



<http://www.gasp-fr.com>

## Le site de formation et d'information sur les ASP (Active Server Pages)

### Tutorial d'apprentissage rapide ASP

Extrait de l'ouvrage : **Initiation à XML – Wrox Press – isbn 2212092482**

Ce tutorial vous concerne si votre expérience en ASP est *limitée*, ou si vous avez simplement besoin d'un bref rappel. Son but n'est pas de vous enseigner tout ce qu'il faut savoir à propos d'ASP : cela fait l'objet de nombreux ouvrages, comme *Initiation à ASP 3.0* (ISBN 2-212-09236-9) de **Wrox Press**.

ASP est un outil extraordinaire servant à créer des pages web dynamiques. C'est une technologie de Microsoft qui offre les fonctionnalités d'un langage de programmation pour générer dynamiquement le HTML de nos pages web. L'utilisation d'ASP permet de faire de nombreuses choses. Vous pouvez vous appuyer sur toute la richesse des données disponibles sur le serveur et dans les multiples bases de données réparties dans l'entreprise. Vous pouvez personnaliser les pages en fonction des différents besoins des utilisateurs qui viennent sur votre site. De plus, en laissant votre code du côté du serveur, vous pouvez construire une bibliothèque de fonctionnalités, réutilisable à volonté pour améliorer d'autres sites web. Mais surtout, l'utilisation de bibliothèques de scripts côté serveur permettra à vos sites web de s'adapter à l'architecture applicative web distribuée.

Vous devez au préalable avoir une bonne connaissance du protocole HTTP et de la manière dont un serveur HTTP interagit avec un navigateur. Il est important de comprendre ce modèle quand on développe des applications réparties entre le serveur et le client. Nous allons examiner HTTP et résumer rapidement la façon dont le protocole fonctionne. Puis, nous vous donnerons une vue globale de Active Server Pages, ou ASP.

Il peut être utilisé pour créer de simples pages web statiques, mais aussi des sites dynamiques puissants qui utilisent des bases de données, HTML et les scripts. Son autre usage important est celui de *colle* de programmation. En utilisant ASP, vous pouvez créer et manipuler des composants côté serveur. Ces composants peuvent, par exemple, fournir

des données à votre application comme la génération de graphiques, ou encore vous connecter à une base de données sur un mainframe ou vers une application XML. ASP se contente de faciliter l'utilisation de ces composants sur le Web. ASP comporte des objets internes qu'il est important de comprendre pour pouvoir utiliser pleinement leur potentiel. Nous examinerons chacun de ces objets en profondeur. Enfin, nous construirons un exemple opérationnel d'utilisation de ASP sur un site web.

## Anatomie du protocole HTTP

Pour un utilisateur, surfer sur le web est très simple puisque cela se limite à cliquer sur un lien dans un navigateur. Mais savez-vous ce qui se passe réellement en dessous du capot du navigateur web ? Ces opérations peuvent être assez complexes mais elles ne sont pas très difficiles à suivre. Cela vous aidera surtout à comprendre le fonctionnement des scripts côté client et côté serveur.

### Vision générale

**HTTP (Hypertext Transfer Protocol)** est un protocole TCP/IP *de niveau applicatif*. Un protocole de niveau applicatif est un protocole qui voyage par dessus un autre protocole. Dans ce cas, HTTP voyage par dessus TCP, qui est également un protocole. Quand deux ordinateurs communiquent par une connexion TCP/IP, les données sont formatées et traitées de telle manière qu'il est garanti qu'elles arrivent à destination. Ce mécanisme élaboré est le protocole TCP/IP.

HTTP prend pour acquis le processus TCP/IP dans son ensemble et il l'ignore largement. Il s'appuie à la place sur des commandes texte comme `GET` et `PUT`. Les protocoles de niveau applicatif sont mis en œuvres habituellement à l'intérieur d'une application (par opposition au niveau driver), d'où leur nom. D'autres exemples de protocoles de niveau applicatif sont **File Transfer Protocol (FTP)** et les protocoles de courrier électronique, **Standard Mail Transfer Protocol (SMTP)** et **Post Office Protocol (POP3)**. Des données purement binaires sont rarement transmises par ces protocoles, mais lorsque c'est le cas, elles sont codées au format ASCII, ce qui est pour le moins inefficace. Les futures versions du protocole HTTP rectifieront ce problème. La version la plus à jour de HTTP est la version 1.1 et la plupart des serveurs web disponibles à ce jour prennent en charge cette version.

Il y a également un nouveau protocole HTTP en préparation appelé HTTP-NG, ou HTTP-Next Generation. Ce protocole plus récent et robuste utilisera la bande passante de façon plus efficace et améliorera certaines limites du HTTP d'origine. La plus grosse amélioration dans le nouveau protocole est le transfert des données qui se fera en binaire et pas en texte, ce qui rendra les transactions plus rapides. Des informations plus techniques sur HTTP-NG sont disponibles à partir de W3C à l'adresse <http://www.w3.org/Protocols/HTTP-NG/Activity.html>.

## Serveur HTTP

Pour effectuer une requête HTTP, un serveur HTTP (serveur web) doit fonctionner sur la machine cible. Ce serveur est une application qui attend les requêtes HTTP et y répond. Cette « écoute » est faite sur un port précis TCP (par défaut, le port 80). Une requête HTTP concerne un élément unique du serveur web. Cet élément peut être, par exemple, une page web ou un fichier son. Le serveur, sur réception de la requête, essaie de récupérer les données demandées. S'il trouve ces données, il les formate et les renvoie au client. À l'inverse, si les informations demandées ne peuvent pas être trouvées, le serveur renvoie un message d'erreur.

La lecture d'une seule page web dans votre navigateur peut se traduire par l'exécution de dizaines de transactions HTTP. Chaque élément de page web qui n'est pas du texte doit être demandé au serveur HTTP séparément. Le point essentiel ici est que chaque transaction HTTP est formée d'une requête et d'une réponse :



C'est dans ce modèle de transactions que vous devez vous placer quand vous programmez des applications web.

## Notions de base sur les protocoles

Une transaction HTTP est constituée de quatre états de base, qui sont :

- Connexion (Connection)
- Requête (Request)
- Réponse (Response)
- Déconnexion (Disconnection)

Un client se connecte à un serveur et envoie une requête. Il attend une réponse, puis se déconnecte. Une connexion ne dure typiquement que quelques secondes. Sur des sites web comme Yahoo où les données ne comportent pratiquement pas de graphiques et où les informations sont relativement statiques, les requêtes durent moins d'une seconde.

### **Connexion**

Le logiciel client, un navigateur web dans ce cas, établit une connexion TCP/IP avec un serveur HTTP sur tout autre port TCP/IP spécifique. Le port 80 est utilisé par défaut. Un serveur web peut cependant résider sur tout port autorisé. Ce choix est entièrement laissé à l'opérateur du serveur web. De plus, la modification délibérée des numéros de port constitue souvent la première ligne de défense contre les utilisateurs non autorisés.

### **Requête**

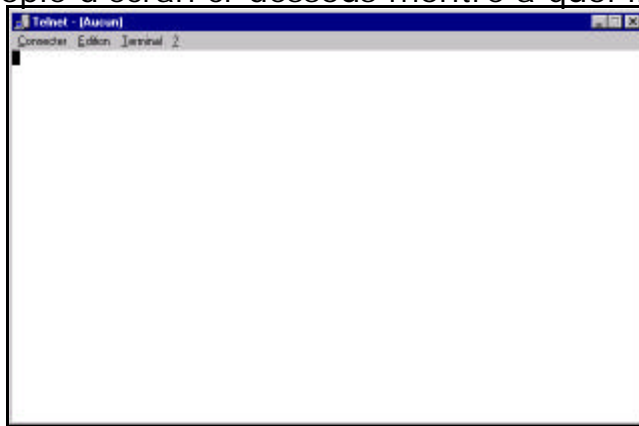
Une fois connecté, le client envoie une requête au serveur. Cette requête est en ASCII et elle doit se terminer par un couple retour-chariot/saut-de-ligne. Chaque requête doit spécifier une méthode qui indique au serveur

ce que veut le client. En HTTP 1.1, il existe huit méthodes: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE et CONNECT. Pour plus d'informations sur les différentes méthodes et leur utilisation, vous pouvez vous référer aux spécifications HTTP sur le site web de W3C. Dans le cadre de ce chapitre, nous allons nous concentrer sur la méthode GET.

La méthode GET demande au serveur web de renvoyer la page spécifiée. Le format de cette requête est le suivant :

```
GET <URL> <HTTP Version>
```

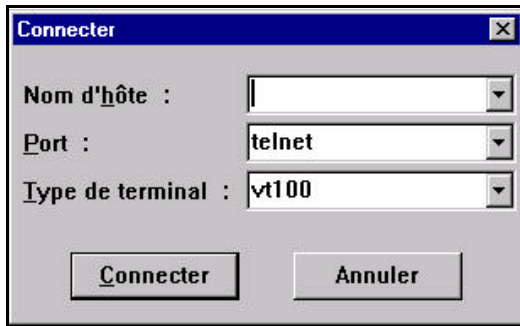
Vous pouvez réaliser vos propres requêtes HTTP avec le programme **telnet**. Telnet est un programme disponible sur la plupart des systèmes informatiques. Il a été conçu au départ pour être utilisé sur les systèmes UNIX. Comme UNIX est à la base orienté caractères, on peut se connecter depuis un site distant et travailler avec le système d'exploitation. Telnet est le programme qui vous permet de vous connecter à une machine distante. En outre, toutes les versions de Windows incluent un programme telnet. La copie d'écran ci-dessous montre à quoi il ressemble .



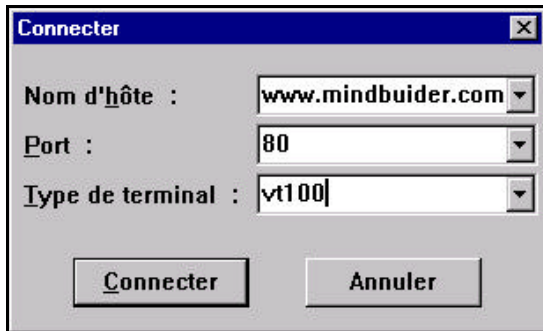
*Le telnet de Microsoft laisse beaucoup à désirer. Heureusement, la société Van Dyke Technologies ([www.vandyke.com](http://www.vandyke.com)) a créé un excellent programme telnet appelé CRT.*

Telnet utilise par défaut le protocole TCP/IP port 23. Pour entrer *via* telnet dans une machine UNIX, cette dernière doit faire tourner un serveur telnet. Ce serveur est à l'écoute des connexions telnet qui arrivent sur le port 23. Ceci dit, presque tous les programmes Telnet vous permettent de spécifier le port sur lequel vous vous connectez. C'est cette fonctionnalité que nous allons utiliser pour regarder fonctionner HTTP. Si vous choisissez de ne pas télécharger le client Telnet de Van Dyke, vous pouvez faire le test en exécutant le Telnet intégré de Windows. Windows n'a pas d'option de menu prédéfinie pour ce programme, mais il est généralement sous C:\windows\telnet.exe. Pour le lancer, appuyez sur le bouton Démarrer et sélectionnez Exécuter. Tapez telnet puis appuyez sur ENTREE. Vous devriez voir apparaître une fenêtre telnet similaire à celle ci-dessus.

Sélectionnez Système distant dans le menu Connexion et vous verrez apparaître la fenêtre de dialogue suivante :



Tapez le nom d'un serveur web quelconque ; nous avons choisi `http://www.mindbuilder.com`. Puis, entrez le port du serveur web, qui est pratiquement toujours 80.



La barre de titre va changer et afficher le nom du serveur sur lequel vous êtes connecté. Il n'y a pas d'autre indication de connexion. Vous devez alors taper votre commande HTTP. Tapez ce qui suit, tout en majuscules :

```
GET / HTTP/1.0
```

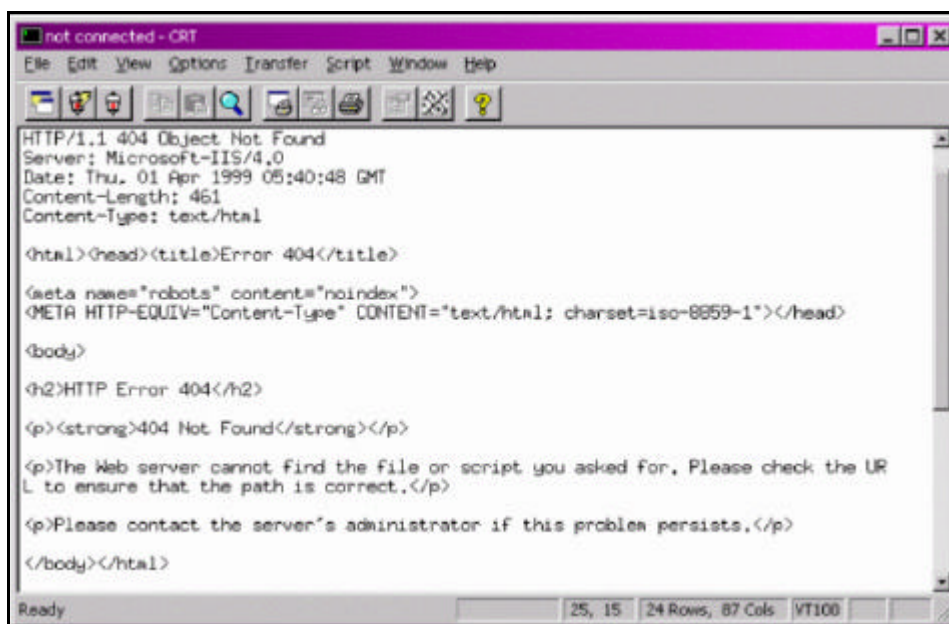
Notez qu'à moins d'avoir coché **Echo Local** dans les **Préférences**, vous ne verrez pas ce que vous tapez. Après avoir entré la commande, vous devez envoyer un retour chariot (*Ctrl-M*) suivi d'un saut de ligne (*Ctrl-J*). La réponse à votre requête http est alors renvoyée de la manière suivante :

```
not connected - CRT
File Edit View Options Transfer Script Window Help
[Icons]
GET / HTTP/1.0
^J
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Content-Location: http://209.224.126.2/Default.htm
Date: Thu, 01 Apr 1999 05:27:57 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Tue, 03 Nov 1998 15:40:08 GMT
ETag: "0fc103c407be1:16ef"
Content-Length: 1062

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<META NAME="KEYWORDS" CONTENT="mindbuilder; multimedia; cd-rom; training software;
corporate communications; internet; design; graphics; web design; chicago; computer
based training; web based training; virtual tour; annual reports; 3d">
<META NAME="DESCRIPTION" CONTENT="At MindBuilder, we are committed to training, com
munication and the transfer of knowledge. Our full suite of training products combi
ne cutting edge educational techniques with true multimedia involving users, studen
ts and customers with the subtleties of your message that frequently get lost, over
looked or ignored. The result is quantifiable gains, quantifiable improvements, bet
ter ROI.">
<TITLE>M i n d B u i l d e r</TITLE>
</HEAD>
<FRAMESET COLS="5,*" border=0>
  <FRAME SCROLLING=NO NAME="preload" SRC="shock_preload.html" NORESIZE MARGIN
WIDTH=0 MARGINHEIGHT=0>
  <FRAME SCROLLING=AUTO NAME="main" SRC="default_main.html" NORESIZE MARGINWI
DTH=0 MARGINHEIGHT=0>
</FRAMESET>
</HTML>
Ready 42, 1 44 Rows, 88 Cols VT100
```

## Réponse

Sur réception de la requête, le serveur web répondra. Ce qui se traduira probablement par certaines données HTML comme dans l'exemple précédent. Néanmoins, vous pouvez également obtenir une erreur :



```
not connected - CRT
File Edit View Options Transfer Script Window Help
[Icons]
HTTP/1.1 404 Object Not Found
Server: Microsoft-IIS/4.0
Date: Thu, 01 Apr 1999 05:40:48 GMT
Content-Length: 461
Content-Type: text/html

<html><head><title>Error 404</title>
<meta name="robots" content="noindex">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1"></head>
<body>
<h2>HTTP Error 404</h2>
<p><strong>404 Not Found</strong></p>
<p>The Web server cannot find the file or script you asked for. Please check the URL to ensure that the path is correct.</p>
<p>Please contact the server's administrator if this problem persists.</p>
</body></html>
Ready [Status Bar] 25, 15 24 Rows, 87 Cols VT100
```

La réponse est à nouveau en HTML, mais le code renvoyé est un code d'erreur (404) au lieu d'un OK (200). Ici, l'URL saisie est inexacte et la page web ne peut être récupérée.

### En-têtes HTTP

Ce qui a été renvoyé est en réalité une réponse en deux parties. La première partie est constituée d'« en-têtes http ». Ces en-têtes fournissent des informations sur la réponse effective à la requête, l'en-tête le plus important étant `status`. Dans la copie d'écran ci-dessus, on peut lire HTTP/1.1 404 Object Not Found. Ceci indique le statut effectif de la requête.

Les autres en-têtes qui ont été renvoyés avec cette requête sont `Server`, `Date`, `Content-Length` et `Content-Type`. Il y a plusieurs types d'en-têtes différents, tous conçus pour aider le navigateur à identifier facilement le type d'informations renvoyées.

### Déconnexion

Après avoir répondu à votre requête, le serveur ferme la connexion et vous déconnecte. Les requêtes suivantes exigent que vous rétablissiez votre connexion avec le serveur.

## Introduction à Active Server Pages

Vous avez pu constater, dans la présentation de l'architecture HTTP du paragraphe précédent, que le véritable cœur du protocole HTTP réside dans la requête et la réponse. Le client envoie une requête au serveur et le serveur fournit la réponse au client. Ce principe est vraiment l'un des fondements de l'informatique client/serveur. Nous rencontrons ce mode de fonctionnement partout dans le monde actuel de la programmation, et pas seulement pour la programmation web.

Microsoft a bien intégré ce schéma de fonctionnement général et développé une nouvelle technologie qui a rendu la programmation web beaucoup plus accessible. Cette technologie s'appelle Active Server Pages ou ASP. ASP est un environnement d'écriture de scripts côté serveur qui fait partie de IIS (Internet Information Server), le serveur web le plus courant de Microsoft. ASP vous permet d'inclure des commandes de script à l'intérieur de vos documents HTML. Les commandes de scripts sont interprétées par le serveur qui génère alors le HTML correspondant, puis le renvoie vers le navigateur. Cela permet au développeur web de créer un contenu dynamique actualisé. L'intérêt de cette approche est que le navigateur de votre utilisateur web n'a aucune espèce d'importance, puisque que le serveur ne renvoie que du HTML pur. Vous pouvez bien sûr compliquer le HTML renvoyé avec une programmation plus avancée et spécifique au navigateur, mais cela reste votre prérogative. ASP peut faire bien d'autres choses, et nous verrons plus tard dans ce chapitre certaines autres de ses capacités, comme la validation de formulaire et la manipulation de données.

Bien qu'il soit possible d'utiliser des langages comme JavaScript ou même Perl, le langage de script par défaut de ASP est VBScript.

## Comment le serveur reconnaît les pages ASP

Les pages ASP ont une extension `.asp` à la place de `.html` ou `.htm`, et ceci, pour deux raisons. Tout d'abord, pour que le serveur web sache traiter le script de votre page web, il a besoin de savoir que cette dernière en contient. C'est pourquoi, en définissant une extension `.asp` pour votre page web, le serveur va supposer qu'elle contient des scripts.

*Nous pouvons alors constater un effet secondaire intéressant lorsque nous donnons une extension `asp` à nos pages web : le processeur ASP sait qu'il n'a pas besoin de traiter nos fichiers purement en HTML. Auparavant, avec ASP 2.0 par exemple, toute page avec l'extension `.asp`, qu'elle contienne ou non des commandes de script côté serveur, était automatiquement envoyée au serveur et elle nécessitait par conséquent plus de temps pour être traitée. Avec l'introduction de ASP 3.0 dans Windows 2000, le serveur est capable de déterminer la présence de code côté serveur et de traiter ou non la page, ce qui augmente la vitesse de récupération des fichiers HTML purs et optimise le fonctionnement de votre serveur web.*

Ensuite, en utilisant une extension `asp` (ce qui force l'interprétation par le processeur ASP chaque fois que votre page est demandée), vous cachez vos scripts ASP. Si quelqu'un demande votre fichier `.asp` à un serveur web, tout ce qu'il va obtenir en retour est le HTML résultant du traitement. Si vous mettez votre code ASP dans un fichier appelé `mycode.scr` et que vous le demandez à un serveur web, vous verrez tout le code interne.

## Notions de base sur ASP

Les fichiers ASP sont tout simplement des fichiers HTML qui comportent des commandes de script. Quand un navigateur demande un fichier ASP au serveur, celui-ci est passé à la DLL de traitement de l'ASP. Après traitement, le fichier résultant est envoyé au navigateur qui l'a demandé. Toutes les commandes de script incluses dans le fichier HTML de départ sont exécutées, puis supprimées du résultat. Ceci est très positif car le code de vos scripts est caché à la personne qui visualise vos pages web.



C'est pourquoi il est si important que les fichiers qui contiennent des scripts ASP aient une extension `.asp`.

### **Balises ASP**

Pour distinguer le code ASP du HTML à l'intérieur de vos fichiers, le code ASP est placé entre les balises `<%` et `%>`. Cette convention doit vous être familière si vous avez déjà travaillé avec des commandes côté serveur en HTML. Le couple de balises indique au processeur ASP que le code à l'intérieur doit être exécuté par le serveur et supprimé des résultats. En fonction du langage de script par défaut de votre site web, ce code peut être en VBScript, JScript, ou tout autre langage que vous avez installé.

Tous les scripts ASP de cette annexe sont écrits en VBScript, puisqu'il s'agit du langage par défaut.

Dans le morceau suivant de HTML, vous verrez un exemple de code ASP entre les balises `<%` et `%>` :

```
<TABLE>
<TR>
<TD>
<%
    x = x + 1
    y = y - 1
%>
</TD>
</TR>
</TABLE>
```

### **Blocs <SCRIPT>**

Vous pouvez également placer votre code ASP entre les blocs `<SCRIPT></SCRIPT>`. Cependant, si vous ne dirigez pas explicitement l'exécution du script vers le serveur, le code placé entre ces balises sera exécuté sur la machine client comme un script client normal. Pour que votre bloc de script soit exécuté sur le serveur, utilisez la commande `RUNAT` à l'intérieur de `<SCRIPT>` comme ceci :

```
<SCRIPT Language="VBScript" RUNAT="Server">
... Votre Script ...
</SCRIPT>
```

### **Langage de script par défaut**

Comme nous l'avons indiqué précédemment, le langage de script utilisé par défaut par ASP est VBScript. Cela dit, vous pouvez le changer pour la totalité de votre site, ou juste pour une page web particulière, en plaçant une balise de script spéciale au début de votre page web. Cette balise spécifie le langage de script à utiliser pour cette page particulière.

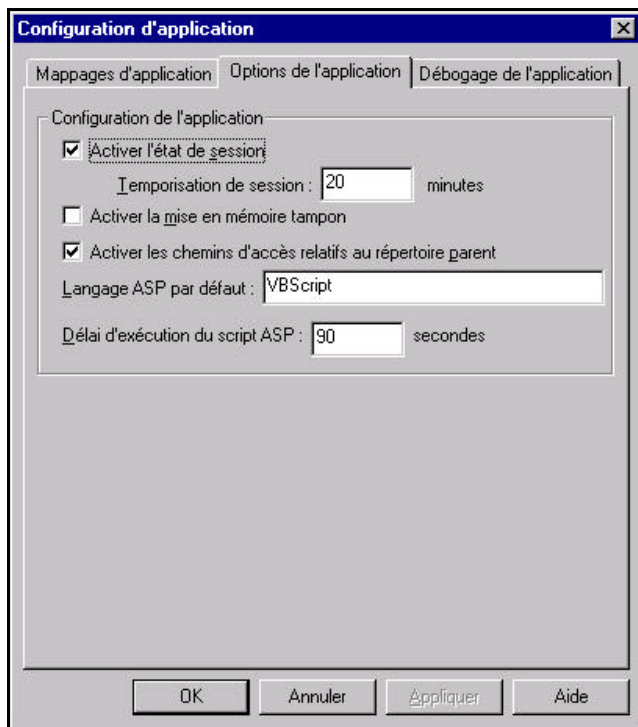
```
<%@ LANGUAGE=Langage de script%>
```

*Langage de script* peut représenter n'importe quel langage pour lequel vous avez un moteur de script installé. ASP intègre JScript et VBScript en standard.

Si vous utilisez les balises `<SCRIPT>`, vous pouvez spécifier le langage à utiliser à l'aide de l'attribut `Language` :

```
<SCRIPT Language="VBScript" RUNAT="Server">
```

Vous pouvez définir le langage de script par défaut pour la totalité de l'application en changeant le champ Langage ASP par défaut dans l'onglet Options de l'application du Gestionnaire de Service Internet.



### **Combiner HTML et ASP**

Vous avez probablement deviné à présent que nous pouvons facilement mélanger du code HTML et des scripts ASP. VBScript possède tous les mécanismes de contrôle de l'exécution comme `If Then`, `For Next`, et les boucles `Do While`. Mais, avec ASP, vous pouvez inclure sélectivement du code HTML en fonction des résultats de ces opérateurs. Examinons un exemple.

Supposez que vous vouliez créer une page web qui salue le visiteur avec « Bonjour », « Bon après-midi », ou « Bonne soirée » en fonction de l'heure. C'est possible de la manière suivante :

```
<HTML>
<BODY>
<P>Il est maintenant<%=Time()%></P>
<%
    Dim iHeure
    iHeure = Hour(Time())
    If (iHeure >= 0 And iHeure < 12 ) Then
%>
Bonjour !
<%
    ElseIf (iHeure > 11 And iHeure < 5 ) Then
%>
Bon après-midi !
<%
    Else
%>
Bonne soirée !
<%
End If
%>
</BODY>
</HTML>
```

Tout d'abord, nous affichons l'heure courante. La notation `<%=` est le raccourci pour imprimer la valeur d'une variable ASP ou le résultat de l'appel à une fonction. Puis nous plaçons l'heure actuelle dans une variable appelée `iHeure`. En fonction de la valeur de cette variable, nous écrivons notre texte HTML normal.

Remarquez le code HTML à l'extérieur des balises de script ASP. Quand le processeur ASP exécute cette page, le HTML inclus dans les blocs de contrôles non exécutés est supprimé, ce qui ne laisse que le code correct. Voici le code source renvoyé par le serveur web après que cette page a été traité :

```
<HTML>
<BODY>
<P>Il est maintenant 19:48:37 heures</P>
Bonne soirée !
</BODY>
</HTML>
```

Comme vous pouvez le voir, le script est complètement supprimé et il ne reste que du HTML et du texte.

L'autre manière d'envoyer des données à votre navigateur est d'utiliser l'un des objets internes de ASP appelé `Response`. Nous verrons cette approche aux paragraphes suivants en étudiant le modèle objet ASP.

### **Commenter votre code ASP**

Comme avec n'importe quel langage de programmation, il est de la plus grande importance de commenter le plus possible votre code ASP. Cependant, combien de fois êtes-vous tombé sur un morceau de code correctement commenté ?

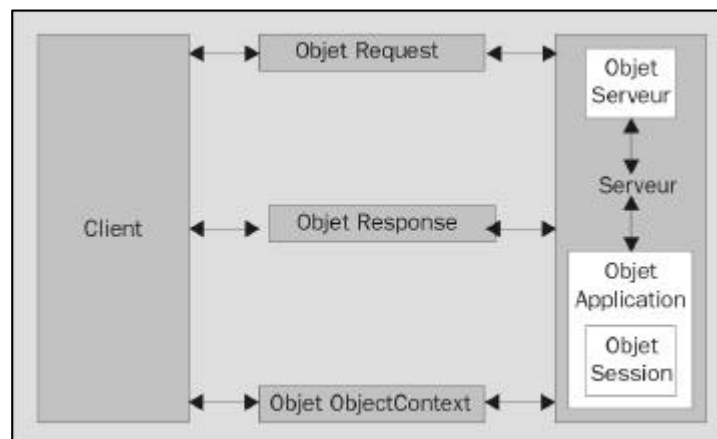
Les commentaires en ASP sont identiques aux commentaires en VBScript. Quand ASP arrive sur le caractère apostrophe simple, il ignore tout simplement le reste de la ligne :

```
<%
Dim iTiti
```

```
'Voici un commentaire  
iTiti = iTiti + 1  
%>
```

## Modèle d'objets ASP

ASP, comme la plupart des technologies Microsoft, utilise COM (Composant Object Model) pour exposer des fonctionnalités aux applications clientes. ASP est en réalité une extension du serveur web qui vous autorise à écrire des scripts côté serveur. Il fournit également un ensemble d'objets et de composants qui gèrent l'interaction entre le serveur web et le navigateur. Ces objets constituent le **modèle d'objets Active Server Pages**. Ces « objets » peuvent être manipulés par des langages de script. Jetez un coup d'œil sur le diagramme suivant :



ASP 2.0 est clairement divisé en six objets qui gèrent chacun une partie de l'interaction entre client et serveur. Comme vous pouvez le voir sur le diagramme, les objets `Request` et `Response`, qui traitent la requête HTTP et la réponse, sont au cœur de l'interaction entre le client et le serveur. Nous allons néanmoins faire un tour rapide de la totalité des objets et composants de ASP.

Le modèle objet est composé de six objets de base, ayant chacun des propriétés et des méthodes distinctes. Ces objets sont les suivants :

- `Request`
- `Response`
- `Application`
- `Session`
- `Server`
- `ObjectContext`

Chacun de ces objets, à l'exception de `Server` et `ObjectContext`, peut utiliser des collections pour stocker des données. Avant d'examiner tour à tour ces objets, nous allons faire une présentation rapide du concept de collection.

## Collections

Les collections ASP sont très semblables à celles de VBScript. Elles agissent en tant que conteneurs de données stockant leurs données d'une

manière proche de celle des tableaux. Les informations sont stockées sous la forme de couples nom/valeur.

Les objets `Application` et `Session` possèdent une « propriété collection » appelée `Contents`. Cette collection de variants peut contenir toutes les informations que vous souhaitez y placer. En utilisant ces collections, vous pouvez partager des informations entre plusieurs pages web.

Pour placer une valeur dans la collection, donnez-lui tout simplement une clé, puis affectez-lui sa valeur :

```
Application.Contents("Nom") = "Jean Luc"
```

OU :

```
Session.Contents("Age") = 36
```

Heureusement pour nous, Microsoft a défini la collection `Contents` comme propriété par défaut de ces deux objets. Par conséquent, l'usage raccourci suivant est parfaitement acceptable :

```
Application("Nom") = "Jean Luc"  
Session("Age") = 36
```

Pour lire des valeurs dans les collections `Contents`, renversez simplement l'appel :

```
sNom = Application("Nom")  
sAge = Session("Age")
```

### ***Parcourir la collection Contents***

Parce que les collections `Contents` fonctionnent comme des collections VBScript classiques, elle peuvent facilement être parcourues. Vous pouvez utiliser la propriété `Count` de la collection ou utiliser une boucle `For Each` :

```
For x = 1 to Application.Contents.Count  
...  
Next  
For each element in Application.Contents  
...  
Next
```

**Notez au passage que les collections `Contents` démarrent à 1. Ce qui revient à dire que le premier élément de la collection est à la position 1, et non 0.**

Pour illustrer ceci, le script ASP suivant affiche le contenu courant des collections `Contents` des objets `Application` et `Session` :

```
<HTML>  
<BODY>  
<P>La collection Application.Contents</P>  
<%  
  Dim element  
  For Each element In Application.Contents  
    Response.Write élément & " = [" & Application(element) & "]<BR>"  
  Next  
&%>  
<P>La collection Session.Contents</P>  
<%  
  For Each element In Session.Contents  
    Response.Write élément & " = [" & Session(element) & "]<BR>"  
  Next  
&%>  
</BODY>  
</HTML>
```

## **Supprimer un élément de la collection Contents**

La collection `Contents` de l'objet `Application` dispose de deux méthodes, `Remove` et `RemoveAll`. Elles vous permettent de supprimer un ou tous les éléments stockés dans la collection `Application.Contents`. Au moment où nous rédigeons ce livre, il n'existe pas de méthode pour supprimer un élément de la collection `Session.Contents`.

Ajoutons un élément à la collection `Application.Contents`, puis supprimons-le.

```
<%  
    Application("MonSigne") = "Balance"  
    Application.Contents.Remove("MonSigne")  
%>
```

Nous pouvons aussi tout simplement nous débarrasser de tout ...

```
<%  
    Application.Contents.RemoveAll  
%>
```

Les collections des autres objets ne fonctionnent pas toutes de cette façon, mais les principes restent les mêmes. Nous expliquerons les différences entre chacune d'entre elles lorsque nous examinerons chaque objet.

## Objet Request

Lorsque votre page web est demandée, de nombreuses informations sont passées avec la requête HTTP, comme, par exemple, l'URL de la page web qui émet la requête et le format des données passées, ou encore des réponses de l'utilisateur entrées dans les zones de saisie ou les listes déroulantes. L'objet `Request` vous permet d'obtenir ces informations jointes à la requête HTTP. La réponse correspondante du serveur est renvoyée à l'intérieur de `Response`. L'objet `Request` dispose de plusieurs collections pour stocker les informations qui constituent l'échange.

### **Collections de l'objet Request**

L'objet `Request` compte cinq collections. Une particularité intéressante est qu'elles sont toutes définies comme propriété par défaut de l'objet. Ce qui revient à dire que vous pouvez récupérer les informations de l'une des cinq collections en utilisant la syntaxe abrégée suivante :

```
AdresseIPClient = Request("REMOTE_ADDR")
```

La valeur `REMOTE_ADDR` fait partie de la collection `ServerVariables`. Cependant, grâce à l'utilisation de la cascade de collections, elle peut être récupérée grâce à la notation ci-dessus. Notez qu'il est inefficace de laisser ASP chercher dans toutes les collections, en particulier si ces dernières possèdent plusieurs valeurs, pour extraire une valeur qui se trouve dans la dernière collection. Il est recommandé de toujours utiliser le nom complet de la collection dans votre code. Ceci est non seulement plus rapide, mais votre code s'en trouve également amélioré puisque la spécification est plus précise et moins cryptée.

ASP effectue des recherches dans les collections selon l'ordre suivant :

- QueryString
- Form
- Cookies
- ClientCertificate
- ServerVariables

Si plusieurs variables portent le même nom, seule la première est renvoyée quand vous faites chercher ASP, ce qui constitue une raison supplémentaire pour toujours préciser complètement votre collection.

### **QueryString**

QueryString contient une collection de toutes les informations ajoutées à la fin d'une URL. Quand vous envoyez l'URL d'une requête, les informations supplémentaires sont ajoutées à la fin de l'URL après un point d'interrogation, sous la forme suivante :

```
URL?item=data[&item=data][...]
```

C'est le point d'interrogation qui sert d'indice pour le serveur. Quand le serveur le voit, il sait que l'URL est terminée et que les variables commencent. Ainsi, un exemple d'URL avec une chaîne de requête (query string) pourrait ressembler à ceci :

```
http://www.achetezcelivre.fr/livre.asp?nomlivre=InitiationXML
```

Nous avons expliqué plus haut que les collections stockent les informations par couples nom/valeur. Malgré la méthode un peu inhabituelle pour créer le couple nom/valeur, le principe reste le même.

nomlivre est le nom et Initiation à XML est la valeur. Quand ASP récupère cette requête URL, il en extrait tous les couples nom/valeur et les place dans cette collection pour en faciliter l'accès. Ceci est une fonctionnalité appréciable d'ASP. Les chaînes de requête sont construites en utilisant des *ampersands* (ou esperluette) pour délimiter chaque couple nom/valeur. Par exemple, si vous souhaitez passer le nom de l'acheteur avec le livre, vous pourriez passer :

```
http://www.achetezcelivre.fr/livre.asp?nomlivre=InitiationaXML&acheteur=ChrisUllman
```

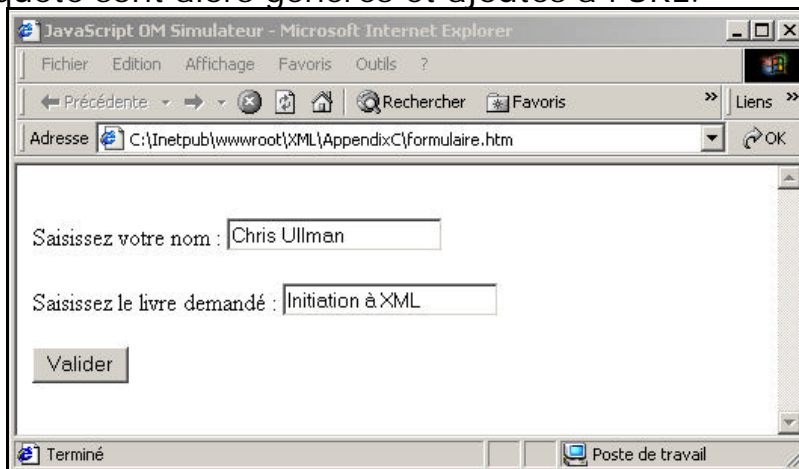
Les chaînes de requête peuvent être générées de trois manières différentes. La première consiste, comme nous l'avons vu, à taper directement l'URL. La deuxième est en tant que partie de l'URL spécifiée dans une balise d'ancrage.

```
<A HREF="livre.asp?nomlivre=InitiationaXML">Aller à la page d'achat</A>
```

Dans ce cas quand vous cliquez sur le lien, le couple nom/valeur est passé avec l'URL. La troisième et dernière méthode utilise un formulaire qui envoie la requête au serveur par le biais de la méthode GET.

```
<FORM ACTION="livre.asp" METHOD="GET">
Saisissez votre nom : <INPUT TYPE="TEXT" NAME="acheteur"><BR>
Saisissez le livre demandé : <INPUT TYPE="TEXT" NAME="nomlivre" SIZE=40><BR>
<INPUT TYPE=SUBMIT VALUE=Valider>
</FORM>
```

Vous entrez les données dans les zones de saisie du formulaire et le texte est envoyé quand vous cliquez sur Valider. Deux morceaux de chaînes de requête sont alors générés et ajoutés à l'URL.



Vous devez ensuite être capable d'extraire ces informations. Vous utilisez pour cela la même technique, quelle que soit la méthode employée pour générer la chaîne de requête.

```
Request.QueryString("acheteur")
Request.QueryString("nomlivre")
```

*Notez que ces lignes n'afficheront rien par elles-mêmes. Vous devez ajouter, soit la notation abrégée (opérateur d'égalité) pour afficher les fonctions en face d'une instruction unique, soit Response.Write pour afficher séparément chaque valeur de la collection quand plusieurs valeurs doivent être affichées.*

Par exemple : `<%=Request.QueryString("acheteur")%>` ou `Response.Write(Request.QueryString("nomlivre"))`

Le premier de ces deux appels à l'objet `Request` doit afficher le nom « Chris Ullman » et le second doit renvoyer « Initiation à XML ». Bien sûr, vous



pouvez toujours stocker cette information dans une variable pour l'utiliser plus tard.

```
SNomlivre = Request.QueryString("nomlivre")
```

### **Form**

Form représente une collection de toutes les variables de formulaires transmises dans la requête HTTP par un formulaire HTML. Les chaînes de requête ne sont pas très discrètes puisqu'elles transmettent les informations par une méthode très visible, l'URL. Si vous souhaitez transmettre les informations depuis le formulaire de manière plus confidentielle, alors vous pouvez utiliser la collection form, car celle-ci envoie ses informations dans le corps de la requête HTTP. La facilité d'accès aux variables des formulaires est l'une des fonctionnalités les plus intéressantes de ASP.

Si nous revenons à notre exemple précédent, la seule modification que nous devons faire au code de notre formulaire HTML est de changer l'attribut METHOD. Les formulaires qui utilisent cette collection doivent être envoyés avec la méthode POST et non la méthode GET. C'est en réalité cet attribut qui détermine la méthode selon laquelle le formulaire envoie les informations. Supposons donc que nous modifiions la méthode du formulaire de la façon suivante :

```
<FORM ACTION="livre.asp" METHOD="POST">  
Saisissez votre nom : <INPUT TYPE="TEXT" NAME="acheteur"><BR>  
Saisissez le livre demandé : <INPUT TYPE="TEXT" NAME="nomlivre" SIZE=40><BR>  
<INPUT TYPE=SUBMIT VALUE=Valider>  
</FORM>
```

Une fois que le formulaire a été soumis de cette manière, nous pouvons extraire et afficher les informations en utilisant la syntaxe suivante :

```
<%=Request.Form("acheteur")%>
```

### **Cookies**

Cookies représente une collection en lecture seule des cookies envoyés par le navigateur du client avec la requête. C'est parce que les cookies ont été envoyés par le client qu'ils ne peuvent pas être modifiés ici. Pour les modifier, vous devez utiliser la collection Response.Cookies. Les cookies sont présentés plus en détail un peu plus loin, lors de l'étude de l'objet Response.

### ClientCertificate

Quand un client se connecte à un serveur qui demande un haut degré de sécurité, chaque partie peut confirmer qui est l'émetteur/récepteur en inspectant son certificat numérique. Un certificat numérique contient plusieurs informations sur l'expéditeur, comme le nom du détenteur, l'adresse et la durée de validité du certificat. Un tiers, connu sous le nom d'Autorité de Certification ou CA, aura préalablement vérifié ces informations.

La collection `ClientCertificate` est utilisée pour accéder aux informations contenues dans un certificat numérique côté client envoyé par le navigateur. Cette collection ne contient des éléments que si votre serveur est sécurisé, et si la requête a été passée par un appel `https://` au lieu d'un appel `http://`. C'est la méthode favorite pour invoquer une connexion sécurisée.

### ServerVariables

Quand le client envoie une requête et que les informations sont passées au serveur, il n'y a pas que la page qui est transmise, mais également des informations telles que le créateur de la page, le nom du serveur et le port auquel a été envoyée la requête. L'en-tête HTTP envoyé avec la requête contient également des informations comme le type de navigateur et le type de connexion. Ces informations sont regroupées dans une liste de variables prédéfinies par le serveur comme variables d'environnement. La plupart d'entre elles sont statiques et ne changent jamais vraiment sauf si vous modifiez la configuration de votre serveur web. Les autres sont liées au navigateur du client.

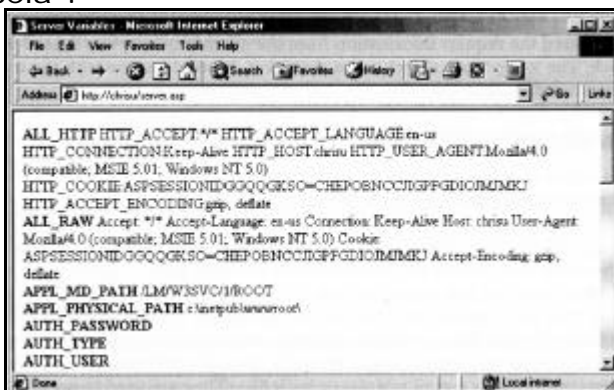
Ces variables serveur sont accessibles par la méthode directe. Par exemple, la variable serveur `HTTP_USER_AGENT`, qui contient des informations sur le type de navigateur utilisé pour visualiser la page, peut être affichée de la manière suivante :

```
<%=Request.ServerVariables("HTTP_USER_AGENT")%>
```

Vous pouvez aussi imprimer la liste complète des variables serveur avec leurs valeurs grâce au code suivant :

```
For Each cle in Request.ServerVariables
    Response.Write "<B>" & (cle) & "</B>&nbsp;"
    Response.Write (Request.ServerVariables(cle)) & "<BR>"
Next
```

Chaque élément de la collection `ServerVariables` est affiché en gras, immédiatement suivi par son contenu, s'il existe. Le produit fini ressemble à cela :



Les variables serveur sont essentiellement informatives, mais elles vous donnent quand même la possibilité de personnaliser le contenu de la page pour des navigateurs spécifiques ou d'éviter les erreurs possibles de script.

### **Propriétés et méthodes de l'objet Request**

L'objet `Request` contient une seule propriété et une seule méthode, qui sont utilisées conjointement pour transférer des fichiers du client vers le serveur. Le téléchargement vers le serveur (*uploading*) est accompli en utilisant des formulaires HTML.

#### **Propriété `TotalBytes`**

Quand la requête sera traitée, cette propriété contiendra le nombre total d'octets de la requête du navigateur du client. L'utilisation la plus fréquente consiste à renvoyer le nombre d'octets du fichier que vous souhaitez transférer. Cette information est importante pour la méthode `BinaryRead`.

#### **Méthode `BinaryRead`**

Cette méthode récupère les informations envoyées au serveur web par le navigateur du client dans une opération `POST`. Quand le navigateur envoie un `POST`, les données sont codées et envoyées au serveur. Quand le navigateur envoie un `GET`, il n'y a pas d'autres données que l'URL. La méthode `BinaryRead` possède un paramètre, le nombre d'octets à lire. Aussi si vous voulez qu'elle lise un fichier complet, vous lui passez le nombre d'octets total du fichier, accessible par la propriété `TotalBytes`.

Cette méthode est très rarement employée parce que `Request.QueryString` et `Request.Form` sont bien plus faciles à utiliser. `BinaryRead` englobe sa réponse dans un tableau sécurisé d'octets, ce qui, pour un langage de script qui gère essentiellement des variants, complique un peu les choses. Cependant, ce format est essentiel pour le transfert de fichiers quand les fichiers contiennent autre chose que du texte pur. Vous pouvez trouver des informations sur la manière de télécharger des fichiers, puis de décoder un tableau d'octets sécurisé dans un excellent article sur [15seconds.com](http://www.15seconds.com/Issue/981121.htm) : <http://www.15seconds.com/Issue/981121.htm>.

## **Objet Response**

Après avoir traité les informations de la requête du navigateur du client, vous devrez être capable de renvoyer des informations. C'est précisément le rôle de l'objet `Response`. Il vous fournit les outils nécessaires pour renvoyer tout ce que vous voulez au client.

### **Collection de l'objet Response**

L'objet `Response` ne contient qu'une collection : `Cookies`. Cette version peut être écrite, contrairement à celle de la collection `Cookies` de l'objet `Request`, dont elle est le pendant.

Si vous ne les avez pas encore rencontrés, les cookies sont de petits fichiers texte (limités à 4ko de données), stockés sur le disque dur du client et qui contiennent des informations sur l'utilisateur, comme par exemple la date de sa dernière visite sur le site. Les cookies suscitent à tort beaucoup d'appréhension. On les dit indiscrets car ils permettent aux serveurs de

stocker des informations sur le disque de l'utilisateur. Cependant, vous devez vous rappeler que, d'une part, l'utilisateur doit volontairement accepter les cookies ou activer un mécanisme d'acceptation sur le navigateur pour qu'ils fonctionnent, et, d'autre part, que ces informations sont complètement bénignes et ne peuvent pas être utilisées pour déterminer l'adresse e-mail de l'utilisateur ou ce type d'informations. Ils sont utilisés pour personnaliser des pages déjà visitées par l'utilisateur. Des exemples de données stockées dans les cookies sont des identifiants utilisateur spécifiques ou des noms d'utilisateur. Ainsi, quand l'utilisateur revient sur votre site web, une vérification rapide des cookies vous permet de savoir s'il est nouveau ou non.

Vous pouvez créer un cookie sur la machine de l'utilisateur de la façon suivante :

```
Response.Cookies("LivreAchete") = "Initiation à XML"
```

Vous pouvez également stocker des valeurs multiples dans un cookie en utilisant une clé d'index. Le cookie contient en effet un objet VBScript Dictionary et, en utilisant la clé, il est possible de récupérer les éléments séparément. Son fonctionnement est très proche de celui d'un tableau.

```
Response.Cookies("LivreAchete")("1") = "Initiation à XML"  
Response.Cookies("LivreAchete")("2") = "Professional XML"
```

Un cookie va automatiquement expirer, c'est-à-dire, disparaître de la machine de l'utilisateur, au moment où ce dernier termine sa session. Pour allonger la durée de vie du cookie, vous pouvez spécifier une date avec la propriété `Expires`. La date utilise le format suivant : `JOURSEM JJ-MM-AA HH:MM:SS`. Par exemple :

```
Response.Cookies("LivreAchete").Expires = #31-Dec-01#
```

*Le signe # peut être utilisé pour délimiter les dates en ASP (comme en VBScript).*

Les propriétés qui peuvent être utilisées en parallèle avec cette collection sont les suivantes :

- ❑ `Domain` : un cookie est envoyé seulement aux pages demandées à l'intérieur du domaine à partir duquel il a été créé ;
- ❑ `Path` : un cookie n'est envoyé qu'aux pages demandées depuis ce répertoire ;
- ❑ `HasKeys` : spécifie si le cookie utilise un objet index/Dictionary ou non ;
- ❑ `Secure` : spécifie si le cookie est sécurisé. Un cookie n'est considéré comme sécurisé que s'il est envoyé par le protocole HTTPS.

Vous pouvez récupérer les informations dans les cookies en utilisant la collection cookies de l'objet `Request`, mentionnée plus haut. Pour cela, vous pouvez faire comme suit :

```
Vous avez effectué un achat <%=Request.Cookies("LivreAchete")%> lors de votre dernière  
visite du site.
```

Si la collection comporte plusieurs cookies, vous pouvez faire une boucle et afficher le contenu de chaque cookie comme ceci :

```
For Each cookie in Request.Cookies  
Response.Write (Request.Cookies(cookie))  
Next
```

## Méthodes de l'objet Response

Pour comprendre ce que font les méthodes et les propriétés de l'objet `Response`, nous devons examiner dans le détail la manière dont ASP envoie une réponse. Quand un script ASP est exécuté, un **flux de sortie HTML** est créé. Ce flux est un réceptacle pour le serveur web qui y stocke les informations et crée la page web dynamique/interactive. Comme nous l'avons vu, la page doit être créée entièrement en HTML pour que le navigateur la comprenne (en excluant les scripts côté client, ignorés par le serveur).

Le flux est vide au moment de sa création. De nouvelles informations sont ajoutées à la fin. Si des en-têtes HTML personnalisés sont requis, alors, ils doivent être ajoutés au début. Puis, le HTML contenu dans la page ASP est ajouté à côté du script. Donc, tout ce qui n'est pas à l'intérieur de balises `<% %>` est ajouté. L'objet `Response` fournit deux moyens d'écrire directement dans le flux de sortie, soit en utilisant la méthode `Write`, soit sa version abrégée.

### Write

`Write` vous permet de renvoyer des informations au navigateur du client, il s'agit probablement de la méthode la plus utilisée parmi toutes celles des objets internes. Vous pouvez écrire du texte directement sur une page web en mettant le texte entre guillemets :

```
Response.Write "Bonjour à tous !"
```

Pour afficher le contenu d'une variable de type variant, vous enlevez les guillemets :

```
sTexte = "Bonjour à tous !"  
Response.Write sTexte
```

Pour des portions isolées d'informations dynamiques qui doivent être simplement ajoutées à de grandes portions de HTML, vous pouvez utiliser le signe égal comme version abrégée de cette méthode, comme nous l'avons mentionné plus haut, à savoir :

```
Mon message est <%=sTexte %>
```

Cette technique réduit la quantité de code nécessaire, mais au prix de la lisibilité. Les deux techniques sont équivalentes en termes de performance.

### AddHeader

Cette méthode vous permet d'ajouter des en-têtes personnalisés à la réponse HTTP. Par exemple, si vous deviez écrire une application navigateur personnalisée qui vérifie la présence d'une certaine valeur dans les en-têtes de vos requêtes HTTP, vous utiliseriez cette méthode pour définir cette valeur. L'usage en est le suivant :

```
Response.AddHeader "ServiceClient", "Application/1.0"
```

Cette instruction ajouterait à la réponse un en-tête `ServiceClient` ayant une valeur de `Application/1.0`. Il n'y a aucune restriction concernant les en-têtes et les valeurs d'en-têtes.



La méthode `Flush` est utilisée avec la propriété `Buffer`. Pour l'utiliser correctement, vous devez d'abord définir la propriété `Buffer` puis, à certains endroits à l'intérieur du script, vous pouvez vider le cache vers le flux de sortie, tout en continuant le traitement. Ceci est utile dans le cas de longues requêtes qui pourraient laisser penser à l'utilisateur que rien n'a été renvoyé.

La méthode `clear` efface tout ce qui a été ajouté dans le cache depuis le dernier appel à `Response.Flush`. Cependant, elle n'efface que le corps de la réponse et laisse intact son en-tête.

#### **CacheControl**

Généralement quand un serveur proxy récupère une page web ASP, il n'en place pas de copie dans son cache. Ceci s'explique par le fait que les pages ASP sont par nature dynamiques et qu'il y a de fortes chances pour qu'une page soit périmée quand elle est redemandée. Vous pouvez déconnecter cette fonctionnalité en plaçant la valeur de cette propriété sur `Public`.

#### **Charset**

Cette propriété ajoutera son contenu à l'en-tête HTTP « type de contenu » qui est renvoyé au navigateur. Toute réponse HTTP possède un en-tête « type de contenu » qui définit le contenu de la réponse. Habituellement, le type de contenu est "text/html". Définir cette propriété modifiera le type renvoyé au navigateur.

#### **ContentType**

Cette propriété vous permet de définir la valeur du type de contenu renvoyé au navigateur client.

#### **Expires**

La plupart des navigateurs conservent les pages web dans un cache local. Le cache est habituellement valide aussi longtemps que le navigateur tourne. En définissant cette propriété, vous pouvez limiter le temps pendant lequel la page reste dans le cache local. La valeur de la propriété `Expires` spécifie le délai en minutes avant que la page n'expire du cache local. Si vous lui affectez zéro, la page n'aura pas de cache.

#### **ExpiresAbsolute**

Tout comme la propriété `Expires`, cette propriété vous permet de spécifier la date et l'heure exactes auxquelles la page expirera.

#### **IsClientConnected**

Cette propriété en lecture seule indique si le client est toujours connecté au serveur. Vous vous rappelez peut-être que le navigateur client émet une requête puis attend une réponse. Imaginez que vous exécutiez un script long et qu'au milieu du traitement, le client se déconnecte parce qu'il juge avoir attendu trop longtemps. En lisant cette propriété, vous saurez si le client est toujours connecté ou non. Malheureusement, avec ASP 2.0, cette propriété semble ne pas fonctionner correctement et elle n'a été réparée que sous ASP 3.0 dans Windows 2000.

#### **Status**

Cette propriété vous permet de définir la valeur renvoyée dans l'en-tête « status » de la réponse HTTP.

## Objets Application et Session

Les objets `Application` et `Session`, tout comme `Request` et `Response`, travaillent en commun. `Application` est utilisé pour lier toutes les pages en une application consistante alors que l'objet `Session` est utilisé pour tracer et présenter au site web une série de requêtes de l'utilisateur comme une action continue et non pas comme un ensemble arbitraire de requêtes.

### ***La portée est éternelle***

En temps normal, vous déclarez une variable pour l'utiliser à l'intérieur de votre page web. Vous l'utilisez, la manipulez, puis peut-être imprimez-vous sa valeur. Toutefois, quand votre page est chargée à nouveau ou que le navigateur passe à une autre page, la variable est irrémédiablement perdue. En plaçant votre variable à l'intérieur de la collection `Contents` des objets `Application` ou `Session`, vous pouvez allonger sa durée de vie !

Toute variable ou tout objet que vous déclarez possède deux portées possibles : procédure ou page. Quand vous déclarez une variable à l'intérieur d'une procédure, sa durée de vie est limitée à cette procédure. Une fois que la procédure est exécutée, votre variable disparaît. Vous pouvez également déclarer une variable au niveau de la page web, mais, tout comme avec la variable définie au niveau de la procédure, quand la page est rechargée, la valeur est réinitialisée.

C'est ici qu'entrent en jeu les objets `Application` et `Session`. Les collections `Contents` de ces deux objets vous permettent d'étendre la portée de vos variables à la session ou à l'application. Si vous placez une valeur dans l'objet `Session`, elle sera à la disposition de toutes les pages web de votre site pendant toute la durée de vie de la session en cours (les sessions seront expliquées en détail plus bas). Des variables typiques ayant la session comme durée de vie sont les identifiants utilisateur, les noms d'utilisateur, l'heure de la connexion, etc, toutes choses qui n'appartiennent qu'à la session. De la même façon, si vous placez votre valeur dans l'objet `Application`, elle existera jusqu'à ce que le site web soit redémarré. Ceci vous permet de placer des définitions du niveau de l'application dans un endroit facilement accessible. Des variables typiques ayant l'application comme durée de vie sont les noms et les tailles des polices, les couleurs des tables, les constantes système, toutes choses relatives à l'application dans son ensemble.

### ***Fichier global.asa***

Toute application ASP peut utiliser un fichier de scripts spécial. Ce fichier est appelé `global.asa` et il doit résider dans le répertoire racine de votre application web. Il peut contenir des scripts relatifs à l'application dans son ensemble, ou à chaque session. Vous pouvez également créer dans ce fichier des objets ActiveX pour les utiliser ultérieurement.

### ***Objet Application***

ASP est basé sur le concept qu'un site web complet est constitué par une application web unique. Par conséquent, il n'y a jamais qu'une seule



instance de l'objet `Application` à la disposition de vos scripts. Notez qu'il est possible de diviser votre site web en applications séparées. Toutefois, dans l'objectif de notre discussion, nous supposons qu'il n'y a qu'une application par site web.

### **Collections**

L'objet `Application` comporte deux collections : `Contents` et `StaticObjects`. La collection `Contents` est présentée ci-dessus. La collection `StaticObjects` est similaire à `Contents`, mais ne contient que les objets qui ont été créés avec la balise `<OBJECT>` dans la portée de votre application. Cette collection peut être parcourue tout comme la collection `Contents`.

*Vous ne pouvez pas stocker de références aux objets internes d'ASP dans les collections `Application`.*

### **Méthodes**

L'objet `Application` contient deux méthodes détaillées ci-dessous .

Lock	La méthode <code>Lock</code> est utilisée pour verrouiller la collection <code>Contents</code> afin qu'elle ne puisse pas être modifiée par d'autres clients. Ceci est utile, par exemple, si vous mettez à jour un compteur, ou un numéro de transaction, stocké dans la collection <code>Contents</code> d' <code>Application</code> .
Unlock	La méthode <code>Unlock</code> déverrouille l'objet <code>Application</code> , ce qui permet ainsi à d'autres de modifier la collection <code>Contents</code> .

### **Événements**

L'objet `Application` génère deux événements : `Application_OnStart` et `Application_OnEnd`. L'événement `Application_OnStart` est déclenché à la première visite de votre page web. L'événement `Application_OnEnd` est déclenché quand le serveur web est arrêté. Si vous choisissez d'écrire des scripts pour ces événements, ils doivent être placés dans votre fichier `global.asa`.

L'utilisation la plus commune de ces événements consiste à initialiser les variables globales de l'application, telles que les noms des polices, les couleurs des tables, les chaînes de connexion aux bases de données, ou alors à écrire des informations dans un fichier journal système. Le code qui suit est un exemple de fichier `global.asa` avec des scripts pour ces événements :

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart
  'Variables globales...
  Application("PageErreur") = "InterceptorErreur.asp"
  Application("limiteTentativeAccesSite") = 10
  Application("AccesPageErreur") = "InterceptorErreur.asp"
  Application("AccesLimite") = False
  'Mémoriser les visites...
  Application("NumVisites") = Application("NumVisites") + 1
End Sub
</SCRIPT>
```

### **Objet Session**

Chaque fois qu'un visiteur vient sur votre site web, un objet `Session` est créé pour ce visiteur, à moins qu'il n'en ait déjà un. Par conséquent, il y a aussi une instance de l'objet `Session` à votre disposition dans votre script. L'objet `Session` est similaire à l'objet `Application` par le fait qu'il peut

contenir des valeurs. Cependant, les valeurs de l'objet `Session` sont perdues quand votre visiteur quitte le site. L'objet `Session` est particulièrement utile pour transférer des informations de page web en page web. En utilisant l'objet `Session`, nul besoin de passer les informations dans l'URL.

L'utilisation la plus courante de l'objet `Session` consiste à stocker des informations dans sa collection `Contents`. Ces informations sont spécifiques à la session puisqu'elles ne concernent que l'utilisateur en cours.

De nombreux sites web offrent de nos jours un service de « personnalisation utilisateurs » qui permet aux utilisateurs de personnaliser une page web en fonction de leurs préférences. Cette fonction est facilement mise en œuvre avec ASP et l'objet `Session`. Les variables utilisateur sont stockées dans le navigateur client pour que le serveur les récupère plus tard. Chargez simplement les préférences de l'utilisateur au début de la session puis, quand l'utilisateur navigue dans votre site, utilisez les informations concernant ses préférences pour afficher des informations.

Supposez que votre site web affiche les cours de la Bourse. Vous pourriez autoriser les utilisateurs à personnaliser la page de départ pour afficher leurs actions favorites quand ils visitent le site. En stockant les symboles de ces actions dans votre objet `Session`, vous pouvez facilement afficher les titres voulus quand vous renvoyez la page web.

Le système de gestion des sessions utilise les cookies du navigateur. Les cookies permettent aux informations de l'utilisateur de persister même après que le client a quitté le site. Malheureusement, si un visiteur de votre site web n'autorise pas l'utilisation des cookies, vous ne pourrez pas passer d'informations entre les pages web à l'intérieur de l'objet `Session`.

### **Collections**

L'objet `Session` contient deux collections : `Contents` et `StaticObjects`. Nous avons présenté la collection `Contents` ci-dessus. La collection `StaticObjects` est similaire à `Contents`, mais ne contient que les objets qui ont été créés avec la balise `<OBJECT>` dans votre page HTML. Cette collection peut être parcourue tout comme la collection `Contents`.

### **Propriétés**

Voici les propriétés que l'objet `Session` vous permet d'utiliser :

Propriété	Description
<code>CodePage</code>	En définissant cette propriété, vous pourrez changer le jeu de caractères utilisé par ASP quand il génère une sortie. Cette propriété peut être utilisée si vous créez un site web multinational.

(Suite du tableau)

Propriété	Description
LCID	<p>Cette propriété définit l'identificateur de paramètres régionaux pour la totalité de l'application web. Par défaut, la région de votre application est celle de votre serveur. Si votre serveur est aux États-Unis, votre application aura le paramètre américain par défaut. La plupart des fonctionnalités de formatage d'ASP utilisent cette définition de région pour afficher les informations correctement pour le pays en question. Par exemple, la date est affichée différemment en Europe et aux États-Unis. Par conséquent, les fonctions de formatage de la date basées sur la définition de la région sortiront la date dans le format correct.</p> <p>Vous pouvez également changer cette propriété de façon temporaire pour sortir les données dans un format différent. La devise est un bon exemple. Supposons que votre site web possède un panier à commissions et que vous vouliez afficher les totaux en dollars américains pour les clients américains et en francs français pour les clients français. Dans ce but, vous auriez à définir la propriété LCID sur la région française avant d'appeler la routine de formatage des devises.</p>
SessionID	Chaque session créée par ASP a un identifiant unique. Cet identifiant est appelé SessionID et il est accessible au travers de cette propriété. On peut l'utiliser pour déboguer des scripts ASP.
Timeout	Par défaut, les sessions ASP seront déconnectées automatiquement après 20 minutes d'inactivité. Chaque fois qu'une page web est demandée ou actualisée par l'utilisateur, son horloge interne ASP se déclenche. Quand l'horloge atteint la valeur définie dans cette propriété, la session est automatiquement détruite. Vous pouvez également utiliser cette propriété pour réduire la durée d'immobilisation.

### **Méthode**

L'objet `Session` contient une seule méthode, `Abandon`, qui ordonne à ASP de détruire l'objet `Session` en cours pour cet utilisateur. Cette méthode est typiquement appelée au moment où un utilisateur se déconnecte de votre site web.

### **Événements**

L'objet `Session` entraîne deux événements : `Session_OnStart` et `Session_OnEnd`. L'événement `Session_OnStart` est déclenché à la première visite de votre page web. L'événement `Session_OnEnd` est déclenché quand la session est arrêtée. Si vous choisissez d'écrire des scripts pour ces événements, ils doivent être placés dans votre fichier `global.asa`.

L'utilisation la plus courante de ces événements est l'initialisation des variables de session. Il s'agit d'éléments comme les comptes courants, les noms de connexion, les noms réels, les options de l'utilisateur, etc. Ce qui suit est un exemple de fichier `global.asa` avec des scripts pour ces événements :

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Session_OnStart
    Session("tentativesconnexion") = 0
    Session("estconnecte") = False
End Sub
Sub Session_OnEnd
    Session("estconnecte") = False
End Sub
</SCRIPT>
```

## **Objet Server**

L'objet suivant dans le modèle objet de ASP est l'objet `Server`. L'objet `Server` vous permet de créer des contrôles ActiveX et de travailler dans

vos pages web. De plus, l'objet `Server` expose des méthodes qui facilitent le codage des URL et du texte HTML.

## Propriété

### *ScriptTimeout*

Cette propriété définit le temps en secondes pendant lequel un script est autorisé à tourner. La valeur par défaut pour tous les scripts du système est de 90 secondes. Sinon, lorsqu'un script est en cours depuis plus de 90 secondes, le serveur web intervient et fait savoir au navigateur client que quelque chose ne va pas. Si vous prévoyez de faire s'exécuter vos scripts pendant une longue période de temps, vous devrez utiliser cette propriété.

## Méthodes

### *CreateObject*

Cette méthode est l'équivalent de `CreateObject` de VBScript. Accessoirement, il est possible d'utiliser le mot clé `New`. Cette méthode crée une nouvelle instance de l'objet passé en paramètre. Le résultat peut être placé dans la collection `Contents` de `Application` ou `Session` pour allonger sa durée de vie.

Généralement vous allez créer un objet au moment où la session est créée et le placer dans la collection `Session.Contents`. Par exemple, supposons que vous ayez créé une DLL ActiveX «killerDLL» avec une classe très pratique qui convertit les degrés Fahrenheit en degrés Celsius et vice versa. Vous pourriez créer une instance de cette classe avec la méthode `CreateObject` et la stocker dans la collection `Session.Contents` de la façon suivante :

```
Set Session("MonConvertisseur") =  
Server.CreateObject("KillerDLL.ConvertisseurCelsius")
```

Cet objet existera aussi longtemps que la session et il sera à votre disposition dès que vous souhaiterez l'appeler. Comme vous le verrez dans les derniers chapitres, cette méthode est inestimable lors d'un travail avec les connexions aux bases de données.

ASP possède son propre ensemble de composants dont vous pouvez créer des instances en utilisant la méthode `CreateObject`. Les voici page suivante.

- ❑ **AdRotator** : utilisé pour afficher un graphique aléatoire avec un lien chaque fois qu'un utilisateur se connecte à la page ;
- ❑ **Browser Capabilities** : manipule un fichier `browscap.ini` contenu sur l'ordinateur du serveur afin de déterminer les possibilités du navigateur d'un client particulier ;
- ❑ **Content Linker** : fournit un fichier annuaire central à partir duquel vous gérez une série de liens, leurs URLs, et leurs descriptions détaillées ;
- ❑ **ContentRotator** : version allégée de Ad Rotator qui fournit la même fonction mais sans la redirection en option ;
- ❑ **PageCounter** : compte le nombre de fois où une page a été appelée (hit) ;
- ❑ **PermissionChecker** : vérifie si un utilisateur a les autorisations nécessaires avant de lui donner l'accès à une page donnée ;
- ❑ **Counters** : compteur quelconque dans une page web, disponible n'importe où à l'intérieur de l'application ASP ;
- ❑ **MyInfo** : utilisé pour stocker d'éventuelles informations personnelles sur un utilisateur dans le cadre d'un fichier XML ;

- ❑ **Status** : utilisé pour collecter les informations relatives au profil du serveur ;
- ❑ **Tools** : ensemble de méthodes diverses groupées sous la terminologie générique d' « Outils » ;
- ❑ **IISLog** : permet de créer un objet qui autorise vos applications à écrire ou accéder au journal d'IIS.

#### **Execute**

Cette méthode exécute un fichier ASP et insère les résultats dans la réponse. Vous pouvez utiliser cet appel pour inclure des morceaux de code ASP, comme des sous-routines.

#### **GetLastError**

Cette méthode renvoie un objet `ASPErrors` qui contient toutes les informations sur la dernière erreur survenue.

#### **HTMLEncode**

Cette méthode convertit une chaîne en HTML correct, ce qui est utile si vous voulez afficher du code HTML sur vos pages web.

#### **MapPath**

Cette méthode renvoie une chaîne qui contient le chemin physique réel vers le fichier en question. Les sous-répertoires de votre site web peuvent être virtuels. Ceci revient à dire qu'ils n'existent pas physiquement dans la hiérarchie de votre site web. Vous pouvez appeler cette méthode pour trouver l'emplacement réel d'un fichier.

#### **Transfer**

La méthode `Transfer` vous permet de transférer immédiatement le contrôle de la page qui s'exécute à une autre page. Elle est similaire à la méthode `Response.Redirect`, à une exception près : la méthode `Transfer` met toutes les variables et les collections de `Request` à la disposition de la page appelée.

#### **URLEncode**

Cette méthode, comme son nom l'indique, chiffre une URL avant la transmission. Ce codage consiste à remplacer les espaces par un signe plus (+) et les caractères non-imprimables par des valeurs hexadécimales. Vous devez toujours exécuter vos URL avec cette méthode lorsque vous utilisez la redirection.

## **ObjetObjectContext**

Le dernier objet que nous allons aborder est l'objet `ObjectContext`, qui est employé quand vous utilisez des transactions dans votre page web. Quand un script ASP a initié une transaction, celle-ci peut être, soit confirmée, soit interrompue par cet objet. Il possède pour cela deux méthodes :

#### **SetAbort**

`SetAbort` est appelée quand la transaction n'a pas été confirmée et que vous voulez que les ressources ne soient pas exploitées.

#### **SetComplete**

`SetComplete` est appelée quand il n'y a pas de raison pour que la transaction échoue. Si tous les composants qui forment la transaction appellent `SetComplete`, la transaction sera exécutée.

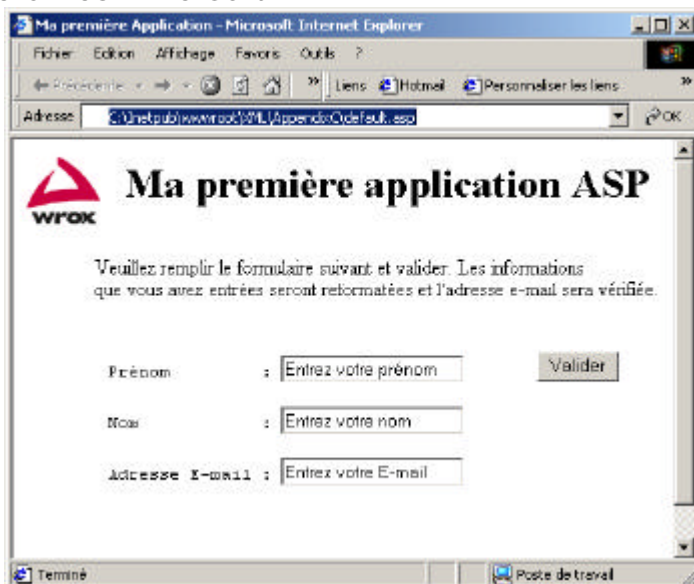
# Utiliser Active Server Pages de façon efficace

Dans la dernière partie de cette annexe, nous allons construire un site web pour effectuer une démonstration des fonctionnalités de ASP. Ce site de démonstration illustrera plusieurs des fonctionnalités et des principes de ASP décrits plus haut dans ce chapitre.

## Concevoir le site

Avant de commencer à créer notre nouveau site web, nous devons parler de sa conception. Pour votre première application ASP, nous resterons relativement simples. Nous allons créer un formulaire HTML qui accepte la saisie des informations suivantes : prénom, nom, et adresse email. Quand l'utilisateur enverra le formulaire, notre page ASP va reformater le prénom et le nom, et vérifier que l'adresse email a une syntaxe correcte.

L'utilisateur aura droit à trois essais pour entrer les informations correctement ou bien un message d'erreur s'affichera en bas de l'écran comme suit.



## Créer le fichier global.asa

La première étape dans la création d'une nouvelle application ASP consiste à créer votre fichier `global.asa`. C'est le fichier qui héberge votre gestionnaire d'événements pour les objets `Application` et `Session`. De plus, vous pouvez définir dans ce fichier des variables du niveau de l'application et de la session et leurs valeurs par défaut. Pour créer ce fichier dans la racine du répertoire de votre serveur web, créez un fichier appelé `global.asa`. Voici le contenu de notre `global.asa` :

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart
    Application("ErreursPermesAvantAvertissement") = 3
End Sub
Sub Session_OnStart
    Session("ComptageErreurs") = 0
End Sub
Sub Session_OnEnd
    Opération inconnue...
End Sub
Sub Application_OnEnd
```

```
Opération inconnue...  
End Sub  
</SCRIPT>
```

Notre fichier définit des gestionnaires d'événements pour `Application_OnStart`, `Application_OnEnd`, `Session_OnStart` et `Session_OnEnd`. Les événements `Application_OnEnd` et `Session_OnEnd` ne sont pas utilisés dans cet exemple mais ils apparaissent ci-dessus afin que notre propos soit le plus complet possible.

Nous allons définir un nombre d'essais limite accordés à l'utilisateur avant que ne s'affiche un message d'erreur. Comme c'est une fonctionnalité de l'application qui affecte tous les utilisateurs, nous stockerons cette constante dans la collection `Application.Contents`. C'est ce qui est réalisé dans l'événement `Application_OnStart`. Nous ajoutons à la collection un élément appelé `ErreursPermesAvantAvertissement` et initialisons sa valeur à 3.

Maintenant que nous savons combien de fois *maximum* un utilisateur peut tenter son opération de saisie, nous avons besoin d'un emplacement pour stocker le nombre réel de fois où l'utilisateur a tenté son opération. Comme ce compteur est différent pour chaque utilisateur, nous le placerons dans la collection `Session.Contents`. Nous initialisons notre variable à 0. C'est ce qui est réalisé dans l'événement `Session_OnStart`: nous ajoutons à la collection un élément `ComptageErreurs` dont la valeur est 0.

## Créer la page principale

Maintenant que nous avons établi les fondations de notre application ASP, il est temps de construire la page principale. Comme c'est un exemple simple, nous n'utiliserons qu'une seule page web. Nous allons commencer par créer cette page.

Créez une nouvelle page web sur votre site et appelez-la `default.asp`. C'est le nom de fichier utilisé par IIS comme page web par défaut. La page web par défaut est celle qui est renvoyée par un serveur web quand aucune page web n'est spécifiée. Par exemple, quand vous appelez `http://www.wrox.fr/`, vous ne spécifiez pas de page web. Le serveur examine sa liste de noms de fichiers par défaut et trouve le premier qui correspond dans le répertoire racine du site web.

La page est assez longue. Toutefois, elle se décompose logiquement en deux sections distinctes : la portion ASP/VBScript et la portion HTML. Nous allons examiner ces sections l'une après l'autre.

## Section ASP/VBScript

C'est dans la moitié supérieure de notre fichier que réside le code ASP. C'est ce code qui est exécuté par le serveur avant que la page ne soit renvoyée au navigateur qui la demande. Rappelez-vous que tout le code qui doit être exécuté sur le serveur avant le renvoi de la réponse est délimité par les balises particulières `<%` et `%>`.

Pour plus de clarté, le code ASP a été divisé en sous-routines. Ceci permet, non seulement, de rendre le code plus lisible, mais aussi d'améliorer son caractère réutilisable. Notre code a deux routines : `Main` et `InitCap`.

Avant de faire quoique ce soit d'autre, nous déclarons des variables :

```
<%@ Language=VBScript %>
<%
Dim txtPrenom, txtNom, txtEmail
Dim sMessage
```

Quand les variables sont déclarées en dehors d'une sous-routine dans une page ASP, elles conservent leurs valeurs jusqu'à ce que la page soit complètement traitée. Ceci vous permet de passer des informations de votre code ASP à votre code HTML, comme nous allons le voir.

Après avoir déclaré nos variables, nous obtenons notre routine Main. C'est elle qui est appelée par notre code ASP chaque fois qu'un navigateur charge la page. La sous-routine Main n'est pas appelée automatiquement : nous devons le faire nous-mêmes de façon explicite.

```
'*****
'* Main
'*
'* La sous-routine main de cette page...
'*****
Sub Main()
  ' Cette page a-t-elle déjà été soumise ?
  if ( Request("cmdEnvoyer") = "Valider" ) Then
    'Reformater les données dans un format plus lisible...
    txtPrenom = InitCap(Request("txtPrenom"))
    txtNom = InitCap(Request("txtNom"))
    txtEmail = LCase(Request("txtEmail"))
    'Vérifier que l'adresse email est correcte...
    if ( Instr(1, txtEmail, "@") = 0 or Instr(1, txtEmail, ".") = 0 ) Then
      sMessage = " L'adresse email que vous avez entrée est invalide."
    Else
      'Vérifier qu'il y a quelque chose après le point...
      if ( Instr(1, txtEmail, ".") = Len(txtEmail) _
        or Instr(1, txtEmailAddr, "@") = 1 or _
        (Instr(1, txtEmail, ".") = Instr(1, txtEmail, "@") + 1) ) Then
        sMessage = "Vous devez entrer une adresse email complète."
      end if
    End If
    La validation s'est bien passée. Donc, tout va bien...
    if ( sMessage = "" ) Then
      sMessage = "Merci. Toutes les données ont été vérifiées avec succès."
    else
      Session("ComptageErreur") = Session("ComptageErreur") + 1
      if ( Session("ComptageErreur") >
Application("ErreursPermesesAvantAvertissement") ) then
        sMessage = sMessage & "<P><Font Size=1>Vous avez atteint le nombre d'essais
maximal!</Font>"
      end if
    End If
  Else
    'S'il s'agit du premier passage, définir certaines valeurs par défaut...
    txtPrenom = "Entrez votre prénom"
    txtNom = "Entrez votre nom"
    txtEmail = "Entrez votre E-mail"
  End If
End Sub
```

Nous regardons tout d'abord si le formulaire a déjà été soumis par l'utilisateur. Sinon, nous initialisons nos variables. Pour déterminer si la page a été soumise, nous vérifions la valeur de la variable cmdEnvoyer de Request. C'est le bouton de notre formulaire. Lorsque celui-ci est cliqué, le formulaire appelle cette page et donne au bouton cmdEnvoyer la valeur Valider. Si un utilisateur charge uniquement la page sans presser le bouton,



la valeur de `cmdEnvoyer` est vide (""). Il y a d'autres méthodes pour déterminer si une page web a été soumise, mais celle-ci est la plus simple.

Après avoir déterminé si la page a bien été soumise, nous faisons passer les noms saisis par la seconde fonction de notre page : `InitCap`. `InitCap` est une petite fonction rapide qui formate correctement les majuscules et minuscules d'un mot. En substance, la première lettre est mise en majuscule et le reste du mot en minuscules. Voici la fonction :

```
'*****  
'* InitCap  
'*  
'* Mettre la première lettre de la chaîne en majuscule  
'*****  
Function InitCap(sStr)  
    InitCap = UCase(Left(sStr, 1)) & LCase(Right(sStr, Len(sStr) - 1))  
End Function
```

Après avoir nettoyé les noms, nous devons vérifier la validité de l'adresse email. Pour cela, nous nous assurons qu'elle contient un signe "@" et un point (.). Une fois ce contrôle effectué, nous nous assurons qu'il y a des données après le point et avant le signe "@". Il s'agit là d'un contrôle de validité « vite fait bien fait » des e-mails !

Si l'un de ces contrôles échoue, nous plaçons un message d'erreur dans la chaîne `sMessage`, qui est affichée par notre section HTML à la fin du traitement de la page.

Si notre adresse e-mail réussit le test, nous initialisons le message (`sMessage`) pour qu'il affiche une note de remerciement. Dans le cas inverse, nous incrémentons le compteur d'erreurs que nous avons défini dans le fichier `global.asa`. Nous vérifions également si nous avons dépassé notre limite d'erreurs autorisées. Si c'est le cas, un message approprié est défini pour l'affichage.

Enfin, la dernière partie de notre section ASP est l'appel à `Main`, exécuté quand la page est chargée :

```
'*****  
'* Appeler notre routine principale  
'*****  
Call Main()
```

## Section HTML

Cette section est un formulaire HTML classique saupoudré d'une bonne pincée d'ASP. Le code ASP inclus dans cette section HTML définit des valeurs par défaut pour les champs d'entrée et affiche tous les messages générés par notre code côté serveur.

```
<HTML>  
<HEAD>  
    <META NAME="GENERATOR" Content="Microsoft FrontPage 3.0">  
    <TITLE>Ma première application ASP</TITLE>  
</HEAD>  
<BODY>  
<TABLE border="0" cellPadding="0" cellSpacing="0" width="600">  
<TBODY>  
<TR>  
    <TD width="100"><a href="http://www.wrox.fr" target="_blank"  
        border=0 alt><IMG border=0 title="Rendez-vous sur le site web de WroxPress!"  
        src="images/wroxlogo.gif" WIDTH="56" HEIGHT="56"></A></TD>
```

```

<TD width="500"><CENTER><FONT size="5" face="Trebuchet MS">Ma première
  application ASP </FONT></CENTER></TD>
</TR>
<TR>
  <TD width="100">&nbsp;</TD>
  <TD width="500" align="left"><FONT face="Trebuchet MS"><BR>
  Veuillez remplir le formulaire suivant et valider. Les informations que
  vous avez entrées seront reformatées et l'adresse email sera
  vérifiée.</FONT>
  <FORM action="default.asp" id="FORM1" method="post" name="frmMain">
    <TABLE border="0" cellPadding="1" cellSpacing="5" width="100%">
      <TR>
        <TD width="100" nowrap align="right"><FONT size="2" face="Trebuchet
          MS">Prénom : </FONT></TD>
        <TD width="350"><font size="2" face="Trebuchet MS">
          <INPUT title="Entrez votre prénom" name="txtPrenom"
            size="30" value="<%=txtPrenom%" tabindex="1"></FONT></TD>
        <TD width="50"><div align="right"><font size="2" face="Trebuchet MS">
          <INPUT type="Valider" title="Soumettre ces données au traitement..."
            value="Valider" name="cmdEnvoyer" tabindex="4"></FONT></TD>
      </TR>
      <TR>
        <TD width="100" nowrap align="right">
          <FONT size="2" face="Trebuchet MS">Nom : </FONT></TD>
        <TD width="400" colspan="2">
          <FONT size="2" face="Trebuchet MS">
          <INPUT title="Entrez votre nom" name="txtNom"
            size="30" value="<%=txtNom%" tabindex="2"></FONT></TD>
      </TR>
      <TR>
        <TD width="100" nowrap align="right">
          <FONT size="2" face="Trebuchet MS">Email : </FONT></TD>
        <TD width="400" colspan="2"><FONT size="2" face="Trebuchet MS">
          <INPUT title="Entrez votre E-mail"
            name="txtEmail" size="40" value="<%=txtEmail%"
            tabindex="3"></font></td>
      </TR>
      <TR>
        <TD nowrap width=500 colspan="3" align="center">
          <FONT face="Trebuchet MS"><BR>
          <STRONG><%=sMessage%></STRONG> </FONT></TD>
      </TR>
    </TABLE>
  </FORM>
  <P>&nbsp;</TD>
</TR>
</TBODY>
</TABLE>
</BODY>
</HTML>

```

La partie la plus importante de ce code HTML est située à l'endroit où le code ASP est inclus. Le morceau suivant illustre bien ceci :

```

<input title="Entrez votre prénom ici" name="txtPrenom" size="30"
  value="<%=txtPrenom%" tabindex="1">

```

Ici, nous voyons une zone de saisie de texte normale. Cependant, pour définir la valeur de cette zone de texte, nous utilisons le raccourci `Response.Write (<%=)` pour insérer la valeur de la variable `txtPrenom`. Vous vous souvenez sans doute que nous l'avons dimensionnée en dehors de nos fonctions ASP afin que sa portée soit la page. Nous utilisons maintenant sa valeur en l'insérant dans notre HTML.

La dernière astuce dans la section HTML est l'affichage de notre message d'erreur ou de succès. Ce message est stocké dans la variable `sMessage`. En

bas du formulaire, nous affichons le contenu de cette variable de la façon suivante :

```
<td nowrap width=500 colspan="3" align="center">
  <font face="Trebuchet MS">
  <br>
  <strong>
  <%=sMessage%>
  </strong>
  </font>
</td>
```

L'intérêt de ce code vient du fait que si `sMessage` est vide, rien n'est affiché.

## Résumé

Dans ce tutoriel d'apprentissage rapide d'ASP, nous avons d'abord vu que HTTP est le système de transactions qui envoie des pages web aux clients qui les demandent. C'est un aspect très important. Puis, nous nous sommes intéressés à Active Server Pages, ou ASP. Vous avez appris comment les pages ASP étaient créées, et quelles balises HTML spécifiques vous deviez inclure dans vos fichiers pour utiliser ASP. Nous avons examiné le modèle objet ASP et vu que les objets `Request` et `Response` étaient utilisés pour gérer les informations des requêtes HTTP et des réponses. Nous avons vu que l'objet `Application` était utilisé pour regrouper les pages en une seule application et que l'objet `Session` était, quant à lui, utilisé pour créer l'illusion que l'interaction entre l'utilisateur et le site représente une action continue. Enfin, nous avons créé une petite application faisant la démonstration de deux utilisations d'ASP : la validation de formulaire et la manipulation de données.

## RAPPEL

### ■ Les livres de référence sur ASP :

- ASP 3 Professionnel (Wrox Press - isbn 2212091516)
- Initiation à ASP3.0 (Wrox Press- isbn 2212092369)

### ■ Des articles de référence sur ASP :

- GASP : <http://www.gasp-fr.com> (en français)
- ASPToday : <http://www.asptoday.com> (en anglais)