMdp%></TD>
</TR>
<TR>
<TD width="84">areaSaisie</TD>
<TD><% =areaSaisie%></TD>
</TR>
<TR>
<TD width="84">cmbValeurs</TD>
<TD><% =cmbValeurs%></TD>
</TR>
<TR>
<TD width="84">lstSimple</TD>
<TD><% =lstSimple%></TD>
</TR>
<TR>
<TD width="84">lstMultiple</TD>
<TD><% =lstMultiple%></TD>
</TR>
<TR>

```

```

        <TD width="84">secret</TD>
        <TD><%=secret%></TD>
    </TR>
</TABLE>
<P>
    <a href="main.aspx?action=formulaire">Retour au formulaire</a>
</P>
</body>
</HTML>

```

[validation.aspx.vb]

```

Imports Microsoft.VisualBasic
Imports System.Text.RegularExpressions

Public Class validation
    Inherits System.Web.UI.Page

    Protected rdMarie As String
    Protected C1 As String
    Protected C2 As String
    Protected C3 As String
    Protected txtSaisie As String
    Protected txtMdp As String
    Protected areaSaisie As String
    Protected cmbValeurs As String
    Protected lstSimple As String
    Protected lstMultiple As String
    Protected secret As String
    Protected delimitateur As New Regex("\r\n")

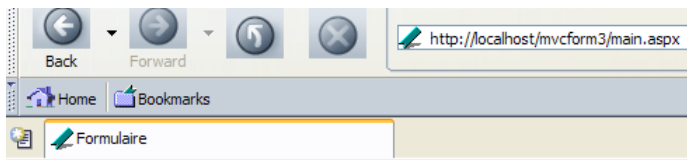
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'on récupère les paramètres postés
        rdMarie = getValue("rdMarie")
        C1 = getValue("C1")
        C2 = getValue("C2")
        C3 = getValue("C3")
        txtSaisie = getValue("txtSaisie")
        txtMdp = getValue("txtMdp")
        areaSaisie = String.Join(",", delimitateur.Split(getValue("areaSaisie")))
        cmbValeurs = getValue("cmbValeurs")
        lstSimple = getValue("lstSimple")
        lstMultiple = getValue("lstMultiple")
        secret = getValue("secret")
        ' on les sauvegarde dans la session
        'Session("formulaire") = Request.Form
    End Sub

    Private Function getValue(ByVal champ As String) As String
        ' récupère la valeur du champ [champ] de la requête postée
        ' qq chose ?
        If Request.Form(champ) Is Nothing Then Return ""
        ' on récupère la ou les valeurs du champ
        Dim valeurs() As String = Request.Form.GetValues(champ)
        Dim valeur As String = ""
        Dim i As Integer
        For i = 0 To valeurs.Length - 1
            valeur += "[" + valeurs(i) + "]"
        Next
        Return valeur
    End Function
End Class

```

4.5.5 Les tests

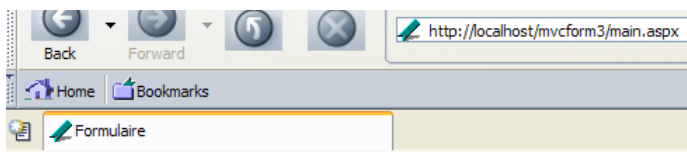
Les fichiers [main.aspx, main.aspx.vb, formulaire.aspx, formulaire.aspx.vb, validation.aspx, validation.aspx.vb] sont placés dans <application-path> et Cassini lancé avec les paramètres (<application-path>/mvcform3). Nous demandons ensuite l'url [http://localhost/mvcform3/main.aspx]. Nous obtenons le formulaire pré-initialisé :



Gestion d'un formulaire

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	qqs mots
Mot de passe	*****
Boîte de saisie	ligne1 ligne2
ComboBox	choix2
Liste à choix simple	liste1 liste2 liste3
Liste à choix multiple	multiple1 multiple2 multiple3
Bouton simple	Effacer
Bouton submit	Envoyer
Bouton reset	Rétablir

Nous remplissons le formulaire de la façon suivante :



Gestion d'un formulaire

Etes-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	programmation asp.net
Mot de passe	***
Boîte de saisie	les bases de la programmation web
ComboBox	choix1
Liste à choix simple	liste1 liste2 liste3
Liste à choix multiple	multiple2 multiple3 multiple4
Bouton simple	Effacer
Bouton submit	Envoyer
Bouton reset	Rétablir

Nous utilisons le bouton [Envoyer] ci-dessus. Nous obtenons la réponse suivante du serveur :

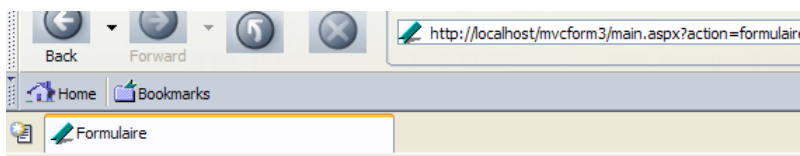


Valeurs saisies

rdMarie	[oui]
C1	[un]
C2	[deux]
C3	
txtSaisie	[programmation asp.net]
txtMdp	[mdp]
areaSaisie	[les bases de la programmation web,]
cmbValeurs	[1]
lstSimple	[3]
lstMultiple	[2][4]
secret	[uneValeur]

[Retour au formulaire](#)

Nous utilisons le lien [Retour au formulaire] ci-dessus pour revenir au formulaire. Nous obtenons la nouvelle réponse suivante :



Gestion d'un formulaire

Etes-vous marié(e)	<input checked="" type="radio"/> Oui <input type="radio"/> Non
Cases à cocher	<input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 2 <input type="checkbox"/> 3
Champ de saisie	programmation asp.net
Mot de passe	***
Boîte de saisie	les bases de la programmation web
ComboBox	choix1
Liste à choix simple	liste1 liste2 liste3
Liste à choix multiple	multiple2 multiple3 multiple4
Bouton simple	Effacer
Bouton submit	Envoyer
Bouton reset	Rétablir

Nous retrouvons bien le formulaire tel que nous l'avons validé.

4.5.6 Conclusion

L'exemple précédent nous a montré qu'il était possible de maintenir l'état d'une page au fil des cycles demande-réponse entre le client et le serveur. Néanmoins, ce travail n'est pas trivial. Nous verrons dans un chapitre ultérieur qu'il est possible avec ASP.NET de laisser le serveur rétablir de lui-même l'état d'une page.

5 Exemples

Nous nous proposons dans ce chapitre d'illustrer ce qui a été vu précédemment par une série d'exemples.

5.1 Exemple 1

5.1.1 Le problème

Cette application doit permettre à un utilisateur de calculer son impôt. On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer (chiffres 2004 pour revenus 2003) :

- 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 :

4262	0	0
8382	0.0683	291.09
14753	0.1914	1322.92
23888	0.2826	2668.39
38868	0.3738	4846.98
47932	0.4262	6883.66
0	0.4809	9505.54

Chaque ligne a 3 champs. Pour calculer l'impôt I , on recherche la première ligne où $QF \leq \text{champ1}$. Par exemple, si $QF = 5000$ on trouvera la ligne

8382 0.0683 291.09

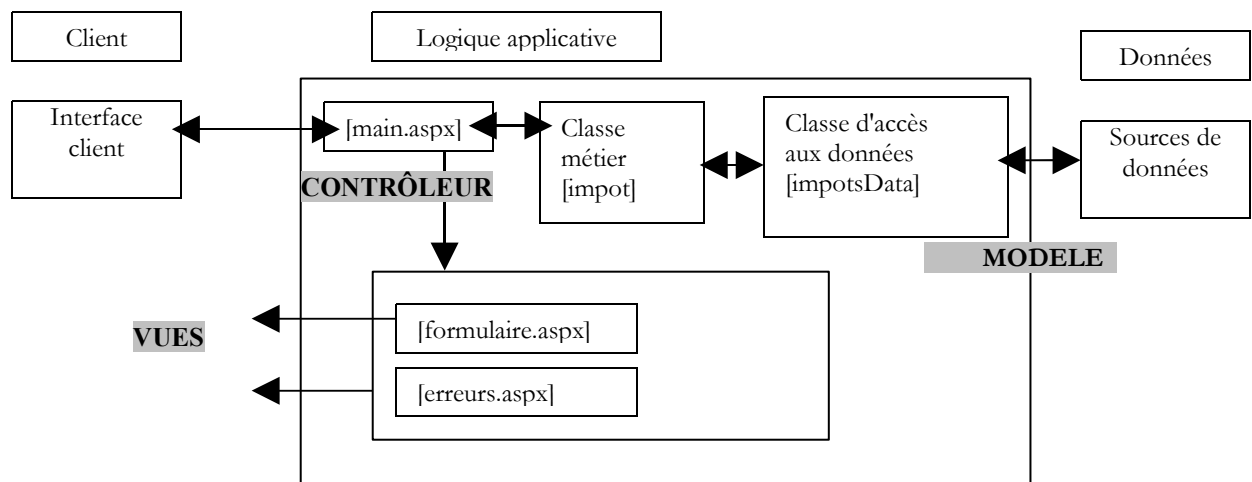
L'impôt I est alors égal à $0.0683 * R - 291.09 * \text{nbParts}$. Si QF est tel que la relation $QF \leq \text{champ1}$ n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

0 0.4809 9505.54

ce qui donne l'impôt $I = 0.4809 * R - 9505.54 * \text{nbParts}$.

5.1.2 La structure MVC de l'application

La structure MVC de l'application sera la suivante :



Le rôle du contrôleur sera joué par la page [main.aspx]. Il y aura trois actions possibles :

- **init** : correspond à la première requête du client. Le contrôleur affichera la vue [formulaire.aspx]
- **calcul** : correspond à la demande de calcul de l'impôt. Si les données du formulaire de saisie sont correctes, l'impôt est calculé grâce à la classe métier [impots]. Le contrôleur retourne au client la vue [formulaire.aspx] telle qu'elle avait été validée avec de plus, l'impôt calculé. Si les données du formulaire de saisie sont incorrectes, le contrôleur retournera la vue [erreurs.aspx] avec la liste des erreurs et un lien pour retourner au formulaire.

- **retour** : correspond au retour au formulaire après une erreur. Le contrôleur affiche la vue [formulaire.aspx] telle qu'elle a été validée avant l'erreur.

Le contrôleur [main.aspx] ne connaît rien au calcul d'impôts. Il est simplement en charge de gérer le dialogue client-serveur et de faire exécuter les actions demandées par le client. Pour l'action [calcul], il s'appuiera sur la classe métier [impot].

5.1.3 La classe métier

La classe **impot** sera définie comme suit :

```
' espaces de noms importés
Imports System

' classe
Namespace st.istia.univangers.fr
Public Class impot
    Private limites(), coeffR(), coeffN() As Decimal

    ' constructeur
    Public Sub New(ByRef source As impotsData)
        ' les données nécessaires au calcul de l'impôt
        ' proviennent d'une source extérieure [source]
        ' on les récupère - il peut y avoir une exception
        Dim data() As Object = source.GetData
        limites = CType(data(0), Decimal())
        coeffR = CType(data(1), Decimal())
        coeffN = CType(data(2), Decimal())
    End Sub

    ' calcul de l'impôt
    Public Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Long)
    As Long
        ' calcul du nombre de parts
        Dim nbParts As Decimal
        If marié Then
            nbParts = CDec(nbEnfants) / 2 + 2
        Else
            nbParts = CDec(nbEnfants) / 2 + 1
        End If
        If nbEnfants >= 3 Then
            nbParts += 0.5D
        End If
        ' calcul revenu imposable & Quotient familial
        Dim revenu As Decimal = 0.72D * salaire
        Dim QF As Decimal = revenu / nbParts
        ' calcul de l'impôt
        limites((limites.Length - 1)) = QF + 1
        Dim i As Integer = 0
        While QF > limites(i)
            i += 1
        End While
        Return CLng(revenu * coeffR(i) - nbParts * coeffN(i))
    End Function
End Class
End Namespace
```

Un objet **impôt** est créé en fournissant à son constructeur une source de données de type [impotsData]. Cette classe a une méthode publique [getData] qui permet d'obtenir les trois tableaux de données nécessaires au calcul de l'impôt et qui ont été présentés précédemment. Cette méthode peut générer une exception si les données n'ont pu être acquises ou si elles s'avèrent incorrectes. Une fois l'objet [impot] créé, on peut appeler de façon répétée sa méthode **calculer** qui calcule l'impôt du contribuable à partir de son statut marital (marié ou non), son nombre d'enfants et son salaire annuel.

5.1.4 La classe d'accès aux données

La classe [impotsData] est la classe qui permet d'accéder aux données. C'est une classe abstraite. On doit créer une classe dérivée pour chaque nouvelle source de données possible (tableaux, fichiers plats, bases de données, console, ...). Sa définition est la suivante :

```
Imports System.Collections

Namespace st.istia.univangers.fr
Public MustInherit Class impotsData
    Protected limites() As Decimal
    Protected coeffR() As Decimal
```

Exemples

```
Protected coeffn() As Decimal
Protected checked As Boolean
Protected valide As Boolean
```

```
' méthode d'accès aux données
Public MustOverride Function getData() As Object()
```

```
' méthode de vérification des données
Protected Function checkData() As Integer
' vérifie les données acquises
' on doit avoir des données
valide = Not limites Is Nothing AndAlso Not coeffr Is Nothing AndAlso Not coeffn Is Nothing
If Not valide Then Return 1
' on doit avoir 3 tableaux de même taille
If valide Then valide = limites.Length = coeffr.Length AndAlso limites.Length = coeffn.Length
If Not valide Then Return 2
' les tableaux doivent être non vides
valide = limites.Length <> 0
If Not valide Then Return 3
' chaque tableau doit contenir des éléments >=0 et en ordre croissant
valide = check(limites, limites.Length - 1) AndAlso check(coeffr, coeffr.Length) AndAlso
check(coeffn, coeffn.Length)
If Not valide Then Return 4
' tout est bon
Return 0
End Function

' vérifie la validité du contenu d'un tableau
Protected Function check(ByRef tableau() As Decimal, ByVal n As Integer) As Boolean
' tableau doit avoir ses n premiers éléments >=0 et en ordre strictement croissant
If tableau(0) < 0 Then Return False
For i As Integer = 1 To n - 1
    If tableau(i) <= tableau(i - 1) Then Return False
Next
' c'est bon
Return True
End Function
End Class
End Namespace
```

La classe a les attributs protégés suivants :

```
limites tableau des limites de tranches d'impôts
coeffr tableau des coefficients appliqués au revenu imposable
coeffn tableau des coefficients appliqués au nombre de parts
checked booléen indiquant si les données (limites, coeffr, coeffn) ont été vérifiées
valide booléen indiquant si les données (limites, coeffr, coeffn) sont valides
```

La classe n'a pas de constructeur. Elle a une méthode abstraite [getData] que les classes dérivées devront implémenter. Cette méthode a pour rôle :

- d'affecter des valeurs aux trois tableaux **limites**, **coeffr**, **coeffn**
- de lancer une exception si les données n'ont pu être acquises ou si elles se sont avérées invalides.

La classe fournit les méthodes protégées [checkData] et [check] qui vérifient la validité des attributs (limites, coeffr, coeffn). Cela dispense les classes dérivées de les implémenter. Elles n'auront qu'à les utiliser.

La première classe dérivée que nous utiliserons sera la suivante :

```
Imports System.Collections
Imports System

Namespace st.istia.univangers.fr
Public Class impotsArray
    Inherits impotsData

' constructeur sans argument
Public Sub New()
' initialisations des tableaux avec des constantes
limites = New Decimal() {4262D, 8382D, 14753D, 23888D, 38868D, 47932D, 0D}
coeffr = New Decimal() {0D, 0.0683D, 0.1914D, 0.2826D, 0.3738D, 0.4262D, 0.4809D}
coeffn = New Decimal() {0D, 291.09D, 1322.92D, 2668.39D, 4846.98D, 6883.66D, 9505.54D}
checked = True
valide = True
End Sub

' constructeur avec trois tableaux en entrée
```

```

Public Sub New(ByRef limites() As Decimal, ByRef coeffr() As Decimal, ByRef coeffn() As Decimal)
    ' on mémorise les données
    Me.limites = limites
    Me.coeffr = coeffr
    Me.coeffn = coeffn
    checked = False
End Sub

Public Overrides Function getData() As Object()
    ' on vérifie éventuellement les données
    Dim erreur As Integer
    If Not checked Then erreur = checkData() : checked = True
    ' si pas valide, alors on lance une exception
    If Not valide Then Throw New Exception("Les données des tranches d'impôts sont invalides (" +
    erreur.ToString + ")")
    ' sinon on rend les trois tableaux
    Return New Object() {limites, coeffr, coeffn}
End Function
End Class
End Namespace

```

Cette classe appelée [impotsArray] a deux constructeurs :

- un constructeur sans argument qui initialise les attributs (limites,coeffr,coeffn) de la classe de base avec des tableaux codés en "dur"
- un constructeur qui initialise les attributs (limites,coeffr,coeffn) de la classe de base avec des tableaux qui lui sont passés en paramètres

La méthode [getData] qui permettra aux classes externes d'obtenir les tableaux (limites,coeffr,coeffn) se contente de vérifier la validité des trois tableaux à l'aide de la méthode [checkData] de la classe de base. Elle lance une exception si les données sont invalides.

5.1.5 Tests des classes métier et des classes d'accès aux données

Il est important de n'inclure dans une application web que des classes métier et d'accès aux données certifiées correctes. Ainsi la phase de débogage de l'application web pourra se concentrer sur la partie contrôleur et vues. Un programme de test pourrait être le suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

Namespace st.istia.univangers.fr
    Module test
        Sub Main()
            ' programme interactif de calcul d'impôt
            ' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
            ' le programme affiche alors l'impôt à payer
            Const syntaxe As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour
            marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire
            : salaire annuel en F"

            ' création d'un objet impôt
            Dim objImpôt As impot = Nothing
            Try
                objImpôt = New impot(New impotsArray)
            Catch ex As Exception
                Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
                Environment.Exit(1)
            End Try
            ' boucle infinie
            Dim marié As String
            Dim nbEnfants As Integer
            Dim salaire As Long
            While True
                ' on demande les paramètres du calcul de l'impôt
                Console.Out.Write("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour
                arrêter :)")
                Dim paramètres As String = Console.In.ReadLine().Trim()
                ' qq chose à faire ?
                If paramètres Is Nothing OrElse paramètres = "" Then
                    Exit While
                End If
                ' vérification du nombre d'arguments dans la ligne saisie
            End While
        End Sub
    End Module
End Namespace

```

```

Dim erreur As Boolean = False
Dim args As String() = paramètres.Split(Nothing)
Dim nbParamètres As Integer = args.Length
If nbParamètres <> 3 Then
    Console.Error.WriteLine(syntaxe)
    erreur = True
End If
' vérification de la validité des paramètres
If Not erreur Then
    ' marié
    marié = args(0).ToLower()
    If marié <> "o" And marié <> "n" Then
        erreur = True
    End If
    ' nbEnfants
    Try
        nbEnfants = Integer.Parse(args(1))
        If nbEnfants < 0 Then
            Throw New Exception
        End If
    Catch
        erreur = True
    End Try
    ' salaire
    Try
        salaire = Integer.Parse(args(2))
        If salaire < 0 Then
            Throw New Exception
        End If
    Catch
        erreur = True
    End Try
End If
' si les paramètres sont corrects - on calcule l'impôt
If Not erreur Then
    Console.Out.WriteLine(("impôt=" & objImpôt.calculer(marié = "o", nbEnfants, salaire) & "
euro(s)"))
Else
    Console.Error.WriteLine(syntaxe)
End If
End While
End Sub
End Module
End Namespace

```

L'application demande à l'utilisateur de taper au clavier les trois informations dont on a besoin pour calculer son impôt :

- son statut marital : o pour marié, n pour non marié
- son nombre d'enfants
- son salaire annuel

Le calcul de l'impôt est fait à l'aide d'un objet de type [impot] créé dès le lancement de l'application :

```

' création d'un objet impôt
Dim objImpôt As impot = Nothing
Try
    objImpôt = New impot(New impotsArray)
Catch ex As Exception
    Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
    Environment.Exit(1)
End Try

```

Comme source de données, on utilise un objet de type [impotsArray]. C'est le constructeur sans argument de cette classe qui est utilisé et qui fournit les trois tableaux (limites, coeffr, coeffn) avec des valeurs en "dur". La création d'un objet [impot] peut en théorie créer une exception puisque pour se créer, l'objet va demander les données (limites,coeffr,coeffn) à sa source de données qu'on lui a passée en paramètre, et que cette acquisition de données peut lancer une exception. Il se trouve qu'ici, la méthode d'obtention des données (codage en dur) ne peut provoquer d'exception. Nous avons cependant laissé la gestion de celle-ci afin d'attirer l'attention du lecteur sur cette possibilité que l'objet [impot] se construise mal.

Voici un exemple d'exécution du programme précédent :

```

dos>dir
05/04/2004 13:28          1 337 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
21/04/2004 08:42          2 490 testimpots1.vb

```

Nous compilons l'ensemble des classes [impot, impotsData, impotsArray] dans un assemblage [impot.dll] :

```
dos>vbc /t:library /out:impot.dll impotsData.vb impotsArray.vb impots.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4
```

```
dos>dir
05/04/2004 13:28          1 337 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
21/04/2004 08:42          2 490 testimpots1.vb
21/04/2004 09:21          5 632 impot.dll
```

Nous compilons le programme de test :

```
dos>vbc /r:impot.dll testimpots1.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4
```

```
dos>dir
05/04/2004 13:28          1 337 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
21/04/2004 08:42          2 490 testimpots1.vb
21/04/2004 09:21          5 632 impot.dll
21/04/2004 09:23          4 608 testimpots1.exe
```

Nous pouvons faire les tests :

```
dos>testimpots1
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 60000
impôt=4300 euro(s)
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 2 60000
impôt=6872 euro(s)
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :
```

5.1.6 Les vues de l'application web

L'application aura deux vues : [formulaire.aspx] et [erreurs.aspx]. Illustrons le fonctionnement de l'application par des copies d'écran. La vue [formulaire.aspx] est présentée lorsque l'url [main.aspx] est demandée la première fois :

Calcul de votre impôt

Etes-vous marié(e) Oui Non

Nombre d'enfants

Salaire annuel (euro)

Impôt à payer :

L'utilisateur renseigne le formulaire :

Etes-vous marié(e) Oui Non

Nombre d'enfants

Salaire annuel (euro)

Impôt à payer :

et utilise le bouton [Calculer] pour obtenir la réponse suivante :

Calcul de votre impôt

Etes-vous marié(e) Oui Non
Nombre d'enfants
Salaire annuel (euro)
Impôt à payer : 4300 euro(s)

Calculer

Effacer

Il peut se tromper dans les données saisies :

Etes-vous marié(e) Oui Non
Nombre d'enfants
Salaire annuel (euro)
Impôt à payer : 4300 euro(s)

L'utilisation du bouton [Calculer] amène alors une autre réponse [erreurs.aspx] :

Les erreurs suivantes se sont produites :

- Le nombre d'enfants est incorrect
- Le salaire annuel est incorrect

[Retour au formulaire](#)

Il peut utiliser le lien [Retour au formulaire] ci-dessus pour retrouver la vue [formulaire.aspx] telle qu'il l'a validée avant l'erreur :

Etes-vous marié(e) Oui Non
Nombre d'enfants
Salaire annuel (euro)
Impôt à payer :

5.1.7 La vue [formulaire.aspx]

La page [formulaire.aspx] sera la suivante :

```
<%@ page src="formulaire.aspx.vb" inherits="formulaire" AutoEventWireup="false"%>
<html>
<head>
<title>Impôt</title>
</head>
<body>
<P>Calcul de votre impôt</P>
<HR>
<form method="post" action="main.aspx?action=calcul">
<TABLE border="0">
<TR>
<TD>Etes-vous marié(e)</TD>
<TD>
<INPUT type="radio" value="oui" name="rdMarie" <%=rdouichecked%>>Oui
<INPUT type="radio" value="non" name="rdMarie" <%=rdnonchecked%>>Non
</TD>
</TR>
<TR>
<TD>Nombre d'enfants</TD>
<TD><INPUT type="text" size="3" maxLength="3" name="txtEnfants" value="<%=txtEnfants%>"></TD>
</TR>
<TR>
<TD>Salaire annuel (euro)</TD>
<TD><INPUT type="text" maxLength="12" size="12" name="txtSalaire" value="<%=txtSalaire%>"></TD>
</TR>
```

```

        <TR>
            <TD>Impôt à payer :
        </TD>
        <TD><%=txtImpot%></TD>
    </TR>
</TABLE>
<hr>
<P>
    <INPUT type="submit" value="Calculer">
</P>
</form>
<form method="post" action="main.aspx?action=effacer">
    <INPUT type="submit" value="Effacer">
</form>
</body>
</html>

```

Les champs dynamiques de cette page sont les suivants :

rdouichecked	"checked" si la case [oui] doit être cochée, "" sinon
rdnonchecked	idem pour la case [non]
txtEnfants	valeur à placer dans le champ de saisie [txtEnfants]
txtSalaire	valeur à placer dans le champ de saisie [txtSalaire]
txtImpot	valeur à placer dans le champ de saisie [txtImpot]

La page a deux formulaires, chacun ayant un bouton [submit]. Le bouton [Calculer] est le bouton [submit] du formulaire suivant :

```

<form method="post" action="main.aspx?action=calcul">
...
    <P>
        <INPUT type="submit" value="Calculer">
    </P>
</form>

```

On voit que les paramètres du formulaire seront postés au contrôleur avec [action=calcul]. Le bouton [Effacer] est le bouton [submit] du formulaire suivant :

```

<form method="post" action="main.aspx?action=effacer">
    <INPUT type="submit" value="Effacer">
</form>

```

On voit que les paramètres du formulaire seront postés au contrôleur avec [action=effacer]. Ici, le formulaire n'a aucun paramètre. Il n'y a que l'action qui importe.

Les champs de [formulaire.aspx] sont calculés par [formulaire.aspx.vb] :

```

Imports System.Collections.Specialized

Public Class formulaire
    Inherits System.Web.UI.Page

    ' champs de la page
    Protected rdouichecked As String
    Protected rdnonchecked As String
    Protected txtEnfants As String
    Protected txtSalaire As String
    Protected txtImpot As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère la précédente requête dans le contexte
        Dim form As NameValueCollection = Context.Items("formulaire")

        ' on prépare la page à afficher
        ' boutons radio
        rdouichecked = ""
        rdnonchecked = "checked"
        If form("rdMarie").ToString = "oui" Then
            rdouichecked = "checked"
            rdnonchecked = ""
        End If

        ' le reste
        txtEnfants = CType(form("txtEnfants"), String)
        txtSalaire = CType(form("txtSalaire"), String)
        txtImpot = CType(Context.Items("txtImpot"), String)

    End Sub
End Class

```

Le calcul des champs de [main.aspx] se fait à partir de deux informations placées par le contrôleur dans le contexte de la page :

- Context.Items("formulaire") : dictionnaire de type [NameValueCollection] contenant les valeurs des champs HTML [rdmarie,txtEnfants,txtSalaire]
- Context.Items("txtImpot") : valeur de l'impôt

5.1.8 La vue [erreurs.aspx]

La vue [erreurs.aspx] est celle qui affiche les erreurs éventuelles qui peuvent se produire lors de la vie de l'application. Son code de présentation est le suivant :

```
<%@ page src="erreurs.aspx.vb" inherits="erreurs" AutoEventWireup="false"%>
<HTML>
  <HEAD>
    <title>Impôt</title>
  </HEAD>
  <body>
    <P>Les erreurs suivantes se sont produites :</P>
    <HR>
    <ul>
      <%=erreursHTML%>
    </ul>
    <a href="<%=href%>">
      <%=lien%>
    </a>
  </body>
</HTML>
```

La page a trois champs dynamiques :

erreursHTML	code HTML d'une liste d'erreurs
href	url d'un lien
lien	texte du lien

Ces champs sont calculés par la partie contrôleur de la page dans [erreurs.aspx.vb] :

```
Imports System.Collections
Imports Microsoft.VisualBasic

Public Class erreurs
  Inherits System.Web.UI.Page

  ' paramètre de page
  Protected erreursHTML As String = ""
  Protected href As String
  Protected lien As String

  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' on récupère les éléments du contexte
    Dim erreurs As ArrayList = CType(context.Items("erreurs"), ArrayList)
    href = context.Items("href").ToString
    lien = context.Items("lien").ToString
    ' on génère le code HTML de la liste
    Dim i As Integer
    For i = 0 To erreurs.Count - 1
      erreursHTML += "<li> " + erreurs(i).ToString + "</li>" + ControlChars.CrLf
    Next
  End Sub
End Class
```

Le contrôleur de la page récupère des informations placées par le contrôleur de l'application dans le contexte de la page :

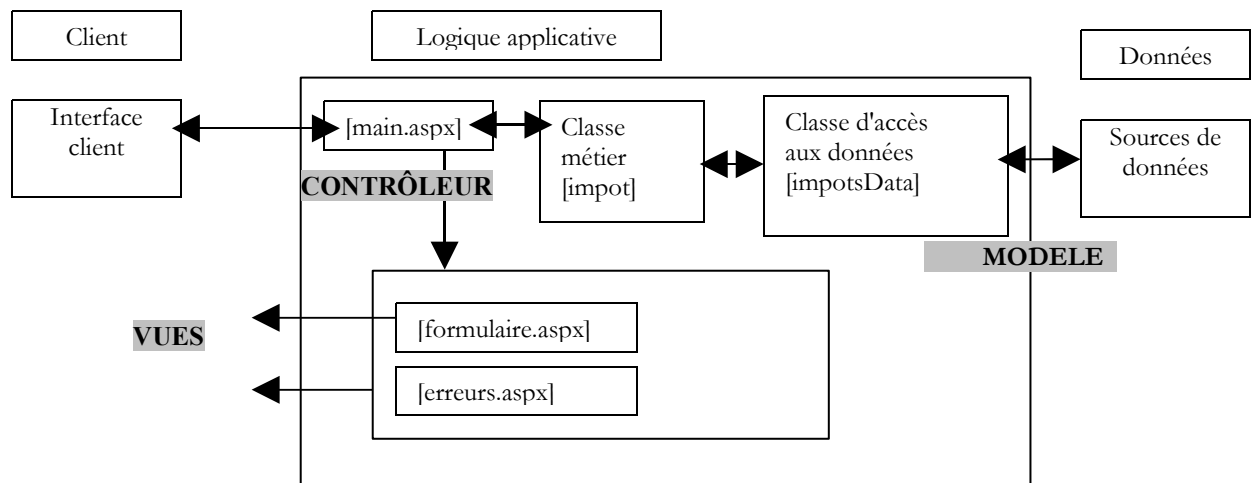
Context.Items("erreurs")	objet ArrayList contenant la liste des messages d'erreurs à afficher
Context.Items("href")	url d'un lien
Context.Items("lien")	texte du lien

Maintenant que nous savons ce que voit l'utilisateur de l'application, nous pouvons passer à l'écriture du contrôleur de celle-ci.

5.1.9 Les contrôleurs [global.asax, main.aspx]

Rappelons le schéma MVC de notre application :

Exemples



Le contrôleur [main.aspx] a à traiter trois actions :

- **init** : correspond à la première requête du client. Le contrôleur affiche la vue [formulaire.aspx]
- **calcul** : correspond à la demande de calcul de l'impôt. Si les données du formulaire de saisie sont correctes, l'impôt est calculé grâce à la classe métier [impots]. Le contrôleur retourne au client la vue [formulaire.aspx] telle qu'elle avait été validée avec de plus l'impôt calculé. Si les données du formulaire de saisie sont incorrectes, le contrôleur retourne la vue [erreurs.aspx] avec la liste des erreurs et un lien pour retourner au formulaire.
- **retour** : correspond au retour au formulaire après une erreur. Le contrôleur affiche la vue [formulaire.aspx] telle qu'elle a été validée avant l'erreur.

On sait par ailleurs, que toute requête vers l'application transite via le contrôleur [global.aspx] s'il existe. Nous avons alors à l'entrée de l'application, une chaîne de deux contrôleurs :

- [global.aspx] qui de par l'architecture ASP.NET reçoit toute requête vers l'application
- [main.aspx] qui de par la décision du développeur reçoit également toute requête vers l'application

La nécessité de [main.aspx] vient du fait que nous aurons une session à gérer. Nous avons vu que [global.aspx] ne convenait pas comme contrôleur dans ce cas. On pourrait ici se passer totalement de [global.aspx]. Nous allons cependant l'utiliser pour exécuter du code au démarrage de l'application. Le schéma MVC ci-dessus montre que nous allons avoir besoin de créer un objet [impot] pour calculer l'impôt. Il est inutile de créer celui-ci plusieurs fois, une fois suffit. Nous allons donc le créer au démarrage de l'application lors de l'événement [Application_Start] géré par le contrôleur [global.aspx]. Le code de celui-ci est le suivant :

[global.aspx]

```
<%@ Application src="Global.aspx.vb" Inherits="Global" %>
```

[global.aspx.vb]

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' on crée un objet impot
        Dim objImpot As impot
        Try
            objImpot = New impot(New impotsArray)
            ' on met l'objet dans l'application
            Application("objImpot") = objImpot
            ' pas d'erreur
            Application("erreur") = False
        Catch ex As Exception
            'il y a eu erreur, on le note dans l'application
            Application("erreur") = True
        End Try
    End Sub
End Class
```

Une fois créé, l'objet de type [impot] est mis dans l'application. C'est là que les différentes requêtes des différents clients iront le chercher. Comme la construction de l'objet [impot] peut échouer, nous gérons l'éventuelle exception et plaçons une clé [erreur] dans l'application pour signaler s'il y a eu ou non une erreur lors de la création de l'objet [impot].

Le code du contrôleur [main.aspx, main.aspx.vb] sera le suivant :

[main.aspx]

```
<%@ page src="main.aspx.vb" inherits="main" AutoEventWireup="false"%>
```

[main.aspx.vb]

```
Imports System
Imports System.Collections.Specialized
Imports System.Collections
Imports st.istia.univangers.fr

Public Class main
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' avant tout, on regarde si l'application a pu s'initialiser correctement
        If CType(Application("erreur"), Boolean) Then
            ' on redirige vers la page d'erreurs
            Dim erreurs As New ArrayList
            erreurs.Add("Application momentanément indisponible...")
            context.Items("erreurs") = erreurs
            context.Items("lien") = ""
            context.Items("href") = ""
            Server.Transfer("erreurs.aspx")
        End If
        ' on récupère l'action à faire
        Dim action As String
        If Request.QueryString("action") Is Nothing Then
            action = "init"
        Else
            action = Request.QueryString("action").ToString.ToLower
        End If
        ' on exécute l'action
        Select Case action
            Case "init"
                ' init application
                initAppli()
            Case "calcul"
                ' calcul impot
                calculImpot()
            Case "retour"
                ' retour au formulaire
                retourFormulaire()
            Case "effacer"
                ' init application
                initAppli()
            Case Else
                ' action inconnue = init
                initAppli()
        End Select
    End Sub

    Private Sub initAppli()
        ' on affiche le formulaire pré-rempli
        Context.Items("formulaire") = initForm()
        Context.Items("txtImpot") = ""
        Server.Transfer("formulaire.aspx", True)
    End Sub

    Private Function initForm() As NameValueCollection
        ' on initialise le formulaire
        Dim form As New NameValueCollection
        form.Set("rdMarie", "non")
        form.Set("txtEnfants", "")
        form.Set("txtSalaire", "")
        Return form
    End Function

    Private Sub calculImpot()
        ' on vérifie la validité des données saisies
        Dim erreurs As ArrayList = checkData()
        ' s'il y a des erreurs, on le signale
        If erreurs.Count <> 0 Then
```

```

' on sauvegarde les saisies dans la session
Session.Item("formulaire") = Request.Form
' on prépare la page d'erreurs
context.Items("href") = "main.aspx?action=retour"
context.Items("lien") = "Retour au formulaire"
context.Items("erreurs") = erreurs
Server.Transfer("erreurs.aspx")
End If
' ici pas d'erreurs - on calcule l'impôt
Dim impot As Long = CType(Application("objImpot"), impot).calculer( _
Request.Form("rdMarie") = "oui", _
CType(Request.Form("txtEnfants"), Integer), _
CType(Request.Form("txtSalaire"), Long))
' on affiche la page de résultat
context.Items("txtImpot") = impot.ToString + " euro(s)"
context.Items("formulaire") = Request.Form
Server.Transfer("formulaire.aspx", True)
End Sub

Private Sub retourFormulaire()
' on affiche le formulaire avec des valeurs prises dans la session
Context.Items("formulaire") = Session.Item("formulaire")
Context.Items("txtImpot") = ""
Server.Transfer("formulaire.aspx", True)
End Sub

Private Function checkData() As ArrayList
' au départ pas d'erreurs
Dim erreurs As New ArrayList
Dim erreur As Boolean = False
' bouton radio marié
Try
Dim rdMarie As String = Request.Form("rdMarie").ToString
If rdMarie <> "oui" And rdMarie <> "non" Then
Throw New Exception
End If
Catch
erreurs.Add("Vous n'avez pas indiqué votre statut marital")
End Try
' nbre d'enfants
Try
Dim txtEnfants As String = Request.Form("txtEnfants").ToString
Dim nbEnfants As Integer = CType(txtEnfants, Integer)
If nbEnfants < 0 Then Throw New Exception
Catch
erreurs.Add("Le nombre d'enfants est incorrect")
End Try
' salaire
Try
Dim txtSalaire As String = Request.Form("txtSalaire").ToString
Dim salaire As Integer = CType(txtSalaire, Long)
If salaire < 0 Then Throw New Exception
Catch
erreurs.Add("Le salaire annuel est incorrect")
End Try
' on rend la liste des erreurs
Return erreurs
End Function
End Class

```

Le contrôleur commence par vérifier que l'application s'est correctement initialisée :

```

' avant tout, on regarde si l'application a pu s'initialiser correctement
If CType(Application("erreur"), Boolean) Then
' on redirige vers la page d'erreurs
Dim erreurs As New ArrayList
erreurs.Add("Application momentanément indisponible...")
context.Items("erreurs") = erreurs
context.Items("lien") = ""
context.Items("href") = ""
Server.Transfer("erreurs.aspx")
End If

```

Si le contrôleur découvre que l'application n'a pas pu s'initialiser correctement (l'objet [impot] nécessaire au calcul n'a pu être créé), alors il fait afficher la page d'erreurs avec les paramètres adéquats. Ici, il n'y a pas lieu de placer le lien de retour sur le formulaire puisque la totalité de l'application s'avère indisponible. Un message d'erreur général est placé dans [Context.Items("erreurs")] de type [ArrayList].

Si le contrôleur découvre que l'application est opérationnelle, il analyse alors l'action qu'on lui demande d'exécuter via le paramètre [action]. Nous avons déjà rencontré souvent ce mode de fonctionnement. Le traitement de chaque type d'action est délégué à une fonction.

5.1.9.1 Les actions init, effacer

Ces deux actions doivent faire afficher le formulaire de saisie vide. On rappelle que celui-ci (cf vues) a deux paramètres :

- Context.Items("formulaire") : dictionnaire de type [NameValueCollection] contenant les valeurs des champs HTML [rdmarie,txtEnfants,txtSalaire]
- Context.Items("txtImpot") : valeur de l'impôt

La fonction [initAppli] initialise ces deux paramètres de façon à afficher un formulaire vide.

5.1.9.2 L'action calcul

Cette action doit calculer l'impôt à payer à partir des données saisies dans le formulaire et renvoyer celui-ci pré-rempli avec les valeurs saisies et avec de plus le montant de l'impôt calculé. La fonction [calculImpot] chargée de ce travail commence par vérifier que les données du formulaire sont bien correctes :

- le champ [rdMarie] doit être présent et avoir la valeur [oui] ou [non]
- le champ [txtEnfants] doit être présent et être un entier ≥ 0
- le champ [txtSalaire] doit être présent et être un entier ≥ 0

Si les données saisies se révèlent invalides, le contrôleur fait afficher la vue [erreurs.aspx] en ayant auparavant mis dans le contexte les valeurs attendues par celle-ci :

- les messages d'erreurs sont placés dans un objet [ArrayList], objet placé ensuite dans le contexte [Context.Items("erreurs")]
- l'url du lien de retour et le texte de ce lien sont également placés dans le contexte.

Avant de passer la main à la page [erreurs.aspx] qui va envoyer la réponse au client, les valeurs saisies dans le formulaire (Request.Form) sont placées dans la session, associées à la clé "formulaire". Ceci permettra à une requête ultérieure de les récupérer.

On peut se demander ici s'il est utile de vérifier que les champs [rdMarie, txtEnfants, txtSalaire] sont présents dans la requête envoyée par le client. C'est inutile si on est sûr que notre client est un navigateur ayant reçu la vue [formulaire.aspx] qui contient ces champs. On ne peut jamais être sûr de cela. Nous montrerons un peu plus tard un exemple où le client est l'application [curl] déjà rencontrée. Nous interrogerons l'application sans envoyer les champs qu'elle attend et nous verrons comment elle réagira. C'est une règle déjà énoncée à plusieurs reprises et que nous rappelons ici : une application ne doit jamais faire d'hypothèses sur le type de client qui l'interroge. Par sécurité, elle doit considérer qu'elle peut être interrogée par une application programmée qui peut lui envoyer des chaînes de paramètres inattendues. Elle doit correctement se comporter dans tous les cas.

Dans notre cas, nous avons vérifié que les champs [rdMarie, txtEnfants, txtSalaire] étaient présents dans la requête mais pas que celle-ci pouvait en contenir d'autres. Dans cette application, ils seraient ignorés. Néanmoins, toujours par mesure de sécurité, il serait intéressant d'enregistrer ce type de demande dans un fichier de logs et de remonter une alerte à l'administrateur de l'application afin que celui-ci sache que l'application reçoit des demandes "bizarres". En analysant celles-ci dans le fichier de logs, il pourrait détecter une éventuelle attaque de l'application et prendre alors les mesures nécessaires à la protection de celle-ci.

Si les données attendues sont correctes, le contrôleur lance le calcul de l'impôt avec l'objet [impot] stocké dans l'application. Puis il stocke dans le contexte les deux informations attendues par la vue [formulaire.aspx] :

- Context.Items("formulaire") : dictionnaire de type [NameValueCollection] contenant les valeurs des champs HTML [rdmarie,txtEnfants,txtSalaire], ici [Request.Form], c.a.d. les valeurs saisies précédemment dans le formulaire
- Context.Items("txtImpot") : valeur de l'impôt qui vient d'être obtenue

Le lecteur attentif s'est peut être posé une question à la lecture de ce qui précède : puisque l'objet [impot] créé au démarrage de l'application est partagé entre toutes les requêtes, ne peut-il y avoir de conflits d'accès entraînant une corruption des données de l'objet [impot]. Pour répondre à cette question, il nous faut revenir au code de la classe [impot]. Les requêtes font appel à la méthode [impot].calculerImpot pour obtenir l'impôt à payer. C'est donc ce code qu'il nous faut examiner :

```
Public Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Long)
As Long
    ' calcul du nombre de parts
    Dim nbParts As Decimal
    If marié Then
        nbParts = CDec(nbEnfants) / 2 + 2
    Else
        nbParts = CDec(nbEnfants) / 2 + 1
    End If
End Function
```

```

End If
If nbEnfants >= 3 Then
    nbParts += 0.5D
End If
' calcul revenu imposable & Quotient familial
Dim revenu As Decimal = 0.72D * salaire
Dim QF As Decimal = revenu / nbParts
' calcul de l'impôt
limites((limites.Length - 1)) = QF + 1
Dim i As Integer = 0
While QF > limites(i)
    i += 1
End While
Dim impot As Long = CLng(revenu * coeffR(i) - nbParts * coeffN(i))
Return impot
End Function

```

Supposons qu'un thread soit en train d'exécuter la méthode précédente et qu'il soit interrompu. Un autre thread exécute alors la méthode. Quels sont les risques ? Pour le savoir, nous avons rajouté le code suivant :

```

Dim impot As Long = CLng(revenu * coeffR(i) - nbParts * coeffN(i))
' on patiente 10 secondes
Thread.Sleep(10000)
Return impot

```

Le thread 1, après avoir calculé la valeur [impot1] de la variable locale [impot] est interrompu. Le thread 2 s'exécute alors et calcule une nouvelle valeur [impot2] pour cette même variable [impot] avant d'être interrompu. Le thread 1 récupère la main. Que retrouve-t-il dans la variable locale [impot] ? Cette variable étant locale à une méthode est stockée dans une structure de la mémoire appelée **pile**. Cette pile fait partie du contexte du thread qui est sauvegardé lorsque celui-ci est interrompu. Lorsque le thread 2 s'installe, son contexte est mis en place avec une nouvelle pile et donc une nouvelle variable locale [impot]. Lorsque le thread 2 va être interrompu à son tour, son contexte sera à son tour sauvegardé. Lorsque le thread 1 est relancé, son contexte est restauré dont sa pile. Il retrouve alors sa variable locale [impot] et non celle du thread 2. On est donc dans une situation où il n'y a pas de conflits d'accès entre requêtes. Les tests faits avec la pause de 10 secondes ci-dessus ont confirmé que les requêtes simultanées obtenaient bien le résultat attendu.

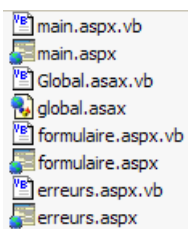
5.1.9.3 L'action retour

Cette action correspond à l'activation du lien [Retour vers le formulaire] de la vue [erreurs.aspx] pour revenir à la vue [formulaire.aspx] pré-rempli avec les valeurs saisies précédemment et sauvegardées dans la session. La fonction [retourFormulaire] récupère cette information. Les deux paramètres attendus par la vue [formulaire.aspx] sont initialisés :

- Context.Items("formulaire") avec les valeurs saisies précédemment et sauvegardées dans la session
- Context.Items("txtImpot") avec la chaîne vide

5.1.10 Test de l'application web

L'ensemble des fichiers précédents sont placés dans un dossier <application-path>.



Dans ce dossier, est créé un sous-dossier [bin] dans lequel est placé l'assemblage [impot.dll] issu de la compilation des fichiers des classes métier : [impots.vb, impotsData.vb, impotsArray.vb]. On rappelle ci-dessous, la commande de compilation nécessaire :

```

dos>vbc /t:library /out:impot.dll impotsData.vb impotsArray.vb impots.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4

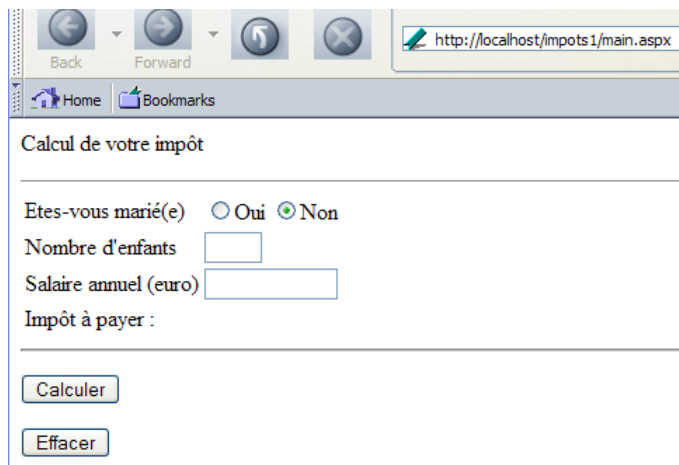
```

```

dos>dir
05/04/2004 13:28          1 337 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
21/04/2004 09:21          5 632 impot.dll

```


Le fichier [impot.dll] ci-dessus doit être placé dans <application-path>\bin afin que l'application web y ait accès. Le serveur Cassini est lancé avec les paramètres (<application-path>,/impots1). Avec un navigateur, nous demandons l'url [http://localhost/impots1/main.aspx] :



Calcul de votre impôt

Etes-vous marié(e) Oui Non

Nombre d'enfants

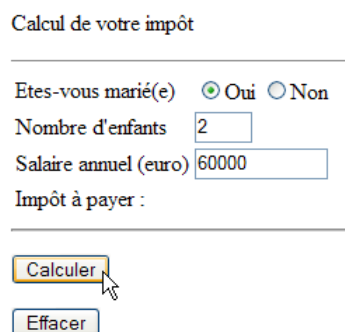
Salaire annuel (euro)

Impôt à payer :

Calculer

Effacer

Nous remplissons le formulaire :



Calcul de votre impôt

Etes-vous marié(e) Oui Non

Nombre d'enfants

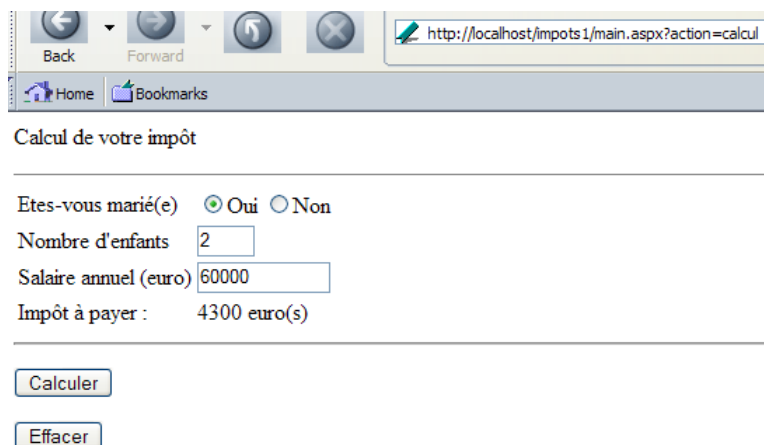
Salaire annuel (euro)

Impôt à payer :

Calculer

Effacer

Puis nous lançons le calcul de l'impôt avec le bouton [Calculer]. Nous obtenons la réponse suivante :



Calcul de votre impôt

Etes-vous marié(e) Oui Non

Nombre d'enfants

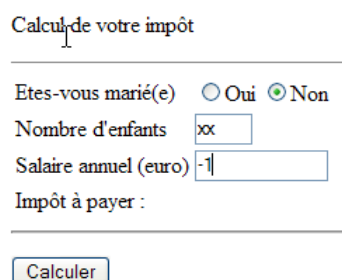
Salaire annuel (euro)

Impôt à payer : 4300 euro(s)

Calculer

Effacer

Puis nous faisons des saisies erronées :



Calcul de votre impôt

Etes-vous marié(e) Oui Non

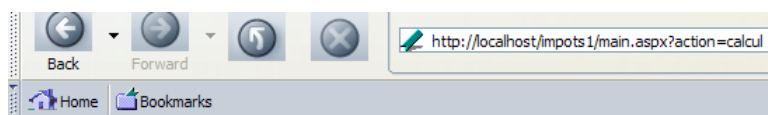
Nombre d'enfants

Salaire annuel (euro)

Impôt à payer :

Calculer

L'appui sur le bouton [Calculer] amène la réponse suivante :

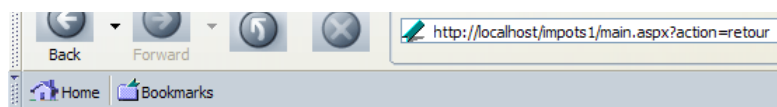


Les erreurs suivantes se sont produites :

- ◆ Le nombre d'enfants est incorrect
- ◆ Le salaire annuel est incorrect

[Retour au formulaire](#)

L'utilisation du lien [Retour au formulaire] nous ramène au formulaire dans l'état où il était lorsqu'il a été validé :



Calcul de votre impôt

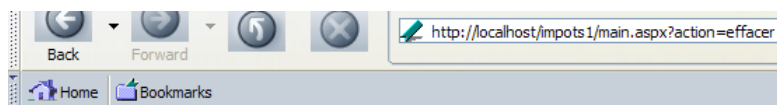
Etes-vous marié(e) Oui Non

Nombre d'enfants

Salaire annuel (euro)

Impôt à payer :

Enfin, l'utilisation du bouton [Effacer] réinitialise la page :



Calcul de votre impôt

Etes-vous marié(e) Oui Non

Nombre d'enfants

Salaire annuel (euro)

Impôt à payer :

5.1.11 Utilisation du client [curl]

Il est important de tester les applications web avec d'autres clients que les navigateurs. Si on envoie à un navigateur un formulaire avec des paramètres à poster lorsqu'il sera validé, le navigateur renverra les valeurs de ces paramètres au serveur au moment du POST. Un autre client pourrait ne pas le faire et alors le serveur aurait une requête où il manquerait des paramètres. Il doit savoir quoi faire dans ce cas. Un autre exemple est celui des vérifications de saisie faites côté client. Si le formulaire contient des données à vérifier, cette vérification peut être faite côté client grâce à des scripts inclus dans le document contenant le formulaire. Le navigateur ne postera le formulaire que si toutes les données vérifiées côté client sont valides. On pourrait alors être tenté, côté serveur, de considérer qu'on va recevoir des données vérifiées et ne pas vouloir faire cette vérification une seconde fois. Ce serait une erreur. En effet, un client autre qu'un navigateur pourrait envoyer au serveur des données invalides et alors l'application web risque d'avoir un comportement inattendu. Nous allons illustrer ces points en utilisant le client [curl].

Tout d'abord, nous demandons l'url [http://localhost/impots1/main.aspx] :

```
dos>curl --include --url http://localhost/impots1/main.aspx
```

Exemples

```

HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 15:18:10 GMT
Set-Cookie: ASP.NET_SessionId=ivthkl45tjdrzznevqsf255; path=/
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 982
Connection: Close

```

```

<html>
  <head>
    <title>Impôt</title>
  </head>
  <body>
    <P>Calcul de votre impôt</P>
    <HR width="100%" SIZE="1">
    <form method="post" action="main.aspx?action=calcul">
      <TABLE border="0">
        <TR>
          <TD>Etes-vous marié(e)</TD>
          <TD>
            <INPUT type="radio" value="oui" name="rdMarie" >Oui <INPUT type="radio" value="non"
name="rdMarie" checked>Non</TD>
          </TR>
          <TR>
            <TD>Nombre d'enfants</TD>
            <TD><INPUT type="text" size="3" maxLength="3" name="txtEnfants" value=""></TD>
          </TR>
          <TR>
            <TD>Salaire annuel (euro)</TD>
            <TD><INPUT type="text" maxLength="12" size="12" name="txtSalaire" value=""></TD>
          </TR>
          <TR>
            <TD>Impôt à payer :
            </TD>
          </TR>
        </TABLE>
        <hr>
        <P>
          <INPUT type="submit" value="Calculer">
        </P>
      </form>
      <form method="post" action="main.aspx?action=effacer">
        <INPUT type="submit" value="Effacer">
      </form>
    </body>
</html>

```

Le serveur nous a envoyé le code HTML du formulaire. Dans les entêtes HTTP nous avons le cookie de session. Nous allons l'utiliser dans les requêtes suivantes afin de maintenir la session. Demandons l'action [calcul] sans fournir de paramètres :

```

dos>curl --cookie ASP.NET_SessionId=ivthkl45tjdrzznevqsf255 --include --url
http://localhost/impots1/main.aspx?action=calcul

```

```

HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 15:22:42 GMT
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 380
Connection: Close

<HTML>
  <HEAD>
    <title>Impôt</title>
  </HEAD>
  <body>
    <P>Les erreurs suivantes se sont produites :</P>
    <HR>
    <ul>
      <li> Vous n'avez pas indiqué votre statut marital</li>
      <li> Le nombre d'enfants est incorrect</li>
      <li> Le salaire annuel est incorrect</li>
    </ul>
    <a href="main.aspx?action=retour">

```

```
Retour au formulaire
</a>
</body>
</HTML>
```

Nous pouvons constater que l'application web a renvoyé la vue [erreurs] avec trois messages d'erreurs pour les trois paramètres manquants. Envoyons maintenant des paramètres erronés :

```
dos>curl --cookie ASP.NET_SessionId=ivthkl45tjdrzznevqsf255 --include --data rdMarie=xx --data
txtEnfants=xx --data txtSalaire=xx --url http://localhost/impots1/main.aspx?action=calcul
```

```
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 15:25:50 GMT
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 380
Connection: Close
```

```
<HTML>
<HEAD>
<title>Impôt</title>
</HEAD>
<body>
<P>Les erreurs suivantes se sont produites :</P>
<HR>
<ul>
<li> Vous n'avez pas indiqué votre statut marital</li>
<li> Le nombre d'enfants est incorrect</li>
<li> Le salaire annuel est incorrect</li>
</ul>
<a href="main.aspx?action=retour">
Retour au formulaire
</a>
</body>
</HTML>
```

Les trois erreurs ont été correctement détectées. Maintenant envoyons des paramètres valides :

```
dos>curl --cookie ASP.NET_SessionId=ivthkl45tjdrzznevqsf255 --include --data rdMarie=oui --data
txtEnfants=2 --data txtSalaire=60000 --url http://localhost/impots1/main.aspx?action=calcul
```

```
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Thu, 01 Apr 2004 15:28:24 GMT
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1000
Connection: Close
```

```
<html>
<head>
<title>Impôt</title>
</head>
<body>
<P>Calcul de votre impôt</P>
<HR width="100%" SIZE="1">
<form method="post" action="main.aspx?action=calcul">
<TABLE border="0">
<TR>
<TD>Etes-vous marié(e)</TD>
<TD>
<INPUT type="radio" value="oui" name="rdMarie" checked>Oui <INPUT type="radio" value="non"
name="rdMarie" >Non</TD>
</TR>
<TR>
<TD>Nombre d'enfants</TD>
<TD><INPUT type="text" size="3" maxLength="3" name="txtEnfants" value="2"></TD>
</TR>
<TR>
<TD>Salaire annuel (euro)</TD>
<TD><INPUT type="text" maxLength="12" size="12" name="txtSalaire" value="60000"></TD>
</TR>
<TR>
<TD>Impôt à payer :
```

```

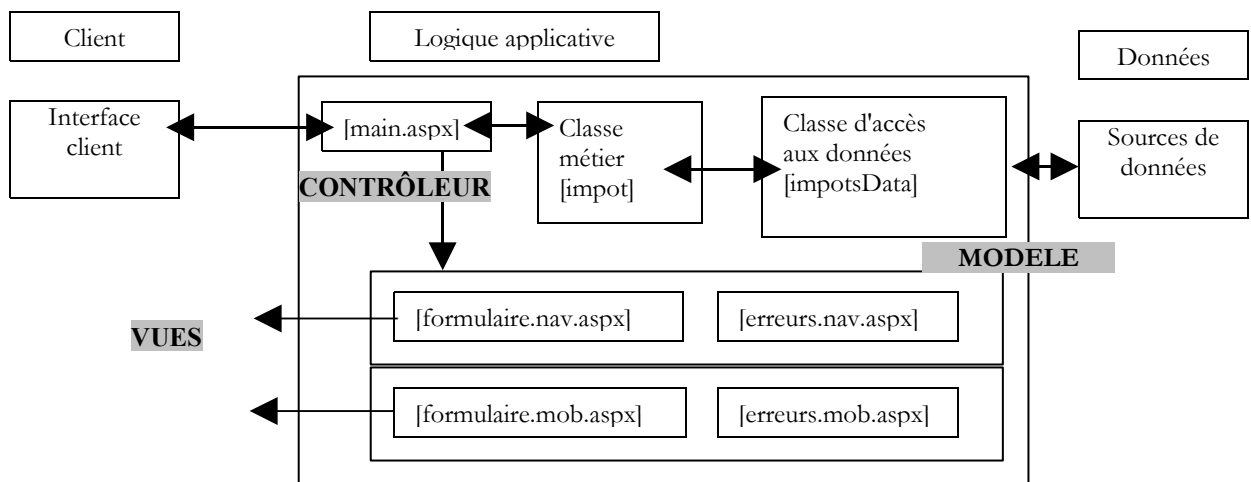
<TD>4300 euro(s)</TD>
</TR>
</TABLE>
<hr>
<P>
  <INPUT type="submit" value="Calculer">
</P>
</form>
<form method="post" action="main.aspx?action=effacer">
  <INPUT type="submit" value="Effacer">
</form>
</body>
</html>

```

Nous avons bien récupéré l'impôt à payer : 4300 euros. Nous retiendrons de cette illustration qu'il ne faut pas se laisser illusionner par le fait qu'on écrit une application web à destination de clients qui sont des navigateurs. Une application web est un service tcp-ip et ce protocole réseau ne permet pas de dire la nature de l'application cliente d'un service. Donc, on ne peut pas savoir si le client d'une application web est un navigateur ou non. On suit alors deux règles :

- à réception d'une requête d'un client, on ne fait aucune hypothèse sur le client et on vérifie que les paramètres attendus dans la requête sont bien présents et valides
- on élabore une réponse à destination des navigateurs, donc en général des documents HTML

Une application web peut être construite pour servir simultanément des clients différents, par exemple des navigateurs et des téléphones mobiles. On peut alors inclure dans chaque requête un nouveau paramètre indiquant le type du client. Ainsi un navigateur demandera le calcul de l'impôt par une requête à l'url `http://machine/impots/main.aspx?client=navigateur&action=calcul` alors que le téléphone mobile lui fera une requête à l'url `http://machine/impots/main.aspx?client=mobile&action=calcul`. La structure MVC facilite l'écriture d'une telle application. Elle devient la suivante :



Le bloc [Classes métier, Classes d'accès aux données] ne change pas. C'est en effet une partie indifférente au client. Le bloc [Contrôleur] change peu mais doit prendre un compte un nouveau paramètre dans la requête, le paramètre [client] indiquant à quel type de client il a affaire. Le bloc [vues] doit générer des vues pour chaque type de client. Il pourrait être intéressant de prendre en compte dès la conception de l'application la présence du paramètre [client] dans la requête, même si on n'a comme objectif à court ou moyen terme que les seuls navigateurs. Si l'application doit ultérieurement gérer un nouveau type de client, seules des vues adaptées à celui-ci sont à écrire.

5.2 Exemple 2

5.2.1 Le problème

Nous nous proposons ici de traiter le même problème que précédemment mais en modifiant la source des données de l'objet [impot] créé par l'application web. Dans la version précédente, celle utilisée délivrait les valeurs de tableaux écrits en "dur" dans le code. Cette fois-ci, la nouvelle source de données les prendra dans une source de données ODBC associée à une base MySQL.

5.2.2 La source de données ODBC

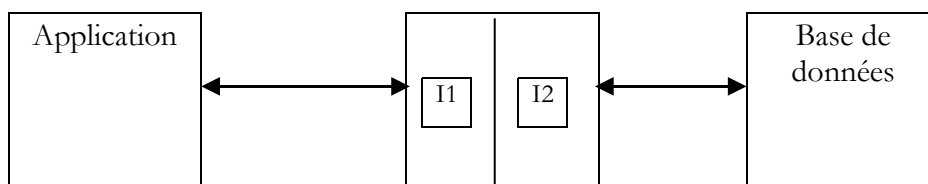
Les données se trouveront dans une table appelée [IMPOTS] d'une base MySQL appelée [dbimpots]. Le contenu de cette table sera le suivant :

limites	coeffr	coeffn
4262	0	0
8382	0.0683	291.09
14753	0.1914	1322.92
23888	0.2826	2668.39
38868	0.3738	4846.98
47932	0.4262	6883.66
0	0.4809	9505.54

Le propriétaire de la base est l'utilisateur [admimpots] de mot de passe [mdpimpots]. Nous associons une source de données ODBC à cette base. Avant de le faire, rappelons d'abord les différents moyens d'accéder à une base de données avec la plate-forme .NET.

Il existe de nombreuses bases de données pour les plate-formes windows. Pour y accéder, les applications passent au travers de programmes appelés **pilotes** (drivers).

Pilote de base de données



Dans le schéma ci-dessus, le pilote présente deux interfaces :

- l'interface I1 présentée à l'application
- l'interface I2 vers la base de données

Afin d'éviter qu'une application écrite pour une base de données B1 doive être réécrite si on migre vers une base de données B2 différente, un effort de normalisation a été fait sur l'interface I1. Si on utilise des bases de données utilisant des pilotes "normalisés", la base B1 sera fournie avec un pilote P1, la base B2 avec un pilote P2, et l'interface I1 de ces deux pilotes sera identique. Aussi n'aura-t-on pas à réécrire l'application. On pourra ainsi, par exemple, migrer une base de données ACCESS vers une base de données MySQL sans changer l'application.

Il existe deux types de pilotes normalisés :

- les pilotes ODBC (Open DataBase Connectivity)
- les pilotes OLE DB (Object Linking and Embedding DataBase)

Les pilotes ODBC permettent l'accès à des bases de données. Les sources de données pour les pilotes OLE DB sont plus variées : bases de données, messageries, annuaires, ... Il n'y a pas de limite. Toute source de données peut faire l'objet d'un pilote Ole DB si un éditeur le décide. L'intérêt est évidemment grand : on a un accès uniforme à une grande variété de données.

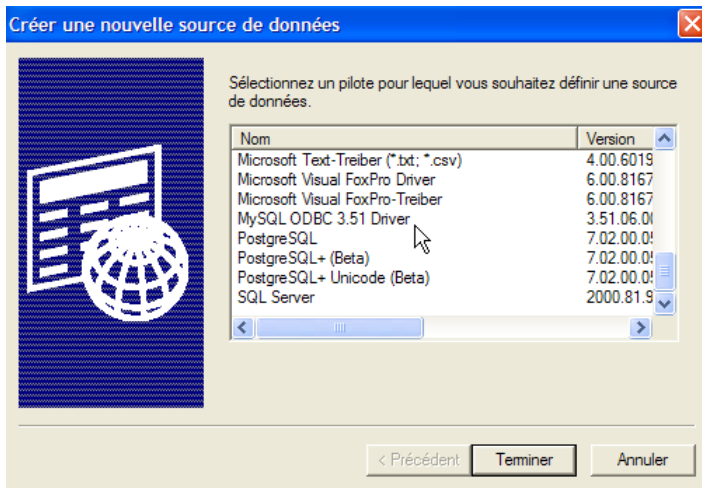
La plate-forme .NET 1.1 est livrée avec trois types de classes d'accès aux données :

1. les classes SQL Server.NET, pour accéder aux bases SQL Server de Microsoft
2. les classes Ole Db.NET, pour accéder aux bases des SGBD offrant un pilote OLE DB
3. les classes odbc.net, pour accéder aux bases des SGBD offrant un pilote ODBC

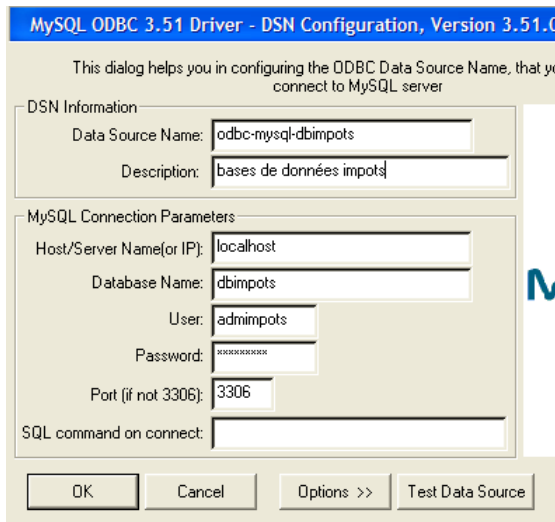
Le SGBD MySQL dispose depuis longtemps d'un pilote ODBC. C'est celui-ci que nous utilisons maintenant. Sous Windows, nous prenons l'option [Menu Démarrer/Panneau de configuration/Outils d'administration/Sources ODBC 32 bits]. Selon la version de windows, ce chemin peut légèrement varier. On obtient l'application suivante qui va nous permettre de créer notre source ODBC :



Nous allons créer une source de données Système, c.a.d. une source de données que tout utilisateur de la machine pourra utiliser. Aussi, ci-dessus sélectionnons-nous l'onglet [Source de données système]. La page présentée a un bouton [Ajouter] que nous utilisons pour créer une nouvelle source de données ODBC :



L'assistant demande de sélectionner le pilote ODBC à utiliser. Windows amène avec lui un certain nombre de pilotes ODBC pré-installés. Le pilote ODBC de MySQL ne fait pas partie du lot. Il faut donc auparavant l'installer. On le trouvera sur internet en tapant la chaîne clé "MySQL ODBC" ou encore "MyODBC" dans un moteur de recherche. Ici, nous avons installé le pilote [MySQL ODBC 3.51]. Nous le sélectionnons et faisons [Terminer] :

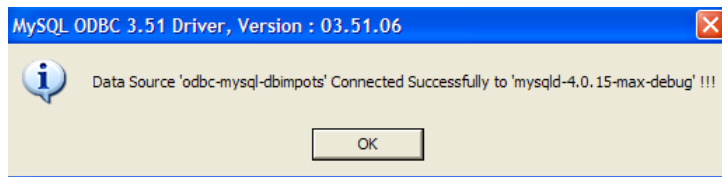


Un certain nombre de renseignements doit être fourni :

- Data Source Name** le nom qui désignera la source de données ODBC. Toute application windows pourra avoir accès à la source via ce nom
- Description** un texte arbitraire décrivant la source de données
- Host Name** le nom de la machine hébergeant le SGBD MySQL. Ici c'est la machine locale. Ce pourrait être une machine distante. Cela permettrait à une application windows d'accéder à une base distante sans aucun codage particulier. C'est un grand intérêt de la source ODBC.
- Database Name** un SGBD MySQL peut gérer plusieurs bases. Ici on précise laquelle on veut gérer : dbimpots
- User** nom d'un utilisateur déclaré au sein du SGBD MySQL. C'est sous son nom que se feront les accès à la

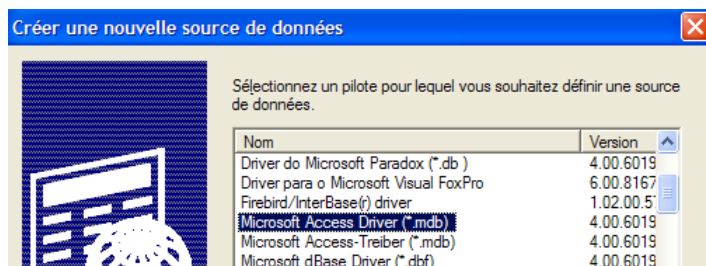
Password source de données. Ici : admimpots
 le mot de passe de cet utilisateur. Ici : mdpimpots
 Port port de travail du SGBD MySQL. Par défaut c'est le port 3306. Nous ne l'avons pas changé

Ceci fait, nous testons la validité de nos paramètres de connexion avec le bouton [Test Data Source] :

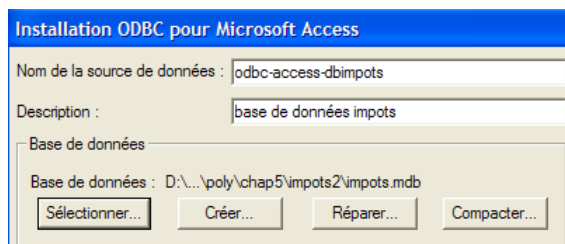


Ceci fait, nous sommes sûrs de notre source de données ODBC. Nous pouvons désormais l'exploiter. Nous faisons autant de fois que nécessaire [OK] pour sortir de l'assistant ODBC.

Si le lecteur ne dispose pas du SGBD MySQL, il peut se le procurer librement à l'url [http://www.mysql.com]. Nous présentons ci-dessous la démarche pour créer une source ODBC avec Access. Les premières étapes sont identiques à ce qui a été décrit précédemment. On ajoute une nouvelle source de données système :



Le pilote sélectionné sera [Microsoft Access Driver]. On fait [Terminer] pour passer à la définition de la source ODBC :

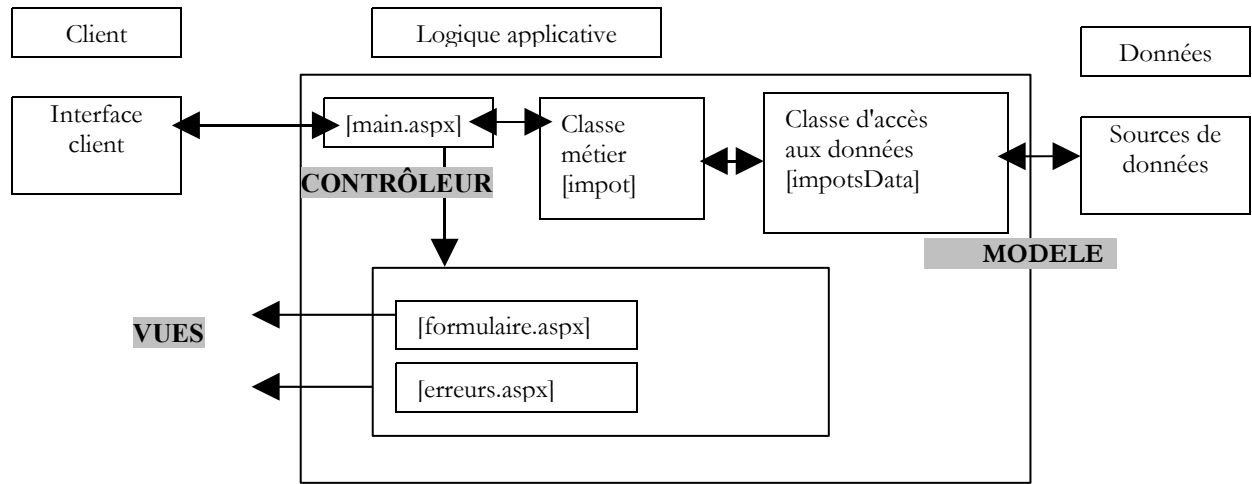


Les renseignements à fournir sont les suivants :

Nom de la source de données le nom qui désignera la source de données ODBC. Toute application windows pourra avoir accès à la source via ce nom
 Description un texte arbitraire décrivant la source de données
 Base de données le nom complet du fichier ACCESS à exploiter

5.2.3 Une nouvelle classe d'accès aux données

Revenons la structure MVC de notre application :



Sur le schéma ci-dessus, la classe [impotsData] est chargée de récupérer les données. Elle devra le faire ici auprès de la base MySQL [dbimpots]. Nous savons depuis la version précédente de cette application, que [impotsData] est une classe abstraite qu'il faut dériver à chaque fois qu'on veut l'adapter à une nouvelle source de données. Rappelons la structure de cette classe abstraite :

```

Imports System.Collections

Namespace st.istia.univangers.fr
Public MustInherit Class impotsData
Protected limites() As Decimal
Protected coeffr() As Decimal
Protected coeffn() As Decimal
Protected checked As Boolean
Protected valide As Boolean

' méthode d'accès aux données
Public MustOverride Function getData() As Object()

' méthode de vérification des données
Protected Function checkData() As Integer
' vérifie les données acquises
...
End Function

' vérifie la validité du contenu d'un tableau
Protected Function check(ByRef tableau() As Decimal, ByVal n As Integer) As Boolean
...
End Function
End Class
End Namespace
  
```

La classe qui dérive [impotsData] doit implémenter deux méthodes :

- un constructeur si le constructeur sans arguments de [impotsData] ne lui convient pas
- la méthode [getData] qui rend les trois tableaux (limites,coeffr,coeffn)

Nous créons la classe [impotsODBC] qui va aller chercher les données (limites,coeffr,coeffn) dans une source ODBC dont on lui donnera le nom :

```

Imports System.Data.Odbc
Imports System.Data
Imports System.Collections
Imports System

Namespace st.istia.univangers.fr
Public Class impotsODBC
Inherits impotsData

' variables d'instance
Protected DSNimpots As String

' constructeur
Public Sub New(ByVal DSNimpots As String)
' on note le nom de la source ODBC
Me.DSNimpots = DSNimpots
End Sub
  
```

```

Public Overrides Function getdata() As Object()
    ' initialise les trois tableaux limites, coeffr, coeffn à partir
    ' du contenu de la table [impots] de la base ODBC DSNimpots
    ' limites, coeffr, coeffn sont les trois colonnes de cette table
    ' peut lancer diverses exceptions

    Dim connectString As String = "DSN=" + DSNimpots + ";"      ' chaîne de connexion à la base
    Dim impotsConn As OdbcConnection = Nothing                ' la connexion
    Dim sqlCommand As OdbcCommand = Nothing                  ' la commande SQL
    ' la requête SELECT
    Dim selectCommand As String = "select limites,coeffr,coeffn from impots"
    ' tableaux pour récupérer les données
    Dim aLimites As New ArrayList
    Dim aCoeffR As New ArrayList
    Dim aCoeffN As New ArrayList
    Try
        ' on tente d'accéder à la base de données
        impotsConn = New OdbcConnection(connectString)
        impotsConn.Open()
        ' on crée un objet command
        sqlCommand = New OdbcCommand(selectCommand, impotsConn)
        ' on exécute la requête
        Dim myReader As OdbcDataReader = sqlCommand.ExecuteReader()
        ' Exploitation de la table récupérée
        While myReader.Read()
            ' les données de la ligne courante sont mis dans les tableaux
            aLimites.Add(myReader("limites"))
            aCoeffR.Add(myReader("coeffr"))
            aCoeffN.Add(myReader("coeffn"))
        End While
        ' libération des ressources
        myReader.Close()
    Catch e As Exception
        Throw New Exception("Erreur d'accès à la base de données (" + e.Message + ")")
    Finally
        impotsConn.Close()
    End Try
    ' les tableaux dynamiques sont mis dans des tableaux statiques
    Me.limites = New Decimal(aLimites.Count - 1) {}
    Me.coeffr = New Decimal(aCoeffR.Count - 1) {}
    Me.coeffn = New Decimal(aCoeffN.Count - 1) {}
    Dim i As Integer
    For i = 0 To aLimites.Count - 1
        limites(i) = Decimal.Parse(aLimites(i).ToString())
        coeffr(i) = Decimal.Parse(aCoeffR(i).ToString())
        coeffn(i) = Decimal.Parse(aCoeffN(i).ToString())
    Next i
    ' on vérifie les données acquises
    Dim erreur As Integer = checkData()
    ' si données pas valides, alors on lance une exception
    If Not valide Then Throw New Exception("Les données des tranches d'impôts sont invalides (" +
    erreur.ToString + ")")
    ' sinon on rend les trois tableaux
    Return New Object() {limites, coeffr, coeffn}
End Function
End Class
End Namespace

```

Intéressons-nous au constructeur :

```

' constructeur
Public Sub New(ByVal DSNimpots As String)
    ' on note le nom de la source ODBC
    Me.DSNimpots = DSNimpots
End Sub

```

Il reçoit en paramètre, le nom de la source ODBC dans laquelle se trouvent les données à acquérir. Le constructeur se contente de mémoriser ce nom. La méthode [getData] est chargée de lire les données de la table [impots] et de les mettre dans trois tableaux (limites, coeffr, coeffn). Commentons son code :

- les paramètres de la connexion à la source de données ODBC sont définis mais celle-ci n'est pas ouverte

```

' chaîne de connexion à la base
Dim connectString As String = "DSN=" + DSNimpots + ";"
' on crée un objet connexion à la base de données - cette connexion n'est pas ouverte
Dim impotsConn As OdbcConnection = New OdbcConnection(connectString)

```

- on définit trois objets [ArrayList] pour récupérer les données de la table [impots] :

```
' tableaux pour récupérer les données
Dim aLimites As New ArrayList
Dim aCoeffR As New ArrayList
Dim aCoeffN As New ArrayList
```

- Toute le code d'accès à la base est entouré d'un try/catch pour gérer une éventuelle erreur d'accès. On ouvre la connexion avec la base :

```
' on tente d'accéder à la base de données
impotsConn = New OdbcConnection(connectString)
impotsConn.Open()
```

- on exécute la commande [select] sur la connexion ouverte. On obtient un objet [OdbcDataReader] qui va nous permettre de parcourir les lignes de la table résultat du select :

```
' on crée un objet command
Dim sqlCommand As OdbcCommand = New OdbcCommand(selectCommand, impotsConn)
' on exécute la requête
Dim myReader As OdbcDataReader = sqlCommand.ExecuteReader()
```

- on parcourt la table résultat, ligne par ligne. Pour cela on utilise la méthode [Read] de l'objet [OdbcDataReader] obtenu précédemment. Cette méthode fait deux choses :
 - elle avance d'une ligne dans la table. Au départ, on est positionné avant la 1ère ligne
 - elle rend le booléen [true] si elle a pu avancer, [false] sinon, ce dernier cas indiquant que toutes les lignes ont été exploitées.

Les colonnes de la ligne courante de l'objet [OdbcDataReader] sont obtenues par [OdbcDataReader](nomColonne). On obtient alors un objet représentant la valeur de la colonne. Nous parcourons la totalité de la table pour mettre son contenu dans les trois objets [ArrayList] :

```
' Exploitation de la table récupérée
While myReader.Read()
' les données de la ligne courante sont mis dans les tableaux
aLimites.Add(myReader("limites"))
aCoeffR.Add(myReader("coeffr"))
aCoeffN.Add(myReader("coeffn"))
```

- ceci fait, nous libérons les ressources associées à la connexion :

```
' libération des ressources
myReader.Close()
impotsConn.Close()
```

- le contenu des trois objets [ArrayList] est transféré dans trois tableaux classiques :

```
' les tableaux dynamiques sont mis dans des tableaux statiques
limites = New Decimal(aLimites.Count - 1) {}
coeffr = New Decimal(aCoeffR.Count - 1) {}
coeffn = New Decimal(aCoeffN.Count - 1) {}
Dim i As Integer
For i = 0 To aLimites.Count - 1
limites(i) = CType(aLimites(i), Decimal)
coeffr(i) = CType(aCoeffR(i), Decimal)
coeffn(i) = CType(aCoeffN(i), Decimal)
Next i
```

- une fois les données de la table [impots] arrivées dans les trois tableaux, il ne reste plus qu'à vérifier le contenu de ceux-ci à l'aide de la méthode [checkData] de la classe de base [impotsData] :

```
' on vérifie les données acquises
Dim erreur As Integer = checkData()
' si données pas valides, alors on lance une exception
If Not valide Then Throw New Exception("Les données des tranches d'impôts sont invalides (" +
erreur.ToString + ")")
' sinon on rend les trois tableaux
Return New Object() {limites, coeffr, coeffn}
```

5.2.4 Tests de la classe d'accès aux données

Un programme de test pourrait être le suivant :

```

Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

Namespace st.istia.univangers.fr

' pg de test
Module testimpots
    Sub Main(ByVal arguments() As String)
        ' programme interactif de calcul d'impôt
        ' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
        ' le programme affiche alors l'impôt à payer
        Const syntaxe1 As String = "pg DSNimpots"
        Const syntaxe2 As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour
marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire
: salaire annuel en F"

        ' vérification des paramètres du programme
        If arguments.Length <> 1 Then
            ' msg d'erreur
            Console.Error.WriteLine(syntaxe1)
            ' fin
            Environment.Exit(1)
        End If
        ' on récupère les arguments
        Dim DSNimpots As String = arguments(0)

        ' création d'un objet impôt
        Dim objImpot As impot = Nothing
        Try
            objImpot = New impot(New impotsODBC(DSNimpots))
        Catch ex As Exception
            Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
            Environment.Exit(2)
        End Try

        ' boucle infinie
        While True
            ' au départ pas d'erreurs
            Dim erreur As Boolean = False

            ' on demande les paramètres du calcul de l'impôt
            Console.Out.Write("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour
arrêter :")
            Dim paramètres As String = Console.In.ReadLine().Trim()

            ' qq chose à faire ?
            If paramètres Is Nothing Or paramètres = "" Then
                Exit While
            End If

            ' vérification du nombre d'arguments dans la ligne saisie
            Dim args As String() = paramètres.Split(Nothing)
            Dim nbParamètres As Integer = args.Length
            If nbParamètres <> 3 Then
                Console.Error.WriteLine(syntaxe2)
                erreur = True
            End If
            Dim marié As String
            Dim nbEnfants As Integer
            Dim salaire As Integer
            If Not erreur Then
                ' vérification de la validité des paramètres
                ' marié
                marié = args(0).ToLower()
                If marié <> "o" And marié <> "n" Then
                    Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument marié incorrect : tapez o ou
n"))
                    erreur = True
                End If
                ' nbEnfants
                nbEnfants = 0
                Try
                    nbEnfants = Integer.Parse(args(1))
                    If nbEnfants < 0 Then
                        Throw New Exception
                    End If
                Catch
            End If
        End Try
    End Sub
End Module

```

```

        Console.Error.WriteLine(syntaxe2 + "\nArgument nbEnfants incorrect : tapez un entier positif
ou nul")
        erreur = True
    End Try
    ' salaire
    salaire = 0
    Try
        salaire = Integer.Parse(args(2))
        If salaire < 0 Then
            Throw New Exception
        End If
    Catch
        Console.Error.WriteLine(syntaxe2 + "\nArgument salaire incorrect : tapez un entier positif ou
nul")
        erreur = True
    End Try
    End If
    If Not erreur Then
        ' les paramètres sont corrects - on calcule l'impôt
        Console.Out.WriteLine(("impôt=" & objImpot.calculer(marié = "o", nbEnfants, salaire).ToString +
" euro(s)"))
    End If
    End While
End Sub
End Module
End Namespace

```

L'application est lancée avec un paramètre :

- DSNimpots : nom de la source de données ODBC à exploiter

Le calcul de l'impôt est fait à l'aide d'un objet de type [impot] créé dès le lancement de l'application :

```

' création d'un objet impôt
Dim objImpôt As impot = Nothing
Try
    objImpot = New impot(New impotsODBC(DSNimpots))
Catch ex As Exception
    Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
    Environment.Exit(1)
End Try

```

Une fois initialisée, l'application demande de façon répétée à l'utilisateur de taper au clavier les trois informations dont on a besoin pour calculer son impôt :

- son statut marital : o pour marié, n pour non marié
- son nombre d'enfants
- son salaire annuel

L'ensemble des classes est compilé :

```
dos>vbc /r:system.dll /r:system.data.dll /t:library /out:impot.dll impots.vb impotsArray.vb impotsData.vb
impotsODBC.vb
```

```
dos>dir
01/04/2004 19:34          7 168 impot.dll
01/04/2004 19:31          1 360 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
01/04/2004 19:34          2 735 impotsODBC.vb
01/04/2004 19:32          3 210 testimpots.vb
```

Le programme de test est compilé à son tour :

```
dos>vbc /r:impot.dll testimpots.vb
```

```
dir>dir
01/04/2004 19:34          7 168 impot.dll
01/04/2004 19:31          1 360 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
01/04/2004 19:34          2 735 impotsODBC.vb
01/04/2004 19:34          6 144 testimpots.exe
01/04/2004 19:32          3 210 testimpots.vb
```

Le programme de test est exécuté d'abord avec la source de données ODBC MySQL :

```
dos>testimpots odbc-mysql-dbimpots
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 60000
impôt=4300 euro(s)
```

On change de source ODBC pour prendre une source Access :

```
dos>testimpots odbc-access-dbimpots
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 60000
impôt=4300 euro(s)
```

5.2.5 Les vues de l'application web

Ce sont celles de l'application précédente : [formulaire.aspx] et [erreurs.aspx]

5.2.6 Les contrôleurs d'application [global.asax, main.aspx]

Seul le contrôleur [global.asax] doit être modifié. Il est en effet en charge de créer l'objet [impot] lors du démarrage de l'application. Le constructeur de cet objet a pour unique paramètre l'objet de type [impotsData] chargé de récupérer les données. Ce paramètre change donc pour chaque nouveau type de sources de données. Le contrôleur [global.asax.vb] devient le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' on crée un objet impot
        Dim objImpot As impot
        Try
            objImpot = New impot(New impotsODBC(ConfigurationSettings.AppSettings("DSNimpots")))
            ' on met l'objet dans l'application
            Application("objImpot") = objImpot
            ' pas d'erreur
            Application("erreur") = False
        Catch ex As Exception
            'il y a eu erreur, on le note dans l'application
            Application("erreur") = True
            Application("message") = ex.Message
        End Try
    End Sub
End Class
```

La source de données de l'objet [impot] est maintenant un objet [impotODBC]. Ce dernier a pour paramètre le nom DSN de la source de données ODBC à exploiter. Plutôt que d'écrire ce nom en dur dans le code, on le met dans le fichier de configuration [web.config] de l'application :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="DSNimpots" value="odbc-mysql-dbimpots" />
  </appSettings>
</configuration>
```

On sait que la valeur d'une clé C de la section <appSettings> du fichier [web.config] est obtenue dans le code de l'application par [ConfigurationSettings.AppSettings(C)].

Afin de connaître la cause de l'exception, on enregistre le message de celle-ci dans l'application afin qu'il reste disponible pour les requêtes. Le contrôleur [main.aspx.vb] inclura ce message dans sa liste d'erreurs :

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' avant tout, on regarde si l'application a pu s'initialiser correctement
    If CType(Application("erreur"), Boolean) Then
        ' on redirige vers la page d'erreurs
        Dim erreurs As New ArrayList
        erreurs.Add("Application momentanément indisponible..." + Application("message").ToString + ")")
        context.Items("erreurs") = erreurs
        context.Items("lien") = ""
        context.Items("href") = ""
        Server.Transfer("erreurs.aspx")
    End If
    ' on récupère l'action à faire
```

5.2.7 Bilan des modifications

L'application est prête à être testée. Listons les modifications amenées à la version précédente :

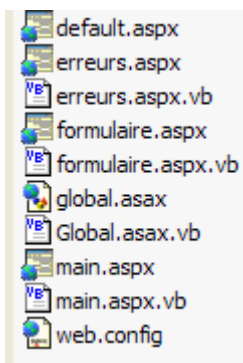
1. une nouvelle classe d'accès aux données a été construite
2. le contrôleur [global.asax.vb] a été modifié en deux endroits : construction de l'objet [impot] et enregistrement dans l'application du message lié à l'éventuelle exception
3. le contrôleur [main.aspx.vb] a été modifié en un endroit pour afficher le message d'exception précédent
4. un fichier [web.config] a été ajouté

Le travail de modification s'est fait essentiellement en 1, c.a.d. en-dehors de l'application web. Ceci a été rendu possible grâce à l'architecture MVC de l'application qui sépare le contrôleur des classes métier. C'est là tout l'intérêt de cette architecture. On pourrait montrer qu'avec un fichier de configuration [web.config] adéquat, on aurait pu éviter toute modification du contrôleur de l'application. Il est possible de mettre dans [web.config] le nom de la classe d'accès aux données à instancier dynamiquement ainsi que les divers paramètres nécessaires à cette instanciation. Avec ces informations, [global.asax] peut instancier l'objet d'accès aux données. Changer de source de données revient alors à :

- créer la classe d'accès à cette source si elle n'existe pas encore
- modifier le fichier [web.config] pour permettre la création dynamique d'une instance de cette classe dans [global.asax]

5.2.8 Test de l'application web

L'ensemble des fichiers précédents sont placés dans un dossier <application-path>.

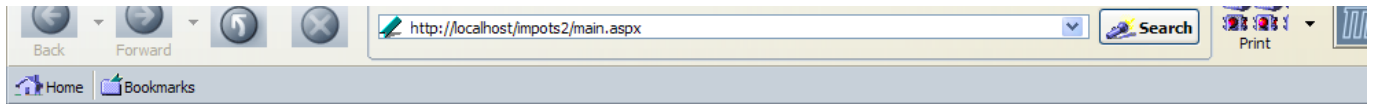


Dans ce dossier, est créé un sous-dossier [bin] dans lequel est placé l'assemblage [impot.dll] issu de la compilation des fichiers des classes métier : [impots.vb, impotsData.vb, impotsArray.vb, impotsODBC.vb]. On rappelle ci-dessous, la commande de compilation nécessaire :

```
dos>vbc /r:system.dll /r:system.data.dll /t:library /out:impot.dll impots.vb impotsArray.vb impotsData.vb impotsODBC.vb
```

```
dos>dir
01/04/2004 19:34          7 168 impot.dll
01/04/2004 19:31          1 360 impots.vb
21/04/2004 08:23          1 311 impotsArray.vb
21/04/2004 08:26          1 634 impotsData.vb
01/04/2004 19:34          2 735 impotsODBC.vb
01/04/2004 19:32          3 210 testimpots.vb
```

Le fichier [impot.dll] ci-dessus doit être placé dans <application-path>\bin afin que l'application web y ait accès. Le serveur Cassini est lancé avec les paramètres (<application-path>, /impots2). Les tests donnent les mêmes résultats que dans la version précédente, la présence de la base de données étant transparente pour l'utilisateur. Pour illustrer néanmoins cette présence, nous faisons en sorte que la source ODBC ne soit pas disponible en arrêtant le SGBD MySQL et nous demandons l'url [http://localhost/impots2/main.aspx]. Nous obtenons la réponse suivante :



Les erreurs suivantes se sont produites :

- Application momentanément indisponible...(Erreur d'accès à la base de données (ERROR [HY000] [MySQL][ODBC 3.51 Driver]Can't connect to MySQL server on 'localhost' (10061) ERROR [HY000] [MySQL][ODBC 3.51 Driver]Can't connect to MySQL server on 'localhost' (10061)))

5.3 Exemple 3

5.3.1 Le problème

Nous nous proposons ici de traiter le même problème en modifiant de nouveau la source des données de l'objet [impot] créé par l'application web. Cette fois-ci, la nouvelle source de données sera une base ACCESS à laquelle on accèdera via un pilote OLEDB. Notre objectif est de montrer une autre façon d'accéder à une base de données.

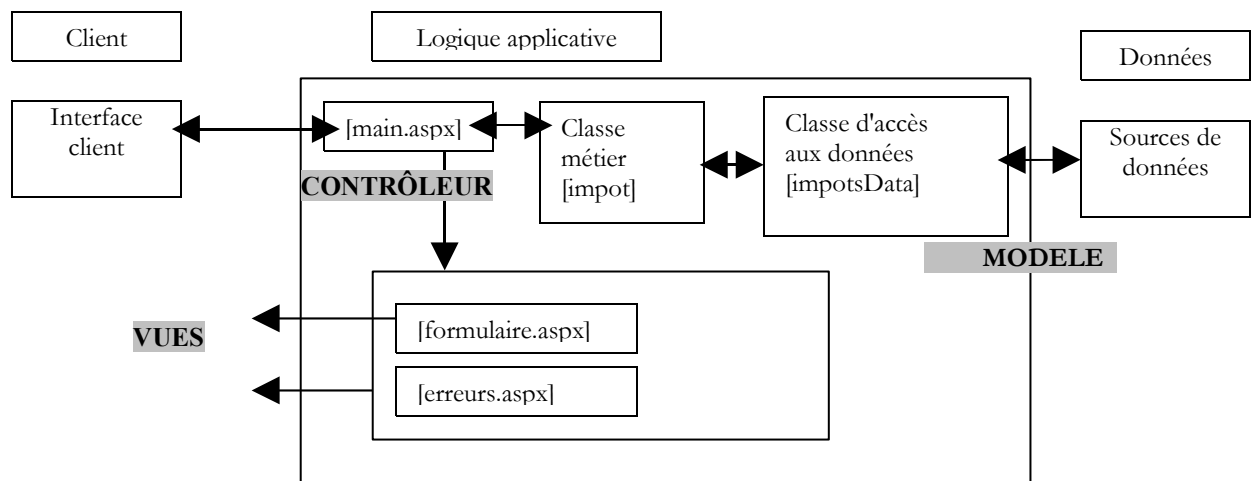
5.3.2 La source de données OLEDB

Les données se trouveront dans une table appelée [IMPOTS] d'une base ACCESS. Le contenu de cette table sera le suivant :

limites	coeffr	coeffn
4262	0	0
8382	0,0683	291,09
14753	0,1914	1322,92
23888	0,2826	2668,39
38868	0,3738	4846,98
47932	0,4262	6883,66
0	0,4809	9505,54

5.3.3 La classe d'accès aux données

Revenons la structure MVC de notre application :



Sur le schéma ci-dessus, la classe [impotsData] est chargée de récupérer les données. Elle devra le faire cette fois-ci auprès d'une source OLEDB.

Nous créons la classe [impotsOLEDB] qui va aller chercher les données (limites,coeffr,coeffn) dans une source ODBC dont on lui donnera le nom :

```
Imports System.Data
Imports System.Collections
Imports System
Imports System.Xml
Imports System.Data.OleDb
```



```

Namespace st.istia.univangers.fr
Public Class impotsOLEDB
    Inherits impotsData

    ' variables d'instance
    Protected chaineConnexion As String

    ' constructeur
    Public Sub New(ByVal chaineConnexion As String)
        ' on note la chaîne de connexion au SGBD
        Me.chaineConnexion = chaineConnexion
    End Sub

    Public Overrides Function getData() As Object()
        ' initialise les trois tableaux limites, coeffr, coeffn à partir
        ' du contenu de la table [impots] de la base OLEDB [chaineConnexion]
        ' limites, coeffr, coeffn sont les trois colonnes de cette table
        ' peut lancer diverses exceptions

        ' on crée un objet DataAdapter pour lire les données de la source OLEDB
        Dim adaptateur As New OleDbDataAdapter("select limites,coeffr,coeffn from impots", chaineConnexion)
        ' on crée une image en mémoire du résultat du select
        Dim contenu As New DataTable("impots")
        Try
            adaptateur.Fill(contenu)
        Catch e As Exception
            Throw New Exception("Erreur d'accès à la base de données (" + e.Message + ")")
        End Try
        ' on récupère le contenu de la table impots
        Dim lignesImpots As DataRowCollection = contenu.Rows
        ' on dimensionne les tableaux de réception
        Me.limites = New Decimal(lignesImpots.Count - 1) {}
        Me.coeffr = New Decimal(lignesImpots.Count - 1) {}
        Me.coeffn = New Decimal(lignesImpots.Count - 1) {}
        ' on transfère le contenu de la table impots dans les tableaux
        Dim i As Integer
        Dim ligne As DataRow
        Try
            For i = 0 To lignesImpots.Count - 1
                ' ligne i de la table
                ligne = lignesImpots.Item(i)
                ' on récupère le contenu de la ligne
                limites(i) = CType(ligne.Item(0), Decimal)
                coeffr(i) = CType(ligne.Item(1), Decimal)
                coeffn(i) = CType(ligne.Item(2), Decimal)
            Next
        Catch
            Throw New Exception("Certaines données des tranches d'impôts sont invalides")
        End Try
        ' on vérifie les données acquises
        Dim erreur As Integer = checkData()
        ' si données pas valides, alors on lance une exception
        If Not valide Then Throw New Exception("Les données des tranches d'impôts sont invalides (" +
            erreur.ToString + ")")
        ' sinon on rend les trois tableaux
        Return New Object() {limites, coeffr, coeffn}
    End Function
End Class
End Namespace

```

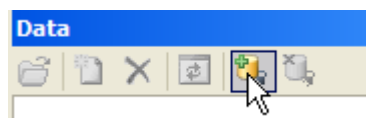
Intéressons-nous au constructeur :

```

' constructeur
Public Sub New(ByVal chaineConnexion As String)
    ' on note la chaîne de connexion au SGBD
    Me.chaineConnexion = chaineConnexion
End Sub

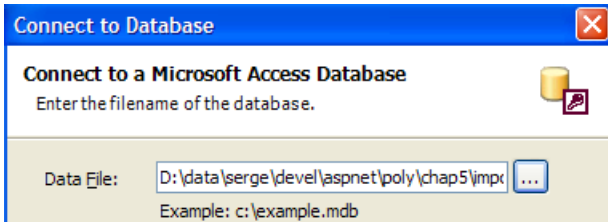
```

Il reçoit en paramètre, la chaîne de connexion de la source OLEDB dans laquelle se trouvent les données à acquérir. Le constructeur se contente de la mémoriser. Une chaîne de connexion contient tous les paramètres nécessaires au pilote OLEDB pour se connecter à la source OLEDB. Elle est en général assez complexe. Pour découvrir celle des bases ACCESS, on peut se faire aider par l'outil [WebMatrix]. On lance cet outil. Il offre une fenêtre permettant de se connecter à une source de données :

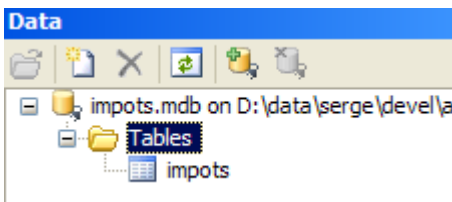




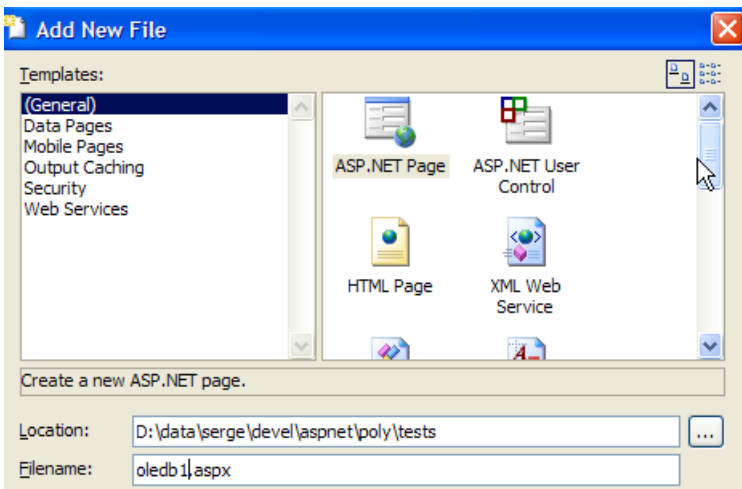
Grâce à l'icône désignée par la flèche ci-dessus, il est possible de créer une connexion à deux types de bases de données Microsoft : SQL Server et ACCESS. Choisissons ACCESS :



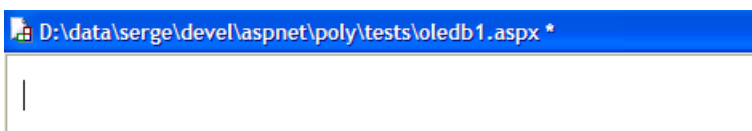
Nous avons utilisé le bouton [...] pour désigner la base ACCESS. Nous validons l'assistant. Dans l'onglet [Data], des icônes symbolisent la connexion :



Maintenant, construisons un nouveau fichier .aspx par [Files/New File] :



Nous obtenons une feuille vierge sur laquelle on peut dessiner notre interface web :



Faisons glisser de l'onglet [Data] la table [impots] sur la feuille ci-dessus. Nous obtenons le résultat suivant :

D:\data\serge\devel\aspnet\poly\tests\oledb1.aspx *

N°	limites	coeffr	coeffn
0	0	0	0
1	0,1	0,1	0,1
2	0,2	0,2	0,2
3	0,3	0,3	0,3
4	0,4	0,4	0,4
			1

AccessDataSourceControl - AccessDataSourceControl1

Cliquons droit sur l'objet [AccessDataSourceControl] ci-dessous pour avoir accès à ces propriétés :

Properties

AccessDataSourceControl1 Microsoft

(Click to see parent HTML elements)

Comportement

 EnableViewState True

Divers

 (ID) AccessDataSourceCo

Données

 ConnectionString Provider=Microsoft.J

 SelectCommand SELECT * FROM [impots]

La chaîne de connexion OLEDB à la base ACCESS est donnée par la propriété [ConnectionString] ci-dessus :

```
Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data Source=D:\data\serge\devel\aspnet\poly\chap5\impots\3\impots.mdb
```

On voit que cette chaîne est formée d'une partie fixe et d'une partie variable qui est tout simplement le nom du fichier ACCESS. Nous utiliserons ce fait pour générer la chaîne de connexion à notre source de données OLEDB.

Revenons maintenant à notre classe [impotsOLEDB]. La méthode [getData] est chargée de lire les données de la table [impots] et de les mettre dans trois tableaux (limites, coeffr, coeffn). Commentons son code :

- on définit l'objet [DataAdapter] qui va nous permettre de transférer en mémoire le résultat d'une requête SQL [select]. Pour cela, nous définissons la requête [select] à exécuter et l'associons à l'objet [DataAdapter]. Le constructeur de celui-ci réclame également la chaîne de connexion qu'il devra utiliser pour se connecter à la source OLEDB

```
' on crée un objet DataAdapter pour lire les données de la source OLEDB
Dim adaptateur As New OleDbDataAdapter("select limites,coeffr,coeffn from impots", chaineConnexion)
```

- on exécute la commande [select] grâce à la méthode [Fill] de l'objet [DataAdapter]. Le résultat du [select] est injecté dans un objet [DataTable] créé pour l'occasion. Un objet [DataTable] est l'image en mémoire d'une table de base de données, c.a.d. un ensemble de lignes et de colonnes. Nous gérons une exception qui peut survenir si par exemple la chaîne de connexion est incorrecte.

```
' on crée une image en mémoire du résultat du select
Dim contenu As New DataTable("impots")
Try
    adaptateur.Fill(contenu)
Catch e As Exception
    Throw New Exception("Erreur d'accès à la base de données (" + e.Message + ")")
End Try
```

- dans [contenu], nous avons la table [impots] ramenée par le [select]. Un objet [DataTable] est une table donc un ensemble de lignes. Celles-ci sont accessibles via la propriété [rows] de [DataTable] :

```
' on récupère le contenu de la table impots
Dim lignesImpots As DataRowCollection = contenu.Rows
```

- chaque élément de la collection [lignesImpots] est un objet de type [DataRow] représentant une ligne de la table. Celle-ci a des colonnes accessibles via l'objet [DataRow] via sa propriété [Item]. [DataRow].[Item(i)] est la colonne n° i de la ligne

[DataRow]. En parcourant la collection des lignes (la collection DataRow de lignesImpots), et la collection des colonnes de chaque ligne, on est capable d'obtenir la totalité de la table :

```
' on dimensionne les tableaux de réception
Me.limite = New Decimal(lignesImpots.Count - 1) {}
Me.coeffr = New Decimal(lignesImpots.Count - 1) {}
Me.coeffn = New Decimal(lignesImpots.Count - 1) {}
' on transfère le contenu de la table impots dans les tableaux
Dim i As Integer
Dim ligne As DataRow
Try
    For i = 0 To lignesImpots.Count - 1
        ' ligne i de la table
        ligne = lignesImpots.Item(i)
        ' on récupère le contenu de la ligne
        limite(i) = CType(ligne.Item(0), Decimal)
        coeffr(i) = CType(ligne.Item(1), Decimal)
        coeffn(i) = CType(ligne.Item(2), Decimal)
    Next
Catch
    Throw New Exception("Les données des tranches d'impôts n'ont pas le bon type")
End Try
```

- une fois les données de la table [impots] arrivées dans les trois tableaux, il ne reste plus qu'à vérifier le contenu de ceux-ci à l'aide de la méthode [checkData] de la classe de base [impotsData] :

```
' on vérifie les données acquises
Dim erreur As Integer = checkData()
' si données pas valides, alors on lance une exception
If Not valide Then Throw New Exception("Les données des tranches d'impôts sont invalides (" +
erreur.ToString + ")")
' sinon on rend les trois tableaux
Return New Object() {limite, coeffr, coeffn}
```

5.3.4 Tests de la classe d'accès aux données

Un programme de test pourrait être le suivant :

```
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

Namespace st.istia.univangers.fr

' pg de test
Module testimpots
    Sub Main(ByVal arguments() As String)
        ' programme interactif de calcul d'impôt
        ' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
        ' le programme affiche alors l'impôt à payer

        Const syntaxe1 As String = "pg bdACCESS"

        Const syntaxe2 As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour
marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire
: salaire annuel en F"

        ' vérification des paramètres du programme
        If arguments.Length <> 1 Then
            ' msg d'erreur
            Console.Error.WriteLine(syntaxe1)
            ' fin
            Environment.Exit(1)
        End If
        ' on récupère les arguments
        Dim chemin As String = arguments(0)

        ' on prépare la chaîne de connexion
        Dim chaineConnexion As String = "Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data Source="
+ chemin

        ' création d'un objet impôt
        Dim objImpot As impot = Nothing
        Try
            objImpot = New impot(New impotsOLEDB(chaineConnexion))
        Catch ex As Exception
```

```

Console.Error.WriteLine("L'erreur suivante s'est produite : " + ex.Message)
Environment.Exit(2)
End Try

' boucle infinie
While True
' au départ pas d'erreurs
Dim erreur As Boolean = False

' on demande les paramètres du calcul de l'impôt
Console.Out.WriteLine("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour
arrêter :")
Dim paramètres As String = Console.In.ReadLine().Trim()

' qq chose à faire ?
If paramètres Is Nothing Or paramètres = "" Then
Exit While
End If

' vérification du nombre d'arguments dans la ligne saisie
Dim args As String() = paramètres.Split(Nothing)
Dim nbParamètres As Integer = args.Length
If nbParamètres <> 3 Then
Console.Error.WriteLine(syntaxe2)
erreur = True
End If
Dim marié As String
Dim nbEnfants As Integer
Dim salaire As Integer
If Not erreur Then
' vérification de la validité des paramètres
' marié
marié = args(0).ToLower()
If marié <> "o" And marié <> "n" Then
Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument marié incorrect : tapez o ou
n"))
erreur = True
End If
' nbEnfants
nbEnfants = 0
Try
nbEnfants = Integer.Parse(args(1))
If nbEnfants < 0 Then
Throw New Exception
End If
Catch
Console.Error.WriteLine(syntaxe2 + "\nArgument nbEnfants incorrect : tapez un entier positif
ou nul")
erreur = True
End Try
' salaire
salaire = 0
Try
salaire = Integer.Parse(args(2))
If salaire < 0 Then
Throw New Exception
End If
Catch
Console.Error.WriteLine(syntaxe2 + "\nArgument salaire incorrect : tapez un entier positif ou
nul")
erreur = True
End Try
End If
If Not erreur Then
' les paramètres sont corrects - on calcule l'impôt
Console.Out.WriteLine("impôt=" & objImpot.calculer(marié = "o", nbEnfants, salaire).ToString +
" euro(s)")
End If
End While
End Sub
End Module
End Namespace

```

L'application est lancée avec un paramètre :

- bdACCESS : nom du fichier ACCESS à exploiter

Le calcul de l'impôt est fait à l'aide d'un objet de type [impot] créé dès le lancement de l'application :

```
| ' on récupère les arguments |
```

Exemples

157/192

```

Dim chemin As String = arguments(0)
' on prépare la chaîne de connexion
Dim chaineConnexion As String = "Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data Source="
+ chemin

' création d'un objet impôt
Dim objImpot As impot = Nothing
Try
    objImpot = New impot(New impotsOLEDB(chaineConnexion))
Catch ex As Exception
    Console.Error.WriteLine("L'erreur suivante s'est produite : " + ex.Message)
    Environment.Exit(2)
End Try

```

La chaîne de connexion à la source OLEDB a été construite à partir des informations obtenues avec [WebMatrix].

Une fois initialisée, l'application demande de façon répétée à l'utilisateur de taper au clavier les trois informations dont on a besoin pour calculer son impôt :

- son statut marital : o pour marié, n pour non marié
- son nombre d'enfants
- son salaire annuel

L'ensemble des classes est compilé :

```

dos>vbc /r:system.dll /r:system.data.dll /t:library /out:impot.dll impots.vb impotsArray.vb impotsData.vb
impotsOLEDB.vb

```

```

dos>vbc /r:impot.dll testimpots.vb

```

Le fichier [impots.mdb] est placé dans le dossier de l'application de test et celle-ci est lancée de la façon suivante :

```

dos>testimpots impots.mdb
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 60000
impôt=4300 euro(s)

```

On peut lancer l'application avec un fichier ACCESS incorrect :

```

dos>testimpots xx
L'erreur suivante s'est produite : Erreur d'accès à la base de données (Fichier
'D:\data\serge\devel\aspnet\poly\chap5\impots\3\xx' introuvable.)

```

5.3.5 Les vues de l'application web

Ce sont celles de l'application précédente : [formulaire.aspx] et [erreurs.aspx]

5.3.6 Les contrôleurs d'application [global.asax, main.aspx]

Seul le contrôleur [global.asax] doit être modifié. Il a en effet en charge de créer l'objet [impot] lors du démarrage de l'application. Le constructeur de cet objet a pour unique paramètre l'objet de type [impotsData] chargé de récupérer les données. Ce paramètre change donc puisqu'on change de sources de données. Le contrôleur [global.asax.vb] devient le suivant :

```

Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration

Public Class Global
    Inherits System.Web.HttpApplication

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' on crée un objet impot
        Dim objImpot As impot
        Try
            objImpot = New impot(New impotsOLEDB(ConfigurationSettings.AppSettings("chaineConnexion")))
            ' on met l'objet dans l'application
            Application("objImpot") = objImpot
            ' pas d'erreur
            Application("erreur") = False
        Catch ex As Exception
            ' il y a eu erreur, on le note dans l'application
            Application("erreur") = True
            Application("message") = ex.Message
        End Try
    End Sub
End Class

```

```
End Try
End Sub
End Class
```

La source de données de l'objet [impot] est maintenant un objet [impotOLEDB]. Ce dernier a pour paramètre la chaîne de connexion de la source de données OLEDB à exploiter. Celle-ci est placée dans le fichier de configuration [web.config] de l'application :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="chaîneConnexion"
      value="Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data
Source=D:\data\serge\devel\aspnet\poly\chap5\impots2\impots.mdb" />
  </appSettings>
</configuration>
```

Le contrôleur [main.aspx] ne change pas.

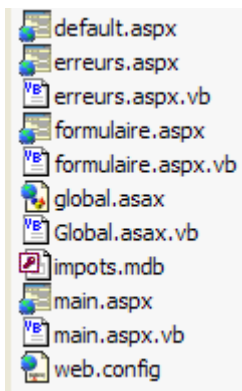
5.3.7 Bilan des modifications

L'application est prête à être testée. Listons les modifications amenées à la version précédente :

1. une nouvelle classe d'accès aux données a été construite
2. le contrôleur [global.asax.vb] a été modifié en un endroit : construction de l'objet [impot]
3. un fichier [web.config] a été ajouté

5.3.8 Test de l'application web

L'ensemble des fichiers précédents est placé dans un dossier <application-path>.



Dans ce dossier, est créé un sous-dossier [bin] dans lequel est placé l'assemblage [impot.dll] issu de la compilation des fichiers des classes métier : [impots.vb, impotsData.vb, impotsArray.vb, impotsOLEDB.vb]. On rappelle ci-dessous, la commande de compilation nécessaire :

```
dos>vbc /r:system.dll /r:system.data.dll /t:library /out:impot.dll impots.vb impotsArray.vb impotsData.vb
impotsOLEDB.vb
```

Le fichier [impot.dll] produit par cette commande doit être placé dans <application-path>\bin afin que l'application web y ait accès. Le serveur Cassini est lancé avec les paramètres (<application-path>, /impots3). Les tests donnent les mêmes résultats que dans la version précédente.

5.4 Exemple 4

5.4.1 Le problème

Nous nous proposons maintenant de transformer notre application en application de simulation de calculs d'impôt. Un utilisateur pourra faire des calculs successifs d'impôts et ceux-ci lui seront présentés sur une nouvelle vue ressemblant à ceci :

http://localhost/impots4/main.aspx?action=calcul

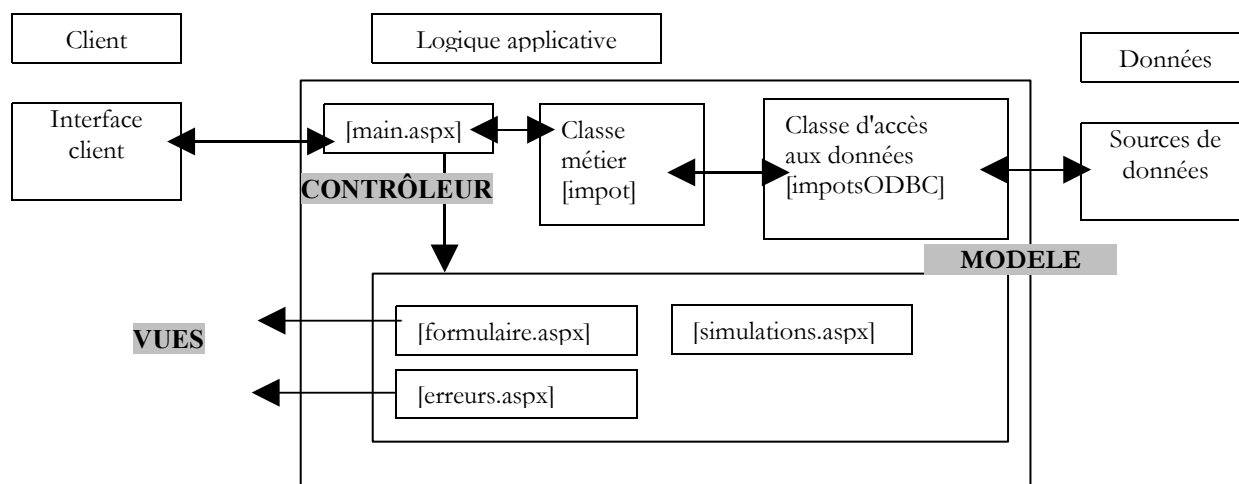
Résultats des simulations

	Marié	Enfants	Salaire annuel (euro)	Impôt à payer (euro)
oui	2	60000	4300	
non	3	60000	4300	

[Retour au formulaire](#)

5.4.2 La structure MVC de l'application

La structure MVC de l'application devient la suivante :



Apparaît une nouvelle vue [simulations.aspx] dont nous venons de donner une copie d'écran. La classe d'accès aux données sera la classe [impotsODBC] de l'exemple 2.

5.4.3 Les vues de l'application web

La vue [erreurs.aspx] ne change pas. La vue [formulaire.aspx] change légèrement. En effet, le montant de l'impôt n'apparaît désormais plus sur cette vue. Il est maintenant sur la vue [simulations.aspx]. Ainsi au démarrage, la page présentée à l'utilisateur est la suivante :

Calcul de votre impôt

Etes-vous marié(e) Oui Non

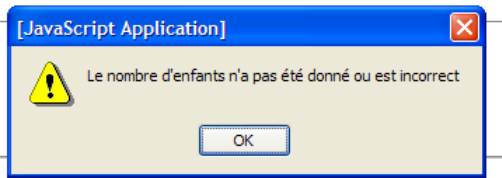
Nombre d'enfants

Salaire annuel (euro)

Par ailleurs, la vue [formulaire] transporte avec elle un script javascript qui vérifie la validité des données saisies avant de les envoyer au serveur comme le montre l'exemple suivant :

Calcul de votre impôt

Etes-vous marié(e) Oui Non
 Nombre d'enfants
 Salaire annuel (euro)



Le code de présentation est le suivant :

```

<%@ page src="formulaire.aspx.vb" inherits="formulaire" AutoEventWireup="false"%>
<html>
<head>
<title>Impôt</title>
<script language="javascript">
function calculer() {
    // vérification des paramètres avant de les envoyer au serveur
    with(document.frmImpots) {
        //nbre d'enfants
        champs=/^\s*(\d+)\s*$/ .exec(txtEnfants.value);
        if(champs==null) {
            // le modèle n'est pas vérifié
            alert("Le nombre d'enfants n'a pas été donné ou est incorrect");
            txtEnfants.focus();
            return;
        } //if
        //salaire
        champs=/^\s*(\d+)\s*$/ .exec(txtSalaire.value);
        if(champs==null) {
            // le modèle n'est pas vérifié
            alert("Le salaire n'a pas été donné ou est incorrect");
            txtSalaire.focus();
            return;
        } //if
        // c'est bon - on envoie le formulaire au serveur
        submit();
    } //with
} //calculer
</script>
</head>
<body>
<P>Calcul de votre impôt</P>
<HR width="100%" SIZE="1">
<form name="frmImpots" method="post" action="main.aspx?action=calcul">
<TABLE border="0">
<TR>
<TD>Etes-vous marié(e)</TD>
<TD>
<INPUT type="radio" value="oui" name="rdMarie" <%=rdouichecked%>>Oui
<INPUT type="radio" value="non" name="rdMarie" <%=rdnonchecked%>>Non</TD>
</TR>
<TR>
<TD>Nombre d'enfants</TD>
<TD><INPUT type="text" size="3" maxLength="3" name="txtEnfants" value="<%=txtEnfants%>"></TD>
</TR>
<TR>
<TD>Salaire annuel (euro)</TD>
<TD><INPUT type="text" maxLength="12" size="12" name="txtSalaire" value="<%=txtSalaire%>"></TD>
</TR>
</TABLE>
<hr>
<P>
<INPUT type="button" value="Calculer" onclick="calculer()">
</P>
</form>
<form method="post" action="main.aspx?action=effacer">
<INPUT type="submit" value="Effacer">
</form>
</body>
</html>
    
```

Les champs dynamiques de la page sont ceux des versions précédentes. Le champ dynamique du montant de l'impôt a disparu. Le bouton [Calculer] n'est plus un bouton de type [submit]. Il est de type [button] et lorsqu'il est cliqué, la fonction javascript [calculer()] est exécutée :

```
<INPUT type="button" value="Calculer" onclick="calculer()">
```

On a donné au formulaire un nom [frmImpots] afin de pouvoir le référencer dans le script [calculer] :

```
<form name="frmImpots" method="post" action="main.aspx?action=calcul">
```

La fonction javascript [calculer] utilise des expressions régulières pour vérifier la validité des champs du formulaire [document.frmImpots.txtEnfants] et [document.frmImpots.txtSalaire]. Si les valeurs saisies sont correctes, elles sont envoyées au serveur par [document.frmImpots.submit()].

La page de présentation obtient ses champs dynamiques auprès de son contrôleur [formulaire.aspx.vb] suivant :

```
Imports System.Collections.Specialized

Public Class formulaire
    Inherits System.Web.UI.Page

    ' champs de la page
    Protected rdouichecked As String
    Protected rdnonchecked As String
    Protected txtEnfants As String
    Protected txtSalaire As String
    Protected txtImpot As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' on récupère la précédente requête dans le contexte
        Dim form As NameValueCollection = Context.Items("formulaire")
        ' on prépare la page à afficher
        ' boutons radio
        rdouichecked = ""
        rdnonchecked = "checked"
        If form("rdMarie").ToString = "oui" Then
            rdouichecked = "checked"
            rdnonchecked = ""
        End If
        ' le reste
        txtEnfants = CType(form("txtEnfants"), String)
        txtSalaire = CType(form("txtSalaire"), String)
    End Sub
End Class
```

Le contrôleur [formulaire.aspx.vb] est identique aux versions précédentes si ce n'est qu'il n'a plus à récupérer le champ [txtImpot] dans le contexte, ce champ ayant disparu de la page.

La vue [simulations.aspx] se présente visuellement comme suit :

Résultats des simulations

Marié Enfants Salaire annuel (euro) Impôt à payer (euro)

oui	2	60000	4300
non	3	60000	4300
oui	4	60000	2315

[Retour au formulaire](#)

et correspond au code de présentation suivant :

```
<%@ page src="simulations.aspx.vb" inherits="simulations" autoeventwireup="false" %>
<HTML>
<HEAD>
<title>simulations</title>
</HEAD>
<body>
<P>Résultats des simulations</P>
<HR width="100%" SIZE="1">
<table>
<tr>
<th>
Marié</th>
```

```

        <th>
            Enfants</th>
        <th>
            Salaire annuel (euro)</th>
        <th>
            Impôt à payer (euro)</th>
    </tr>
    <%=simulationsHTML%>
</table>
<p></p>
<a href="<%=href%>">
    <%=lien%>
</a>
</body>
</HTML>

```

Ce code présente trois champs dynamiques :

<code>simulationsHTML</code>	code HTML d'une liste de simulations sous la forme de lignes de table HTML
<code>href</code>	url d'un lien
<code>lien</code>	texte du lien

Ils sont générés par la partie contrôleur [simulations.aspx.vb] :

```

Imports System.Collections
Imports Microsoft.VisualBasic

Public Class simulations
    Inherits System.Web.UI.Page

    Protected simulationsHTML As String = ""
    Protected href As String
    Protected lien As String

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'on récupère les simulations dans le contexte
        Dim simulations As ArrayList = CType(context.Items("simulations"), ArrayList)
        'chaque simulation est un tableau de 4 éléments string
        Dim simulation() As String
        Dim i, j As Integer
        For i = 0 To simulations.Count - 1
            simulation = CType(simulations(i), String())
            simulationsHTML += "<tr>"
            For j = 0 To simulation.Length - 1
                simulationsHTML += "<td>" + simulation(j) + "</td>"
            Next
            simulationsHTML += "</tr>" + ControlChars.CrLf
        Next
        'on récupère les autres éléments du contexte
        href = context.Items("href").ToString
        lien = context.Items("lien").ToString
    End Sub
End Class

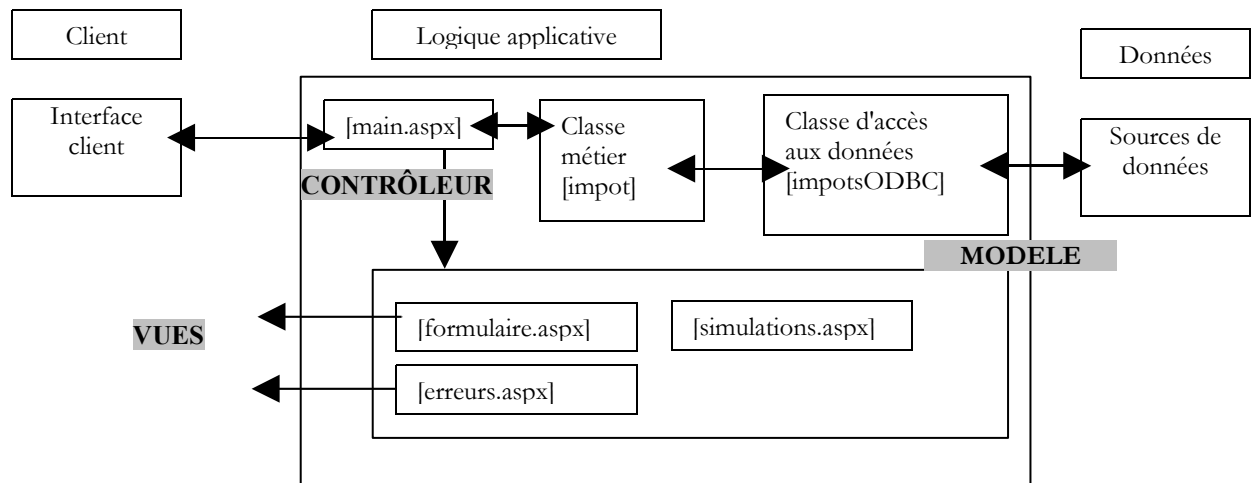
```

Le contrôleur de la page récupère des informations placées par le contrôleur de l'application dans le contexte de la page :

<code>Context.Items("simulations")</code>	objet ArrayList contenant la liste des simulations à afficher. Chaque élément est un tableau de 4 chaînes de caractères représentant les informations (marié, enfants, salaire, impot) de la simulation.
<code>Context.Items("href")</code>	url d'un lien
<code>Context.Items("lien")</code>	texte du lien

5.4.4 Les contrôleurs [global.asax, main.aspx]

Rappelons le schéma MVC de notre application :



Le contrôleur [main.aspx] a à traiter trois actions :

- **init** : correspond à la première requête du client. Le contrôleur affiche la vue [formulaire.aspx]
- **calcul** : correspond à la demande de calcul de l'impôt. Si les données du formulaire de saisie sont correctes, l'impôt est calculé grâce à la classe métier [impotsODBC]. Le contrôleur retourne au client la vue [simulations.aspx] avec le résultat de la simulation courante plus toutes les précédentes. Si les données du formulaire de saisie sont incorrectes, le contrôleur retourne la vue [erreurs.aspx] avec la liste des erreurs et un lien pour retourner au formulaire.
- **retour** : correspond au retour au formulaire après une erreur. Le contrôleur affiche la vue [formulaire.aspx] telle qu'elle a été validée avant l'erreur.

Dans cette nouvelle version, seule l'action [calcul] a changé. En effet, si les données sont valides, elle doit aboutir à la vue [simulations.aspx] alors qu'auparavant elle aboutissait à la vue [formulaire.aspx]. Le contrôleur [main.aspx.vb] devient le suivant :

```
Imports System
...

Public Class main
    Inherits System.Web.UI.Page

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ' avant tout, on regarde si l'application a pu s'initialiser correctement
        ...
        ' on exécute l'action
        Select Case action
            Case "init"
                ' init application
                initAppli()
            Case "calcul"
                ' calcul impot
                calculImpot()
            Case "retour"
                ' retour au formulaire
                retourFormulaire()
            Case "effacer"
                ' init application
                initAppli()
            Case Else
                ' action inconnue = init
                initAppli()
        End Select
    End Sub

    ...

    Private Sub calculImpot()
        ' on sauvegarde les saisies
        Session.Item("formulaire") = Request.Form
        ' on vérifie la validité des données saisies
        Dim erreurs As ArrayList = checkData()
        ' s'il y a des erreurs, on le signale
        If erreurs.Count <> 0 Then
            ' on prépare la page d'erreurs
            context.Items("href") = "main.aspx?action=retour"
            context.Items("lien") = "Retour au formulaire"
            context.Items("erreurs") = erreurs
            Server.Transfer("erreurs.aspx")
        End If
        ' ici pas d'erreurs - on calcule l'impôt
        Dim impot As Long = CType(Application("objImpot"), impot).calculer( _
```

```

Request.Form("rdMarie") = "oui", _
CType(Request.Form("txtEnfants"), Integer), _
CType(Request.Form("txtSalaire"), Long)
' on rajoute le résultat aux simulations existantes
Dim simulations As ArrayList
If Not Session.Item("simulations") Is Nothing Then
    simulations = CType(Session.Item("simulations"), ArrayList)
Else
    simulations = New ArrayList
End If
' ajout de la simulation courante
Dim simulation() As String = New String() {Request.Form("rdMarie").ToString, _
Request.Form("txtEnfants").ToString, Request.Form("txtSalaire").ToString, _
impot.ToString}
simulations.Add(simulation)
' on met les simulations dans la session et le contexte
context.Items("simulations") = simulations
Session.Item("simulations") = simulations
' on affiche la page de résultat
context.Items("href") = "main.aspx?action=retour"
context.Items("lien") = "Retour au formulaire"
Server.Transfer("simulations.aspx", True)
End Sub
...
End Class

```

Nous n'avons conservé ci-dessus que ce qui était nécessaire pour comprendre les modifications qui se trouvent uniquement dans la fonction [calculImpots] :

- tout d'abord la fonction sauve le formulaire [Request.Form] dans la session, ceci afin de pouvoir régénérer le formulaire dans l'état où il a été validé. C'est à faire dans tous les cas, puisque que l'opération aboutisse sur la réponse [erreurs.aspx] ou sur la réponse [simulations.aspx], on revient au formulaire par le lien [Retour au formulaire]. Pour restituer correctement celui-ci, il faut avoir sauvegardé ses valeurs auparavant dans la session.
- si les données saisies s'avèrent correctes, la fonction met la simulation courante (marié, enfants, salaire, impôt) dans la liste des simulations. Celle-ci est trouvée dans la session associée à la clé "simulations".
- la liste des simulations est replacée dans la session pour un usage futur. Elle est également placée dans le contexte courant car c'est là que l'attend la vue [simulations.aspx]
- la vue [simulations.aspx] est affichée une fois mises dans le contextes les autres informations qu'elle attend

5.4.5 Bilan des modifications

L'application est prête à être testée. Listons les modifications amenées aux versions précédentes :

1. une nouvelle vue a été construite
2. le contrôleur [main.aspx.vb] a été modifié en un endroit : traitement de l'action [calcul]

5.4.6 Test de l'application web

Le lecteur est invité à faire les tests. On rappelle la démarche. L'ensemble des fichiers de l'application sont placés dans un dossier <application-path>. Dans ce dossier, est créé un sous-dossier [bin] dans lequel est placé l'assemblage [impot.dll] issu de la compilation des fichiers des classes métier : [impots.vb, impotsData.vb, impotsArray.vb, impotsODBC.vb]. Le fichier [impot.dll] produit par cette commande doit être placé dans <application-path>\bin afin que l'application web y ait accès. Le serveur Cassini est lancé avec les paramètres (<application-path>, /impots4).

5.5 Conclusion

Les exemples précédents ont montré sur un cas concret des mécanismes couramment utilisés dans le développement web. Nous avons utilisé systématiquement l'architecture MVC pour son intérêt pédagogique. On aurait pu traiter ces mêmes exemples différemment et peut-être plus simplement sans cette architecture. Mais celle-ci apporte de grands avantages dès que l'application devient un peu complexe avec de multiples pages.

Nous pourrions poursuivre nos exemples de différentes façons. En voici quelques unes :

- l'utilisateur pourrait vouloir garder ses simulations au fil du temps. Il ferait des simulations un jour J et pourrait les retrouver le jour J+3 par exemple. Une solution possible à ce problème est l'utilisation de cookies. Nous savons que le jeton de session entre le serveur et un client est transporté par ce mécanisme. On pourrait utiliser également celui-ci pour transporter les simulations entre le client et le serveur.

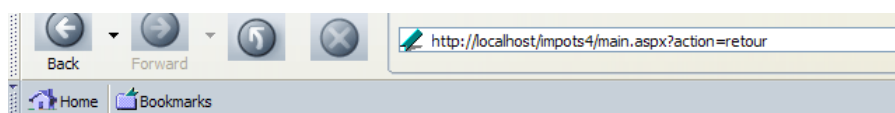
- en même temps que le serveur envoie la page résultat des simulations, il envoie dans ses entêtes HTTP un cookie contenant une chaîne de caractères représentant les simulations. Comme celles-ci sont dans un objet [ArrayList] il y a un travail de transformation de cet objet en [String] à opérer. Le serveur donnerait au cookie une durée de vie, par exemple 30 jours.
- le navigateur client stocke les cookies reçus dans un fichier et les renvoie à chaque fois qu'il fait une requête à un serveur qui les lui a envoyés s'ils sont encore valides (durée de vie pas dépassée). Le serveur recevra pour les simulations une chaîne de caractères [String] qu'il devra transformer en objet [ArrayList].

Les cookies sont gérés par [Response.Cookies] lors de leur envoi vers le client et par [Request.Cookies] lors de leur réception sur le serveur.

- le mécanisme précédent peut devenir assez lourd si il y a un nombre important de simulations. Par ailleurs, il est fréquent qu'un utilisateur nettoie périodiquement ses cookies en les supprimant tous même si par ailleurs il autorise son navigateur à les utiliser. Donc un jour ou l'autre on perdra le cookie des simulations. On peut vouloir alors stocker celles-ci sur le serveur plutôt que sur le client, dans une base de données par exemple. Pour lier des simulations à un utilisateur particulier, l'application pourrait démarrer par une phase d'identification réclamant un login et un mot de passe eux-mêmes stockés dans une base de données ou tout autre type de dépôt de données.
- nous pourrions également vouloir sécuriser le fonctionnement de notre application. Celle-ci fait actuellement deux hypothèses :
 1. l'utilisateur passe toujours par le contrôleur [main.aspx]
 2. et dans ce cas, il utilise toujours les actions proposées dans la page qu'on lui a envoyée

Que se passe-t-il par exemple, si l'utilisateur demande directement l'url [http://localhost/impots4/formulaire.aspx] ? Ce cas est peu probable puisque l'utilisateur ne connaît pas l'existence de cette url. Cependant il doit être prévu. Il peut être géré par le contrôleur d'application [global.asax] qui voit passer toutes les requêtes faites à l'application. il peut ainsi vérifier que la ressource demandée est bien [main.aspx].

Un cas plus probable est qu'un utilisateur n'utilise pas les actions présentes sur la page que le serveur lui a envoyée. Par exemple, que se passe-t-il si l'utilisateur demande directement l'url [http://localhost/impots4/main.aspx?action=retour] sans passer auparavant par le remplissage du formulaire ? Essayons. Nous obtenons la réponse suivante :



Erreur du serveur dans l'application '/impots4'.

La référence d'objet n'est pas définie à une instance d'un objet.

Description : Une exception non gérée s'est produite au moment de l'exécution de la demande Web actuelle. Contrôlez la trace c

Détails de l'exception: System.NullReferenceException: La référence d'objet n'est pas définie à une instance d'un objet.

On a un plantage du serveur. C'est normal. Pour l'action [retour] le contrôleur s'attend à trouver dans la session un objet [NameValueCollection] représentant les valeurs du formulaire qu'il doit afficher. Il ne les trouve pas. Le mécanisme du contrôleur permet de donner une solution élégante à ce problème. Pour chaque requête, le contrôleur [main.aspx] peut vérifier que l'action demandée est bien l'une des actions de la page précédemment envoyée à l'utilisateur. On peut utiliser le mécanisme suivant :

- le contrôleur avant d'envoyer sa réponse au client, stocke dans la session de celui-ci une information identifiant cette page
 - lorsqu'il reçoit une nouvelle demande du client, il vérifie que l'action demandée appartient bien à la dernière page envoyée à ce client
 - les informations liant pages et actions autorisées dans ces pages peuvent être introduites dans le fichier de configuration [web.config] de l'application.
- l'expérience montre que les contrôleurs d'applications ont une large base commune et qu'il est possible de construire un contrôleur générique, la spécialisation de celui-ci pour une application donnée se faisant via un fichier de configuration. C'est la voie suivie par exemple, par l'outil [Struts] dans le domaine de la programmation web en Java.

6 Annexe - Les outils du développement web

Nous indiquons ici où trouver et comment installer des outils gratuits permettant de faire du développement web en java, php, asp et asp.net. 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... :

- 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. Le 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, Cassini), Tomcat.
 - Apache peut être utilisé pour le développement d'applications web en PERL (Practical Extracting and Reporting Language) ou PHP (Personal Home Page)
 - PWS peut être utilisé pour le développement d'applications web en ASP (Active Server Pages) ou PHP sur les plate-formes Windows. Cassini lui permet le développement en ASP.NET.
 - Tomcat est utilisé pour le développement d'applications web à l'aide de servlets Java ou de pages JSP (Java Server pages)
- un **système de gestion de base de données** : MySQL
- **EasyPHP** : un outil qui amène ensemble le serveur Web Apache, le langage PHP et le SGBD MySQL

6.1 Serveurs Web, Navigateurs, Langages de scripts

1. **Serveurs Web principaux**
 - Apache (Linux, Windows)
 - Internet Information Server IIS (NT), Personal Web Server PWS (Windows 9x), Cassini (plate-formes .NET)
2. **Navigateurs principaux**
 - Internet Explorer (Windows)
 - Netscape (Linux, Windows)
 - Mozilla (Linux, Windows)
 - Opera (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)

6.2 Où trouver les outils

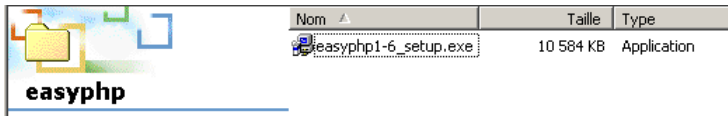
Netscape	http://www.netscape.com/ (lien downloads)
Internet Explorer	http://www.microsoft.com/windows/ie/default.asp
Mozilla	http://www.mozilla.org
PHP	http://www.php.net
Active PERL	http://www.activestate.com
Vbscript, Javascript	http://msdn.microsoft.com/scripting (suivre le lien windows script)
JAVA	http://java.sun.com/
Apache	http://www.apache.org/
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
IIS (windows NT/2000)	http://www.microsoft.com
Tomcat	http://jakarta.apache.org/tomcat/
JBuilder	http://www.borland.com/jbuilder/
EasyPHP	http://www.easyphp.org/
Cassini	http://www.asp.net

6.3 EasyPHP

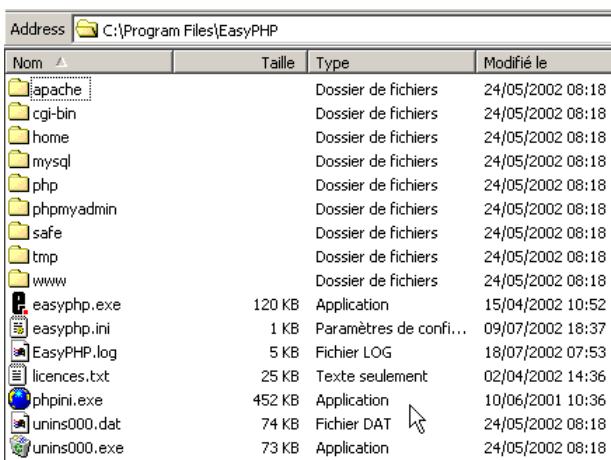
Cette application est très pratique en ce qu'elle amène dans un même paquetage :

- le serveur Web Apache
- un interpréteur PHP
- le SGBD MySQL (3.23.x)
- un outil d'administration de MySQL : PhpMyAdmin

L'application d'installation se présente sous la forme suivante :

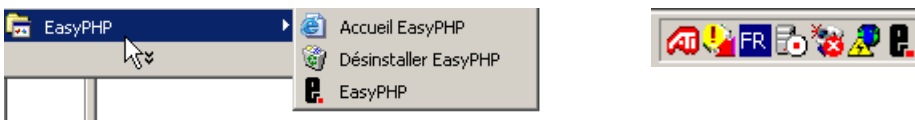


L'installation d'EasyPHP ne pose pas de problème et une arborescence est créée dans le système de fichiers :



easyphp.exe	l'exécutable de l'application
apache	l'arborescence du serveur apache
mysql	l'arborescence du SGBD mysql
phpmysqladmin	l'arborescence de l'application phpmysqladmin
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

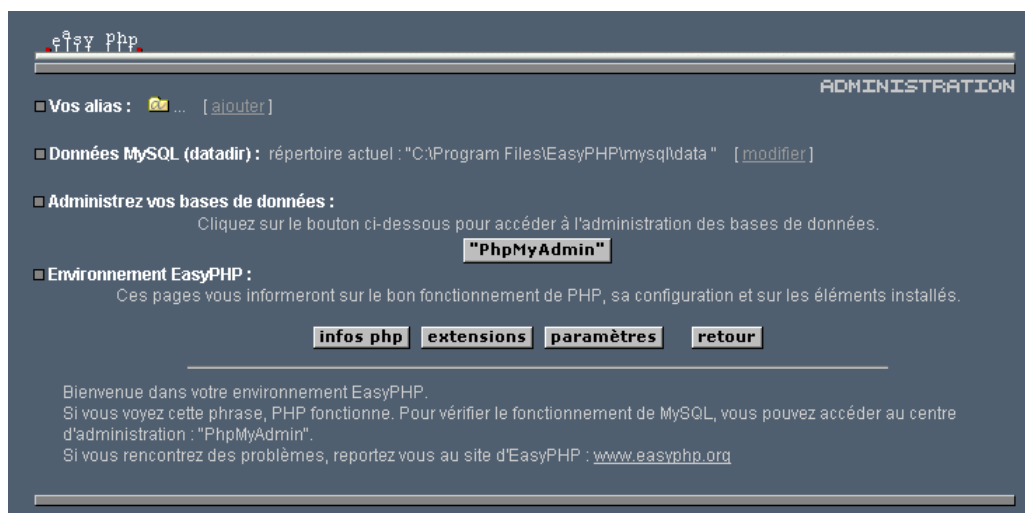
L'intérêt principal d'EasyPHP est que l'application arrive préconfigurée. Ainsi Apache, PHP, MySQL sont déjà configurés pour travailler ensemble. Lorsqu'on lance *EasyPhp* par son lien dans le menu des programmes, une icône se met en place en bas à droite de l'écran.



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 :

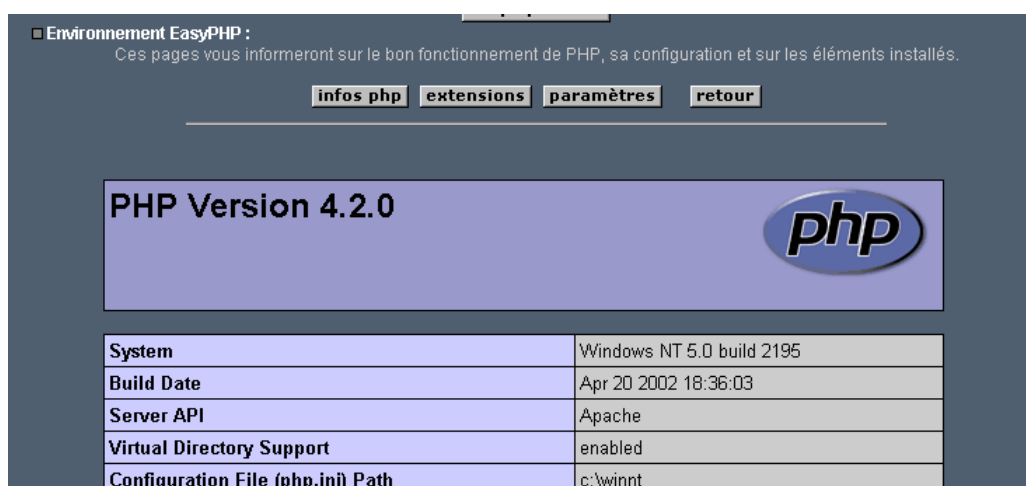


L'option *Administration* permet de faire des réglages et des tests de bon fonctionnement :



6.3.1 Configuration de l'interpréteur PHP

Le bouton **infos php** doit vous permettre de vérifier le bon fonctionnement du couple Apache-PHP : une page d'informations PHP doit apparaître :

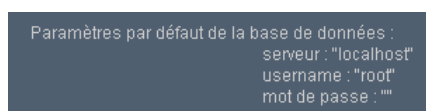


Le bouton **extensions** donne la liste des extensions installées pour php. Ce sont en fait des bibliothèques de fonctions.



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.



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.

6.3.2 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.

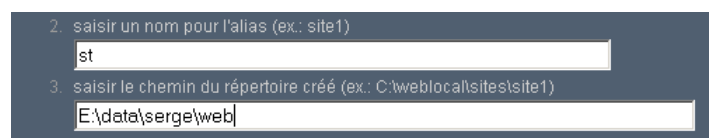


Vos alias : Les alias permettent de placer vos développements dans un ou plusieurs répertoires indépendamment du répertoire racine d'apache (www).

1. créer votre répertoire (ex.: C:\weblocalsites\site1)
2. saisir un nom pour l'alias (ex.: site1)
3. saisir le chemin du répertoire créé (ex.: C:\weblocalsites\site1)
4. paramètres par défaut du répertoire

Options	Indexes	FollowSymLinks	Includes
AllowOverride	All		
#Order	allow,deny		
Allow from	all		

Si dans la page ci-dessus, on met les informations suivantes :



2. saisir un nom pour l'alias (ex.: site1)
st

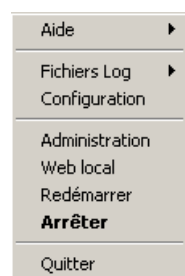
3. saisir le chemin du répertoire créé (ex.: C:\weblocalsites\site1)
E:\data\serge\web

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



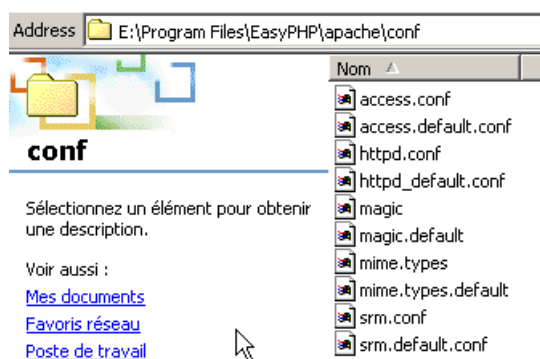
le site de l'ISTIA [ici](#)

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

6.3.3 Le fichier [httpd.conf] 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 <code>http://machine/rep1/fic1.html</code> correspondra au fichier <code>E:\Program Files\EasyPHP\www\rep1\fic1.html</code>
<Directory "E:/Program Files/EasyPHP/www">	fixe les propriétés du dossier précédent
ErrorLog logs/error.log	dossier des logs, donc en fait <code><ServerRoot>\logs\error.log</code> : <code>E:\Program Files\EasyPHP\apache\logs\error.log</code> . 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/"
```

E:\Program Files\EasyPHP\cgi-bin sera la racine de l'arborescence où l'on pourra mettre des scripts CGI. Ainsi l'URL *http://machine/cgi-bin/rep1/script1.pl* sera l'url du script CGI *E:\Program Files\EasyPHP\cgi-bin\rep1\script1.pl*.

```
<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
```

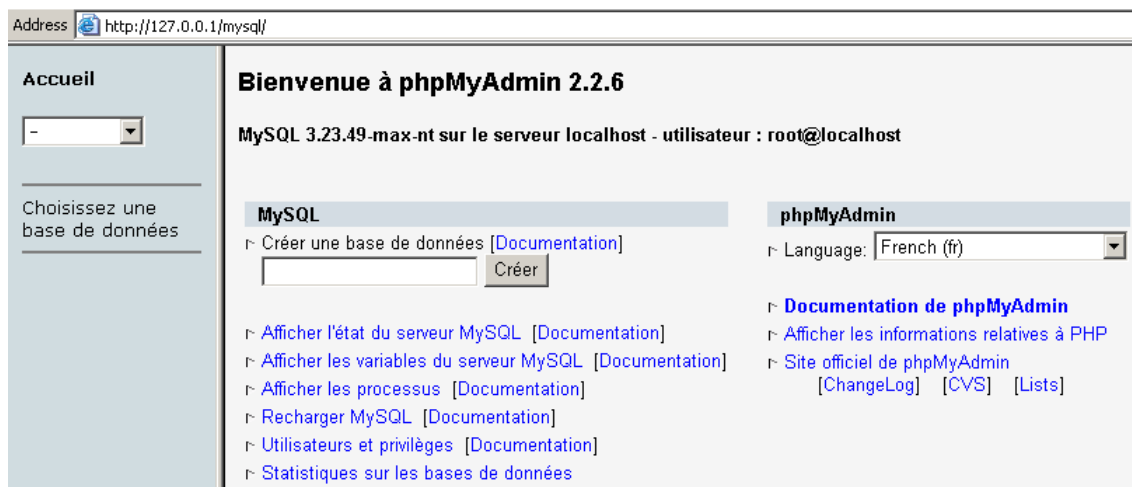
lignes de chargement des modules permettant à Apache de travailler avec PHP4.

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

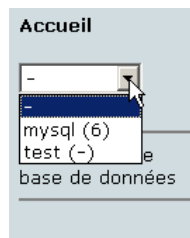
fixe les suffixes des fichiers à considérer comme des fichiers comme devant être traités par PHP

6.3.4 Administration de MySQL avec PhpMyAdmin

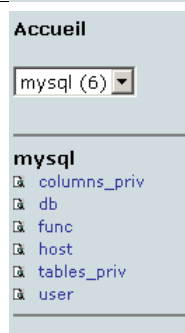
Sur la page d'administration d'EasyPhp, on clique sur le bouton **PhpMyAdmin** :



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	

Tout cocher / Tout décocher
 Pour la sélection :

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.

6.4 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 :

```
dos>"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
```

6.5 PERL

Il est préférable que 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> A 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
```

6.6 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 Active Perl.
- 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
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

nom=dupont
```

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
Copyright (C) Microsoft Corporation 1996-2001. Tous droits réservés.

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
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. Tous droits réservés.

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.

6.7 JAVA

Java est disponible à l'URL : <http://www.sun.com> 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 COPYRIGHT
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
```

Dans la commande ci-dessus [c:\jdk1.3\bin\java] doit être remplacé par le chemin exact de la machine virtuelle java [java.exe].

6.8 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.tpl
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

6.8.1 Configuration

Dans le dossier **<Apache>\conf**, on trouve les fichiers suivants : *httpd.conf*, *srm.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*.



6.8.2 Lien PHP - Apache

Pour tester, créer le fichier **intro.php** avec la seule ligne suivante :

Exemples

<? 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

6.8.3 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>
```


Exemples

```

<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 :

Address  http://localhost/cgi-bin/heure.pl

Il est 9:52:55

6.9 Le serveur PWS

6.9.1 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*.

6.9.2 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/rep1/page1.html` correspondra au fichier `d:\inetpub\wwwroot\rep1\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**

6.9.3 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() ?>

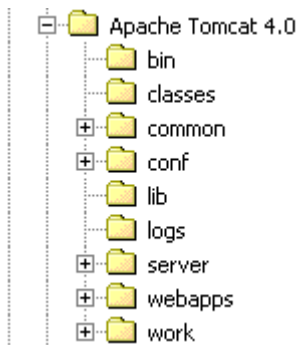
5. Demander au serveur PWS l'URL *http://localhost/php/intro.php*. On doit voir la liste d'informations php déjà présentées avec Apache.

6.10 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.

6.10.1 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 JDK 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

6.10.2 Démarrage/Arrêt du serveur Web Tomcat

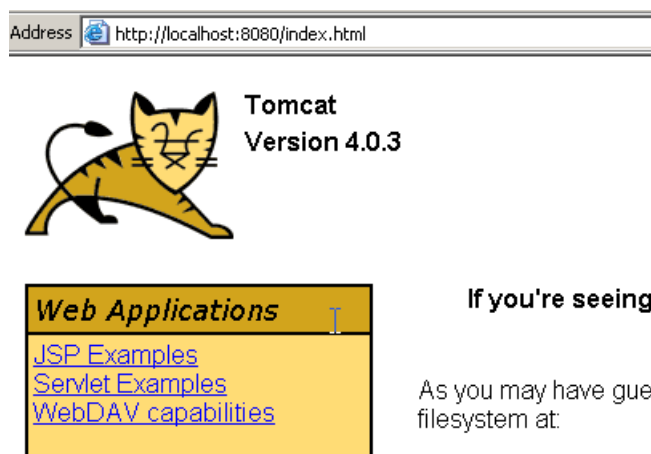
Tomcat est un serveur Web comme l'est Apache ou PWS. Pour le lancer, on dispose de liens dans le menu des programmes :

Start Tomcat pour lancer Tomcat
Stop Tomcat pour l'arrêter

Lorsqu'on lance Tomcat, une fenêtre Dos s'affiche avec le contenu suivant :

```
Start Tomcat
Starting service Tomcat-Standalone
Apache Tomcat/4.0.3
Starting service Tomcat-Apache
Apache Tomcat/4.0.3
```

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 :



Suivez le lien **Servlet Examples** :



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.

6.11 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.

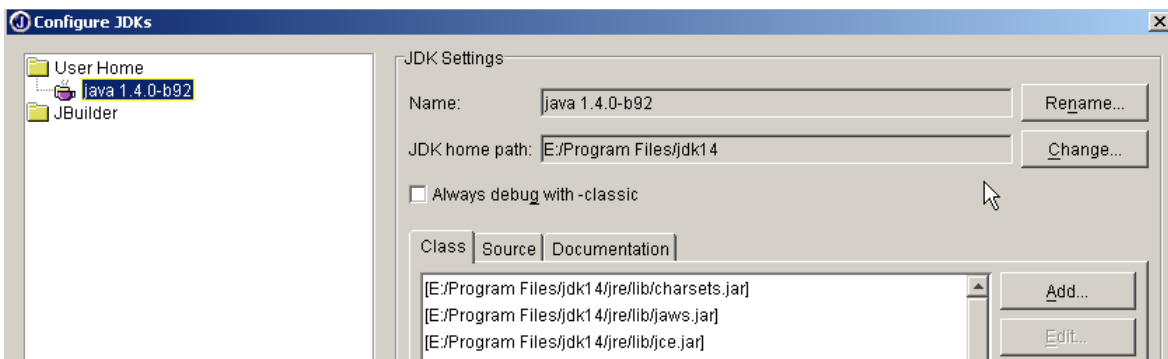
Il y a quelques configurations utiles à faire si on veut utiliser JBuilder pour construire des servlets Java. En effet, la version dite Jbuilder personnel est une version allégée qui ne vient notamment pas avec toutes les classes nécessaires pour faire du

développement web en Java. On peut faire en sorte que JBuilder utilise les bibliothèques de classes amenées par Tomcat. On procède ainsi :

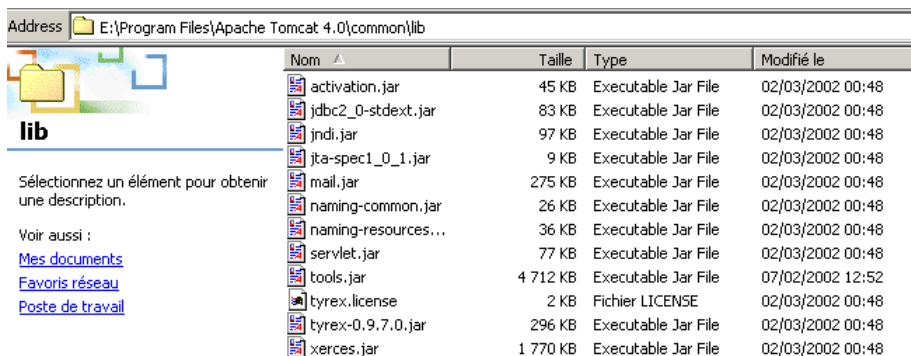
- lancer JBuilder



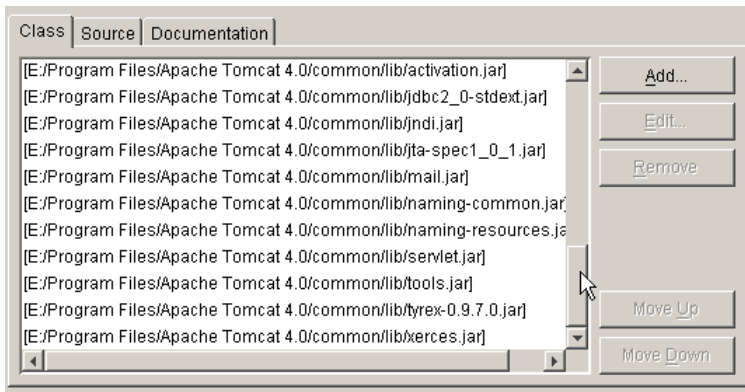
- activer l'option *Tools/Configure JDKs*



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 :



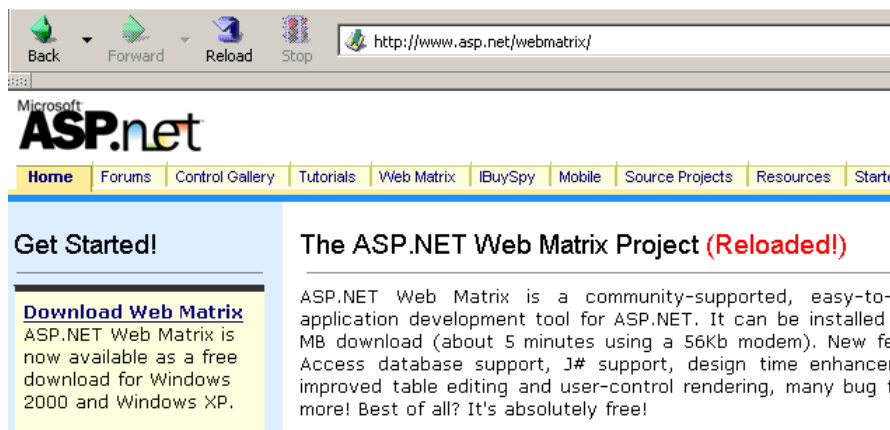
Avec le bouton *Add*, on va ajouter ces bibliothèques, une à une, à la liste des bibliothèques explorées par JBuilder :



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.

6.12 Le serveur Web Cassini

Pour travailler avec la plate-forme .NET de Microsoft, on peut utiliser le serveur web Cassini. Celui-ci est disponible via un autre produit appelé [WebMatrix] qui est un environnement gratuit de développement web sur les plate-formes .NET disponible à l'url :



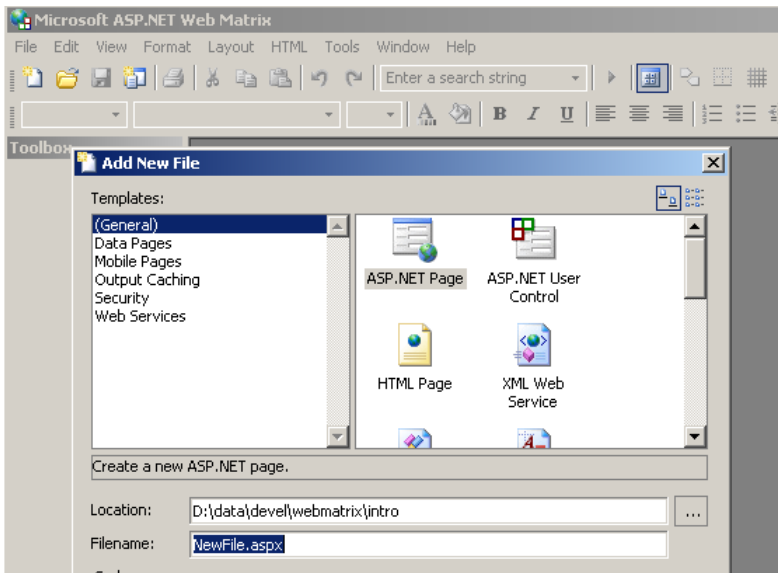
On suivra attentivement la démarche d'installation du produit :

- télécharger et installer la plate-forme .NET (1.1 en mars 2004)
- télécharger et installer WebMatrix
- télécharger et installer MSDE (Microsoft Data Engine) qui est une version limitée de SQL Server.

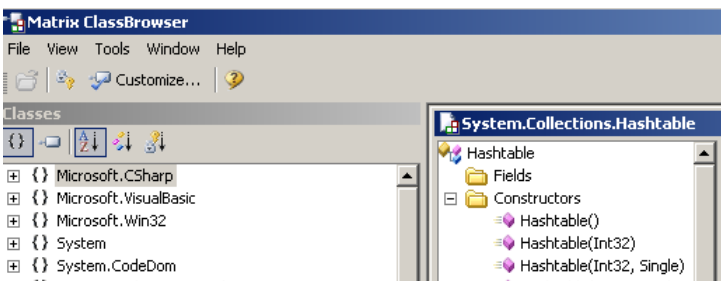
Une fois l'installation terminée, le produit [WebMatrix] est disponible dans les programmes installés :



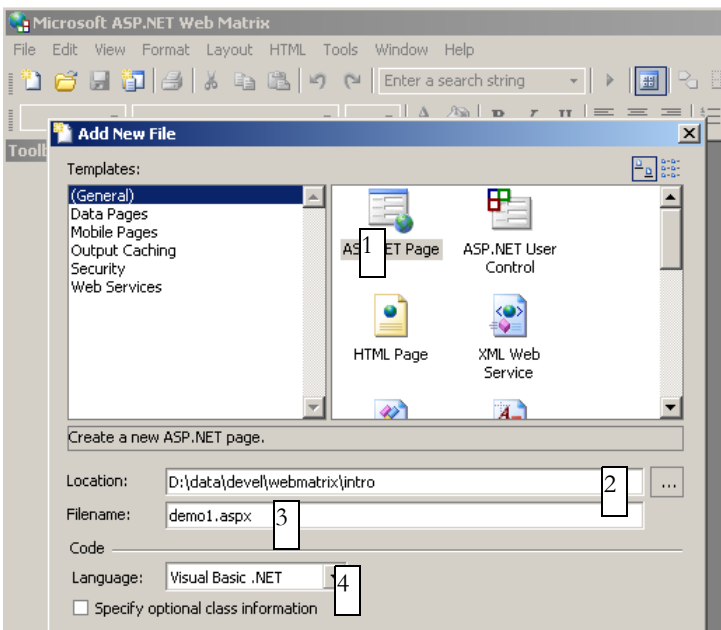
Le lien [ASP.NET] Web Matrix lance l'IDE de développement ASP.NET :



Le lien [Class Browser] lance un outil d'exploration des classes .NET :

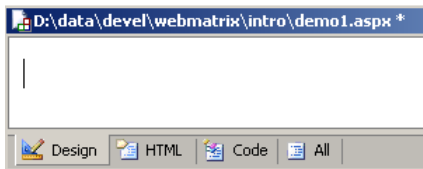


Pour tester l'installation, lançons [WebMatrix] :



Lors du démarrage initial, [WebMatrix] demande les caractéristiques du nouveau projet. C'est sa configuration par défaut. On peut le configurer pour qu'il ne fasse pas apparaître cette boîte de dialogue au démarrage. On l'obtient alors par l'option [File/New File]. [WebMatrix] permet de construire des squelettes pour différentes applications web. Ci-dessus, nous avons précisé avec (1) que nous voulions construire une application [ASP.NET Page] qui est une page Web. Avec (2), nous précisons le dossier dans lequel sera placée cette page Web. Dans (3) nous donnons le nom de la page. Elle doit avoir le suffixe .aspx. Enfin dans (4), nous précisons que nous voulons travailler avec le langage VB.NET, [WebMatrix] supportant par ailleurs les langages C# et J#.

Ceci fait, [WebMatrix] affiche une page d'édition du fichier [demo1.aspx]. Nous y plaçons le code suivant :



- l'onglet [Design] permet de "dessiner" la page web que l'on veut construire. Cela se passe comme avec un IDE de construction d'applications windows.
- la conception graphique de la page Web dans [Design] va générer du code HTML dans l'onglet [HTML]
- la page Web peut contenir des contrôles générant des événements auxquels il faut réagir, un bouton par exemple. Ces événements seront gérés par du code VB.NET qui sera placé dans l'onglet [Code]
- au final, le fichier demo1.aspx est un fichier texte mélangeant code HTML et code VB.NET, résultat de la conception graphique faite dans [Design], du code HTML qu'on a pu ajouter à la main dans [HTML] et du code VB.NET placé dans [Code]. La totalité du fichier est disponible dans l'onglet [All].
- un développeur ASP.NET expérimenté peut construire le fichier demo1.aspx directement avec un éditeur de texte sans l'aide d'aucun IDE.

Sélectionnons l'option [All] On constate que [WebMatrix] a déjà généré du code :

```
<%@ Page Language="VB" %>
<script runat="server">

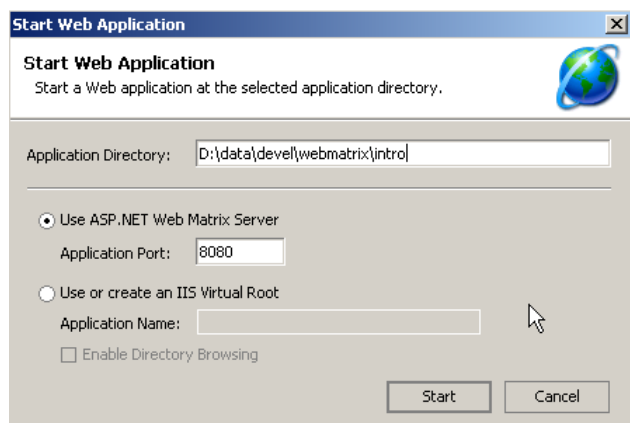
    ' Insert page code here
    '

</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <!-- Insert content here -->
    </form>
</body>
</html>
```

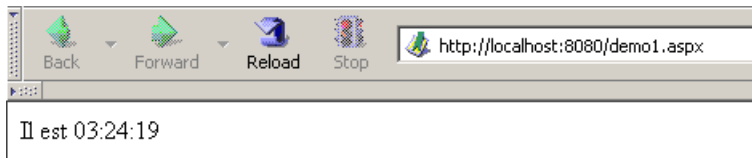
Nous n'allons pas essayer ici d'expliquer ce code. Nous le transformons de la façon suivante :

```
<html>
<head>
<title>Démo asp.net </title>
</head>
<body>
    Il est <% =Date.Now.ToString("hh:mm:ss") %>
</body>
</html>
```

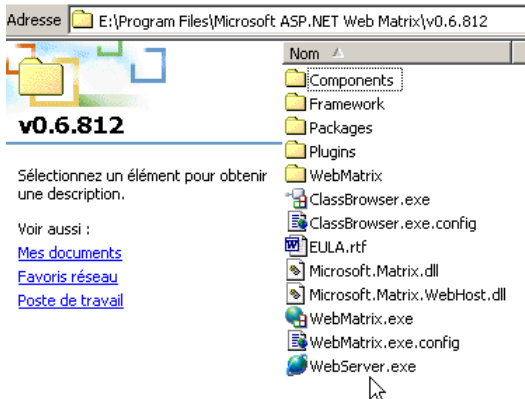
Le code ci-dessus est un mélange de HTML et de code VB.NET. Celui-ci a été placé dans les balises <% ... %>. Pour exécuter ce code, nous utilisons l'option [View/Start]. [WebMatrix] lance alors le serveur Web Cassini s'il n'est pas déjà lancé



On peut accepter les valeurs par défaut proposées dans cette boîte de dialogue et choisir l'option [Start]. Le serveur Web est alors actif. [WebMatrix] va alors lancer le navigateur par défaut de la machine sur laquelle il se trouve et demander l'URL <http://localhost:8080/demo1.aspx> :



Il est possible d'utiliser le serveur Cassini en-dehors de [WebMatrix]. L'exécutable du serveur se trouve dans <WebMatrix>\<version>\WebServer.exe où <WebMatrix> est le répertoire d'installation de [WebMatrix] et <version> son n° de version :



Ouvrons une fenêtre Dos et positionnons-nous dans le dossier du serveur Cassini :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>dir
...
29/05/2003  11:00                53 248 WebServer.exe
...
```

Lançons [WebServer.exe] sans paramètres :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>webserver
```

Nous obtenons une fenêtre d'aide :



L'application [WebServer] appelée également serveur web Cassini admet trois paramètres :

- **/port** : n° de port du service web. Peut-être quelconque. A par défaut la valeur 80
- **/path** : chemin physique d'un dossier du disque
- **/vpath** : dossier virtuel associé au dossier physique précédent. On prêtera attention au fait que la syntaxe n'est pas /path=chemin mais /vpath:chemin, contrairement à ce que dit le panneau d'aide ci-dessus.

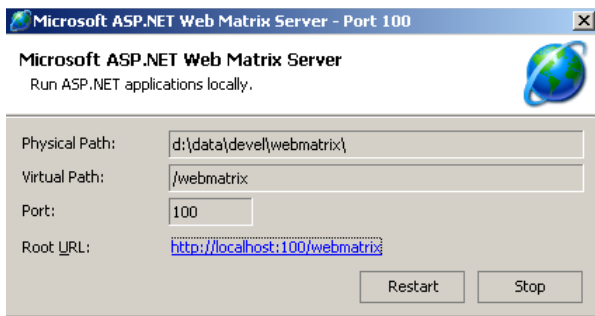
Plaçons le fichier [demo1.aspx] dans le dossier suivant :



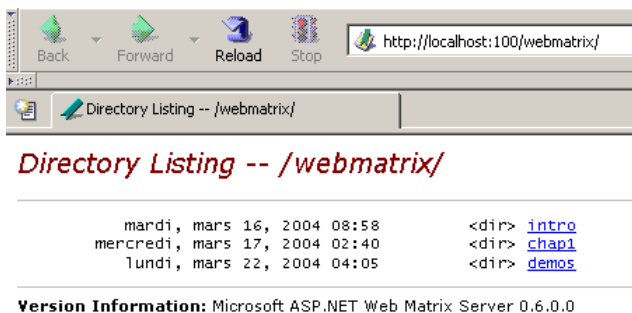
Associations au dossier physique [d:\data\devel\webmatrix] le dossier virtuel [/webmatrix]. Le serveur web pourrait être lancé de la façon suivante :

```
E:\Program Files\Microsoft ASP.NET Web Matrix\v0.6.812>webserver /port:100 /path:"d:\data\devel\webmatrix" /vpath:"/webmatrix"
```

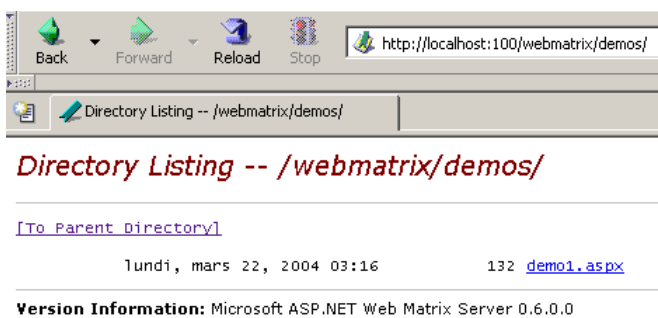
Le serveur Cassini est alors actif et son icône apparaît dans la barre des tâches. Si on double-clique dessus :



On retrouve les paramètres de lancement du serveur. On dispose également de la possibilité d'arrêter [Stop] ou de relancer [Restart] le serveur web. Si on clique sur le lien [Root URL], on obtient la racine de l'arborescence web du serveur dans un navigateur :



Suivons le lien [demos] :



puis le lien [demo1.aspx] :



Il est 04:16:23

On voit donc que si le dossier physique P=[d:\data\devel\webmatrix] a été associé au dossier virtuel V=[/webmatrix] et que le serveur travaille sur le port 100, la page web [demo1.aspx] se trouvant physiquement dans [P\demos] sera accessible localement via l'URL [http://localhost:100/V/demos/demo1.aspx].

Nous avons montré, sur ce cas particulier, que le développement web ASP.NET pouvait être fait avec un simple éditeur de texte pour l'écriture des pages Web et le serveur Web Cassini pour tester celles-ci. Ceci est vrai quelque soit la complexité de l'application. L'IDE [WebMatrix] amène certaines facilités dans le développement mais peu. Un outil comme Visual Studio.NET est beaucoup plus performant mais c'est un produit commercial.

Table des matières

1 LES BASES.....	3
1.1 LES COMPOSANTES D'UNE APPLICATION WEB.....	3
1.1.1 LES ÉCHANGES DE DONNÉES DANS UNE APPLICATION WEB AVEC FORMULAIRE.....	4
1.2 NOTATIONS.....	4
1.3 PAGES WEB STATIQUES, PAGES WEB DYNAMIQUES.....	5
1.3.1 PAGE STATIQUE HTML (HYPERTEXT MARKUP LANGUAGE).....	5
1.3.2 UNE PAGE ASP (ACTIVE SERVER PAGES).....	6
1.3.3 UN SCRIPT PERL (PRACTICAL EXTRACTING AND REPORTING LANGUAGE).....	6
1.3.4 UN SCRIPT PHP (HYPERTEXT PROCESSOR).....	7
1.3.5 UN SCRIPT JSP (JAVA SERVER PAGES).....	8
1.3.6 UNE PAGE ASP.NET.....	9
1.3.7 CONCLUSION.....	9
1.4 SCRIPTS CÔTÉ NAVIGATEUR.....	9
1.4.1 UNE PAGE WEB AVEC UN SCRIPT VBSCRIPT, CÔTÉ NAVIGATEUR.....	10
1.4.2 UNE PAGE WEB AVEC UN SCRIPT JAVASCRIPT, CÔTÉ NAVIGATEUR.....	11
1.5 LES ÉCHANGES CLIENT-SERVEUR.....	11
1.5.1 LE MODÈLE OSI.....	12
1.5.2 LE MODÈLE TCP/IP.....	13
1.5.3 LE PROTOCOLE HTTP.....	15
1.5.3.1 La réponse d'un serveur HTTP.....	15
1.5.3.2 La demande d'un client HTTP.....	19
1.5.4 CONCLUSION.....	21
1.6 LE LANGAGE HTML.....	21
1.6.1 UN EXEMPLE.....	21
1.6.1.1 Le formulaire.....	26
1.6.1.2 Champ de saisie.....	26
1.6.1.3 Champ de saisie multilignes.....	27
1.6.1.4 Boutons radio.....	27
1.6.1.5 Cases à cocher.....	27
1.6.1.6 Liste déroulante (combo).....	28
1.6.1.7 Liste à sélection unique.....	28
1.6.1.8 Liste à sélection multiple.....	28
1.6.1.9 Bouton de type button.....	29
1.6.1.10 Bouton de type submit.....	29
1.6.1.11 Bouton de type reset.....	30
1.6.1.12 Champ caché.....	30
1.6.2 ENVOI À UN SERVEUR WEB PAR UN CLIENT WEB DES VALEURS D'UN FORMULAIRE.....	30
1.6.2.1 Méthode GET.....	31
1.6.2.2 Méthode POST.....	35
1.7 CONCLUSION.....	36
2 INTRODUCTION AU DÉVELOPPEMENT WEB ASP.NET.....	37
2.1 INTRODUCTION.....	37
2.2 LES OUTILS.....	38
2.3 PREMIERS EXEMPLES.....	40
2.3.1 EXEMPLE DE BASE - VARIANTE 1.....	41
2.3.2 EXEMPLE DE BASE - VARIANTE 2.....	41
2.3.3 EXEMPLE DE BASE - VARIANTE 3.....	42
2.3.4 EXEMPLE DE BASE - VARIANTE 4.....	43
2.3.5 EXEMPLE DE BASE - VARIANTE 5.....	43
2.3.6 EXEMPLE DE BASE - VARIANTE 6.....	45
2.3.7 CONCLUSION.....	46
3 LES FONDAMENTAUX DU DÉVELOPPEMENT ASP.NET.....	47

3.1	LA NOTION D'APPLICATION WEB ASP.NET	47
3.1.1	INTRODUCTION	47
3.1.2	CONFIGURER UNE APPLICATION WEB	48
3.1.3	APPLICATION, SESSION, CONTEXTE	49
3.1.3.1	Le fichier global.asax	49
3.1.3.2	Exemple 1	50
3.1.3.3	Une vue d'ensemble	53
3.1.3.4	Exemple 2	54
3.1.3.5	De la nécessité de synchroniser les clients d'une application	56
3.1.3.6	Synchronisation des clients	57
3.1.3.7	Gestion du jeton de session	60
3.2	RÉCUPÉRER LES INFORMATIONS D'UNE REQUÊTE CLIENT	68
3.2.1	LE CYCLE REQUÊTE-RÉPONSE DU CLIENT-SERVEUR WEB	68
3.2.2	RÉCUPÉRER LES INFORMATIONS TRANSMISES PAR LE CLIENT	69
3.2.3	EXEMPLE 1	70
3.2.4	EXEMPLE 2	72
3.3	MISE EN OEUVRE D'UNE ARCHITECTURE MVC	75
3.3.1	LE CONCEPT	75
3.3.2	CONTRÔLER UNE APPLICATION MVC SANS SESSION	76
3.3.3	CONTRÔLER UNE APPLICATION MVC AVEC SESSION	78
3.4	CONCLUSION	81
4	GESTION DE L'INTERFACE UTILISATEUR	81
4.1	INTRODUCTION	81
4.2	LE LANGAGE HTML	81
4.2.1	UN EXEMPLE	81
4.2.2	CONSTRUCTION D'UN FORMULAIRE	90
4.2.2.1	Le formulaire	93
4.2.2.2	Champ de saisie	93
4.2.2.3	Champ de saisie multilignes	93
4.2.2.4	Boutons radio	94
4.2.2.5	Cases à cocher	94
4.2.2.6	Liste déroulante (combo)	94
4.2.2.7	Liste à sélection unique	95
4.2.2.8	Liste à sélection multiple	95
4.2.2.9	Bouton de type button	95
4.2.2.10	Bouton de type submit	96
4.2.2.11	Bouton de type reset	96
4.2.2.12	Champ caché	96
4.3	ENVOI À UN SERVEUR WEB PAR UN NAVIGATEUR DES VALEURS D'UN FORMULAIRE	97
4.3.1	MÉTHODE GET	97
4.3.2	MÉTHODE POST	102
4.4	TRAITEMENT CÔTÉ SERVEUR DES VALEURS D'UN FORMULAIRE	104
4.4.1	PRÉSENTATION DE L'EXEMPLE	104
4.4.2	LE CONTRÔLEUR DE L'APPLICATION	105
4.4.3	TRAITEMENT DE L'ACTION INIT	106
4.4.4	TRAITEMENT DE L'ACTION VALIDATION	107
4.4.5	TESTS	109
4.5	MAINTENIR L'ÉTAT D'UNE PAGE	112
4.5.1	MAINTENIR L'ÉTAT D'UNE PAGE AVEC UNE SESSION	112
4.5.2	LE NOUVEAU CONTRÔLEUR D'APPLICATION	114
4.5.3	LE NOUVEAU FORMULAIRE	115
4.5.4	LA PAGE DE VALIDATION	119
4.5.5	LES TESTS	120
4.5.6	CONCLUSION	122
5	EXEMPLES	123
5.1	EXEMPLE 1	123
5.1.1	LE PROBLÈME	123

5.1.2	LA STRUCTURE MVC DE L'APPLICATION.....	123
5.1.3	LA CLASSE MÉTIER.....	124
5.1.4	LA CLASSE D'ACCÈS AUX DONNÉES.....	124
5.1.5	TESTS DES CLASSES MÉTIER ET DES CLASSES D'ACCÈS AUX DONNÉES.....	126
5.1.6	LES VUES DE L'APPLICATION WEB.....	128
5.1.7	LA VUE [FORMULAIRE.ASPX].....	129
5.1.8	LA VUE [ERREURS.ASPX].....	131
5.1.9	LES CONTRÔLEURS [GLOBAL.ASAX, MAIN.ASPX].....	131
5.1.9.1	Les actions init, effacer.....	135
5.1.9.2	L'action calcul.....	135
5.1.9.3	L'action retour.....	136
5.1.10	TEST DE L'APPLICATION WEB.....	136
5.1.11	UTILISATION DU CLIENT [CURL].....	138
5.2	EXEMPLE 2.....	141
5.2.1	LE PROBLÈME.....	141
5.2.2	LA SOURCE DE DONNÉES ODBC.....	141
5.2.3	UNE NOUVELLE CLASSE D'ACCÈS AUX DONNÉES.....	144
5.2.4	TESTS DE LA CLASSE D'ACCÈS AUX DONNÉES.....	147
5.2.5	LES VUES DE L'APPLICATION WEB.....	150
5.2.6	LES CONTRÔLEURS D'APPLICATION [GLOBAL.ASAX, MAIN.ASPX].....	150
5.2.7	BILAN DES MODIFICATIONS.....	151
5.2.8	TEST DE L'APPLICATION WEB.....	151
5.3	EXEMPLE 3.....	152
5.3.1	LE PROBLÈME.....	152
5.3.2	LA SOURCE DE DONNÉES OLEDB.....	152
5.3.3	LA CLASSE D'ACCÈS AUX DONNÉES.....	152
5.3.4	TESTS DE LA CLASSE D'ACCÈS AUX DONNÉES.....	156
5.3.5	LES VUES DE L'APPLICATION WEB.....	158
5.3.6	LES CONTRÔLEURS D'APPLICATION [GLOBAL.ASAX, MAIN.ASPX].....	158
5.3.7	BILAN DES MODIFICATIONS.....	159
5.3.8	TEST DE L'APPLICATION WEB.....	159
5.4	EXEMPLE 4.....	159
5.4.1	LE PROBLÈME.....	159
5.4.2	LA STRUCTURE MVC DE L'APPLICATION.....	160
5.4.3	LES VUES DE L'APPLICATION WEB.....	160
5.4.4	LES CONTRÔLEURS [GLOBAL.ASAX, MAIN.ASPX].....	163
5.4.5	BILAN DES MODIFICATIONS.....	165
5.4.6	TEST DE L'APPLICATION WEB.....	165
5.5	CONCLUSION.....	165

6 ANNEXE - LES OUTILS DU DÉVELOPPEMENT WEB..... 167

6.1	SERVEURS WEB, NAVIGATEURS, LANGAGES DE SCRIPTS.....	167
6.2	OÙ TROUVER LES OUTILS.....	167
6.3	EASYPHP.....	168
6.3.1	CONFIGURATION DE L'INTERPRÉTEUR PHP.....	169
6.3.2	ADMINISTRATION APACHE.....	170
6.3.3	LE FICHIER [HTPD.CONF] DE CONFIGURATION D'APACHE HTPD.CONF.....	171
6.3.4	ADMINISTRATION DE MYSQL AVEC PHPMYADMIN.....	172
6.4	PHP.....	173
6.5	PERL.....	173
6.6	VBSOFT, JAVASCRIPT, PERLSOFT.....	174
6.7	JAVA.....	176
6.8	SERVEUR APACHE.....	177
6.8.1	CONFIGURATION.....	177
6.8.2	LIEN PHP - APACHE.....	177
6.8.3	LIEN PERL-APACHE.....	178
6.9	LE SERVEUR PWS.....	179
6.9.1	INSTALLATION.....	179
6.9.2	PREMIERS TESTS.....	179
6.9.3	LIEN PHP - PWS.....	179
6.10	TOMCAT : SERVLETS JAVA ET PAGES JSP (JAVA SERVER PAGES).....	180
6.10.1	INSTALLATION.....	180

6.10.2 DÉMARRAGE/ARRÊT DU SERVEUR WEB TOMCAT.....	180
6.11 JBUILDER.....	181
6.12 LE SERVEUR WEB CASSINI.....	183