

## Cours de C# pour l'ESSI - Cours 03 : ASP.NET et ADO.NET

### Index Rapide

PARTIE 1 : INTRODUCTION .....	1
PARTIE 2 : ASP.NET & WEBFORMS.....	2
<i>Qu'est ce qu'ASP.NET ?</i> .....	2
<i>Sans VS.NET</i> .....	3
Pourquoi ? Comment ? .....	3
Les outils.....	4
Les choses à savoir.....	4
Un exemple.....	5
Dans le code.....	5
Installer, compiler, configurer.....	7
<i>Avec VS.NET</i> .....	8
Introduction aux WebForms.....	8
Comment fait-on ? .....	9
1/ Création du projet.....	9
2/ Les outils .....	10
3/ Exemple d'une page.....	11
4/ Des contrôles magiques.....	11
5/ Du code .....	12
ASP.NET dans la page.....	12
Cohabitation ASP.NET et HTML.....	12
ASP.NET vers HTML et réciproquement .....	13
Débuguer.....	15
PARTIE 3 : ADO.NET .....	16
Pour changer, un exemple .....	17
Du code !.....	17
La connexion.....	18
Le DataAdapter.....	18
Le DataSet.....	19
La Liaison .....	20
PARTIE 4 : ASP.NET SUR LE WEB.....	21
<i>Les hosts ASP.NET sur le Web</i> .....	21
<i>Ma page chez Brinkster.com</i> .....	21
PARTIE 5 : LE MOT DE LA FIN.....	22
<i>Conclusion</i> .....	22
<i>Remerciements</i> .....	22
<i>Sources</i> .....	22

## Partie 1 : Introduction

Etant donné qu'il existe d'excellents sites et livres sur ASP.NET, ce cours n'entrera pas dans une présentation fastidieuse de chaque composant ASP.NET. Il ne sera pas non plus un cours recherché sur ASP(.NET). Le but est de vous montrer que l'on peut en faire sans Visual Studio .NET, que

ce n'est pas impossible. Je souhaite aussi montrer quelles sont les principales nouveautés dans ASP.NET. Et tout ceci en donnant un tour d'horizon sur ce que vous offre .NET maintenant.

Je présenterai aussi ADO.NET, la nouvelle mouture de Microsoft et en quoi elle surpasse ADO.

Je traiterai cette première partie SANS Visual Studio .NET pour montrer que vous pouvez faire de l'ASP.NET sans (si vous avez par exemple chez vous .NET Framework et IIS 5+ mais pas de VS.NET). Puis certaines spécificités de ASP.NET à l'aide de VS.NET, et pour finir un peu de base de données avec ADO.NET.

*Il est plus que conseillé de se référer à la présentation [C# vs. Java](#) ainsi qu'à [l'initiation](#) en cas de difficulté.*

Si vous constatez des erreurs, des imprécisions ou que vous avez des problèmes avec ce TP, adressez vous à [l'auteur](#).

## Partie 2 : ASP.NET & WebForms

### Qu'est ce qu'ASP.NET ?

L'idée est simple. Vous faites des applications Web comme vous faites des applications standard. Le client ne voit toujours pas votre code, et vous gagnez en facilité d'utilisation. Autre bénéfice, si vous utilisez VS.NET (comme expliqué plus bas) vous n'avez même plus à connaître le HTML.

Avec ASP.NET, votre page est un objet sur votre serveur. Quand vous l'envoyez au client, on lui demande sa version HTML (statique donc) et on la fournit au client. Quand un évènement à lieu (validation, touche pressée, ...) l'état de ses champs est renvoyé au serveur, qui met à jour son image "objet" de la page, réagis en accord avec VOTRE programmation, puis renvoi la version modifiée de la page au client.



*Comment fonctionne une demande de page ASP.NET*

Evidemment, la page n'est recompilée QUE quand il y a une modification de sa source. Ainsi, on n'interprète pas la page à chaque fois comme en ASP (ce qui était lent) mais on la compile UNE FOIS POUR TOUTE et on la lit dans le cache.

## Sans VS.NET

### Pourquoi ? Comment ?

Vous êtes amoureux de la liberté, du je travaille partout et même avec debug s'il le faut ?

Vous détestez les IDE car vous n'aimez pas qu'on fasse des choses dans votre dos ?

Vous êtes un tantinet parano ?

Vous agréez aux idées de .NET mais vous êtes un fervent défenseur des manchots ? ;)

Hormis ASP.NET qui va avec IIS, les WebServices, les specs du C#, du CLR/CLI (nom de code "Rotor") sont  GRATUITS :

C# : déposés auprès de l'[ECMA](#) (ersatz de l'[ANSI](#)). Vous pouvez les obtenir sur leur site où auprès de Microsoft [à cette adresse](#).

CLI/CLR : Ce sont les specs du langage et de la machine virtuelle. Déposés *aussi* auprès de l'[ECMA](#), vous pouvez *aussi* les obtenir sur leur site où auprès de Microsoft [à cette adresse](#).

Vous pouvez trouver [en suivant ce lien](#) un texte décrivant la position officielle

de Microsoft par rapport à Rotor.

Conséquence, de nombreux projets tels **mono::** qui veut faire "comme .NET mais mieux sous Windows et Linux".

### Les outils

Un bon éditeur de texte. Nous allons taper de l'ASPX et du C# (ça marche aussi avec VB.NET sans aucun problème, mais chacun son langage préféré). Personnellement je vais prendre mon Emacs Win32 avec son mode html-helper qui reconnaît l'ASPX. Et idem pour le code C# : sous Emacs ! Je ne triche pas.

Un IIS à la maison avec le .NET Framework installé.

Un invite de commande.

Un site sur HTML si vous n'êtes pas un HTML-guru (**mon préféré**)

Vous (si si !).

Nous allons faire une simple page. On pourra ensuite à loisir lui faire faire tout ce que l'on veut avec.

### Les choses à savoir

Une évolution très appréciable entre ASP et ASP.NET est l'idée du *codeBehind*. Tout ceux ayant fait de l'ASP savent à quel point il est impossible de déboguer un code ASP un peu compliqué car le HTML et le code VBScript sont mélangés dans un même fichier. Inextricable ! L'idée du *codeBehind* est d'une simplicité évangélique : séparer ce qui n'a pas à être mélangé, partie HTML et partie "code". Conséquence : on peut laisser un développeur faire son code C# ou VB.NET et faire faire une belle interface web par quelqu'un de compétent. Chose impossible avant car ces deux parties étaient indissociables. Ainsi au fichier *MaPage.aspx* correspond *MaPage.aspx.cs* qui définissent la partie statique et dynamique de la page (de manière imagée, évidemment).



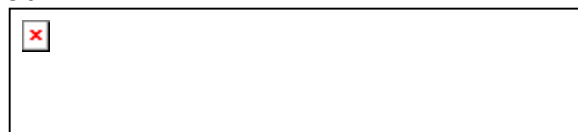
Maintenant, vous décrivez une sous classe de la classe mère `System.Web.UI.Page` et vous définissez dedans les méthodes que vous allez utiliser. Vous retrouverez dans chaque instance les objets ASP courants tels

Request et Response qui s'utilisent en ASP.NET comme en ASP. Bien évidemment, vous pouvez comme en ASP mélanger HTML et C# mais c'est refuser un bien grand avantage. (Sauf si vous n'avez qu'une ligne de code dans la page)

Votre code compilé est stocké *dans une assembly (sous la forme d'une DLL)* dans le sous répertoire *bin/* de votre site. C'est à partir de cette librairie que le runtime ASP.NET va instancier les différents objets de vos pages.

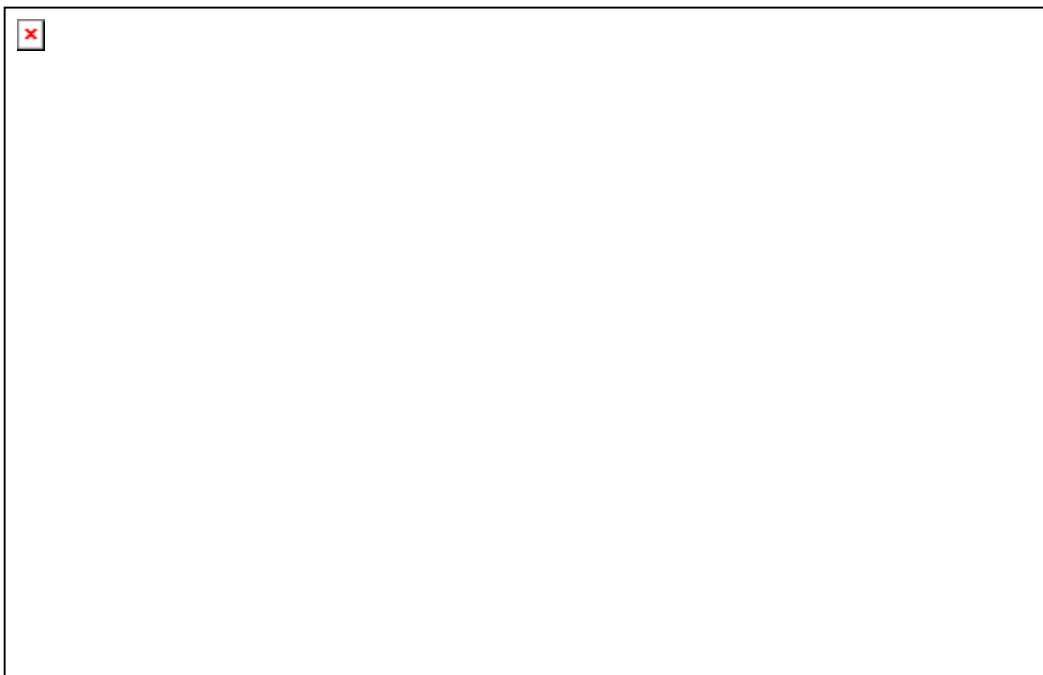
### Un exemple

Notre exemple s'appellera **AlanZero** (pour Zero VS.NET). La page principale sera *zero.aspx*. Voici ce à quoi devrait ressembler le sous-répertoire *AlanZero* de votre serveur web ...



Dans la racine de nom **AlanZero** (nom de l'application et du namespace) se trouve deux fichiers. Ils vont par paire : *Zero.aspx* et *Zero.aspx.cs*. A chaque fichier *xyz.aspx* correspond un fichier *xyz.aspx.cs* ou *xyz.aspx.vb* suivant votre langage.

C'est ça le *code behind* (comme expliqué plus haut).



### Dans le code

Vous remarquerez que c'est dans le sous répertoire *bin/* que doit se trouver une DLL nommée *AlanZero.dll*. C'est dans celle-ci que sont stockées toutes les classes de l'application.

Notamment, en regardant le code de notre page *aspx* on lit ceci en première

ligne :

```
<%@ Page Language="C#" Inherits="AlanZero.ZeroPage" %>
```

<% et %> : ce sont des chevrons ASP (ce qui est entre doit être traité par IIS avant envoi au client)

@ Page : chevron ASP.NET (signifie juste que c'est une PAGE en ASP.NET)

Language="C#" : le code de script dans la page (sauf indication contraire) sera du C# ; "ATTENTION, ce n'est pas un langage de script mais un vrai langage bien comme il faut qui sera compilé avant d'être exécuter. Donc pas du script mais un code binaire natif compilé. Donc plus performant qu'en ASP classique !" (dixit Laurent ;)

Inherits="AlanZero.ZeroPage" : cette page hérite (en tant qu'objet) de la classe ZeroPage du namespace AlanZero (*codeBehind*)

Vous trouverez la liste complète des attributs dans la documentation .NET Framework [@ Page](#).

Vous voudrez certainement inclure du code dans votre page (c'est l'intérêt !). Là, deux possibilités:

1. Vous voulez faire du code **inline** : pas que ce ne soit pas impossible, c'est renoncer à bien des avantages. Voilà comment faire :
  - o Soit vous tapez du code C# entre deux chevrons <% %>. Ainsi pour faire afficher l'heure et le jour courant vous taperiez

```
<% Response.Write(System.DateTime.Now); %>
```

2. Vous remarquerez la présence des objets usuels ASP. Ici `Response` et son appel à `write` permettent d'écrire dans la réponse (la page) la date et heure courante.
  - o Soit vous créez des "blocs" de code contenant des déclarations de méthodes. Vous pouvez ensuite les appeler à loisir dans votre page. Les blocs doivent être placés AVANT le chevron HTML <body> (idéalement entre les chevrons <head>...</head>). Ainsi si vous écrivez le bloc ci-dessous

```
o <script language="C#" runat="server">
o   void AfficherHeure() {
o       Response.Write(System.DateTime.Now);
o   }
o </script>
```

d'orénavant vous pourrez utiliser des appels dans votre page type

```
<% AfficherHeure(); %>
```

3. Vous voulez faire du **code-behind** : très simple. Vous tapez votre code dans *votrepag.aspx.cs* et vous l'appellez dans le corps de la page comme fait ci-dessus.

Vous devez décrire une classe qui héritera de `System.Web.UI.Page` et qui

appartiendra au namespace du nom de votre projet. Autre modification, votre page aspx n'hériterait plus de `System.Web.UI.Page` mais de votre classe.



Un classe d'une page ASP.NET

Les objets courants ASP sont encore présents : Request, Response, Session, ... Vous trouverez aussi Application et Cache, mais leur utilisation sort du cadre de ce cours.

### Installer, compiler, configurer

Vous devez avoir maintenant votre site. Il comprend des pages .aspx, du code dans des fichiers .aspx.cs, d'autres ressources (images, son, ...). Comment l'installer sur votre serveur IIS (ou un distant) ? Si vous l'installez chez un fournisseur de service, ce dernier vous donnera sûrement la marche à suivre, qui devrait ressembler à celle pour votre IIS.

1. Créez un répertoire *du nom de votre projet* dans le répertoire de base des pages pour IIS. Typiquement : `c:\InetPub\wwwroot\`. Dans notre exemple, nous créons `c:\InetPub\wwwroot\AlanZero`.
2. Copiez tout vos fichiers dans ce répertoire et créez y un répertoire `bin\`.
3. Il va falloir maintenant faire le boulot que faisait VS.NET dans notre dos : compiler. Sauf si vous n'avez que du code inline (vous êtes têtus !), vous devez donner un moyen à IIS d'utiliser vos pages. En effet, s'il comprend à merveille l'ASP.NET (si installé), il n'entend rien par contre au C# ou autre VB.NET.
4. Sachez que le filtre ISAPI (ce qui permet à IIS de comprendre ASP.NET) va aller chercher tout code exécutable dans le sous répertoire `bin\` et espère le trouver dans une DLL du nom *le-nom-de-mon-projet.dll*. Rien ne vous empêche d'avoir d'autres assemblies installées dans ce répertoire, et que vos pages les utilisent. Par contre leurs codes à elles DOIVENT être dans cette DLL. Comment faire ? Tapez ceci:

```
csc /out:bin/AlanZero.dll /t:library Zero.aspx.cs
```

- `csc` : c'est le compilateur C# (vous ne l'avez pas oublié tout de même !)
- `/out:bin/AlanZero.dll` : nous allons produire un fichier de sortie où il faut et bien nommé.
- `/t:library` : nous indiquons que nous souhaitons produire une DLL

(Dynamic Link *Library*)

- [Zero.aspx.cs](#) : la liste des fichiers .aspx.cs à compiler.
5. Faites *dans IIS* de votre répertoire contenant vos pages une "application". Clic droit sur le répertoire -> Paramètres d'application -> Créer.
  6. Un petit tour pour voir si tout va bien : <http://localhost/le-nom-de-mon-projet/le-nom-de-ma-page.aspx>

## Avec VS.NET

*NB : Tout ce que je vais raconter ici est également faisable et vrai sans utiliser Visual Studio .NET. Toutefois, l'intérêt présenté ici est que le développeur n'a plus à connaître le HTML ! Ce qui n'est pas possible en faisant tout le code "à la main".*

---

### Introduction aux WebForms

Quel est l'avantage d'utiliser VS.NET plutôt qu'un autre éditeur ? Pourquoi ne pas tout faire à la main ?

Je n'ai pas envie de discuter sur les bienfaits des IDE (Integrated Development Environnement). Chacun fait comme il veut :P ! Par contre c'est avec VS.NET que l'ASP.NET montre son réel pouvoir : permettre de faire des applications internet SANS CONNAITRE **HTML NI ASP** !!!!

On est en droit de se demander comment est-ce possible ? En effet, une application sur Internet passe par le HTML. Elle a beau le contourner, le "javascripter" ou "vbscripter", le DHTMLiser ou l'ASPiser, cela repose toujours sur des structures issues du HTML. Comment donc faire une application sans ? Où sont passées ces heures de tortures à essayer de faire fonctionner une page ASP ? A se torturer avec des chevrons barbares ? Et ne me dites pas qu'avec des outils clé en main, on fait plus simple, ce n'est pas forcément vrai. Et programmer quelque chose qui va plus loin que d'afficher l'heure ou de dire "hello world" devenait un casse tête.

Quand vous programmez une application avec GUI (IHM dans la langue de Voltaire) vous ne vous souciez pas de communiquer avec GDI et de sombres appels systèmes interminables. Vous "dessinez" (je vous le souhaite car à la main c'est ignoble) vos composants sur votre frame : boutons, labels, zone de texte, ...

Hop un double clic et on se branche sur tel ou tel évènement de tel contrôle. On manipule des propriétés et des méthodes simples et sympathiques : Name, Font, Text, OnClick(..), OnTextChanged(..), ...



Le plus important est que vous vous moquez (dans 95% des cas) de comment est géré l'affichage et tout ce qui touche à l'interface graphique ! Ce n'est pas votre problème. Et bien elle est là l'idée des **WebForms** ...

Et cette idée, c'est de mettre le développeur devant une page blanche, avec ses boutons à dessiner, qui ont des propriétés simples, des noms avenants et les méthodes *ad'hoc*. Gérer les événements ? Un aller retour client serveur hélas, mais c'est le prix des applications *server-side*.

Votre page est sur le serveur et on en envoie une version (statique) HTML au client à chaque demande. C'est une "*sérialisation HTML*" comme j'aime à dire. J'en profite pour dire que celle-ci est optimisée pour le client. C'est à dire que si ce dernier supporte DHTML, la page produite en contiendra pour "délester" le serveur de quelques tâches. (le tout *100% W3C Compliant*)

---

## Comment fait-on ?


Fini la théorie ici. Prenons un cas concret (je sais, j'aime bien). Nous allons faire une page d'une application qui prendrait des RDV pour une personne à une certaine date (pour changer). Je ne ferai pas le code qui gère des fichiers, rdv, etc ... ce n'est pas le but.

### 1/ Création du projet

Simple. Ce coup-ci, demandez à Visual Studio .NET de faire un nouveau "ASP.NET Web Application" que vous nommerez selon votre goût. Après quelques secondes de boulot, vous devriez vous trouver devant quelque chose ressemblant à ceci :

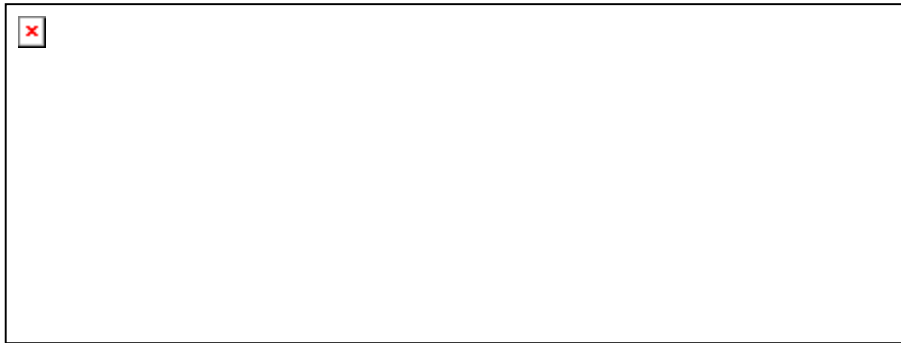


*Une nouvelle application web ASP.NET*

 Je rappelle au passage qu'il faut que vous ayez IIS 5+ installé sur votre machine, et que les extensions ASP.NET soient bien enregistrées. Si ce n'est pas le cas pour X raison : soit vous désinstallez IIS + .NET et vous réinstallez, soit vous lancez C:\WINDOWS\Microsoft.NET\Framework\la-version-du-framework\aspnet\_regiis.exe

## 2/ Les outils

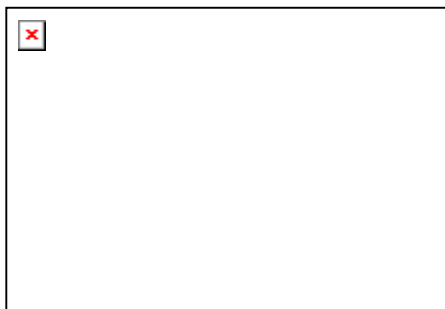
Vous êtes maintenant face à une page qui ressemble à s'y méprendre (pour ceux qui connaissent) à celles de tout bon IDE. Vous allez faire une application web (le nom n'est pas usurpé). Voici la palette d'outils qui est à votre disposition. Vous remarquerez qu'elle est très proche de celle de l'environnement Windows.



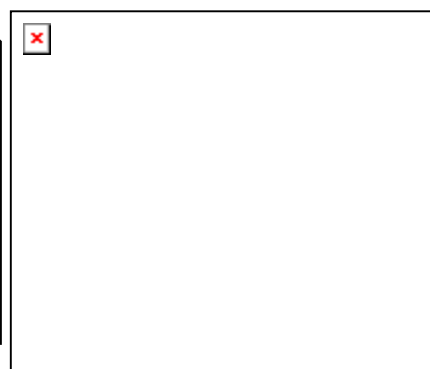
*La palette des contrôles serveurs ASP.NET*

Ces contrôles sont nommés *contrôles serveurs* car, comme expliqué plus haut, ces derniers fonctionnent du côté serveur. N'est envoyé chez le client que la représentation HTML des objets affichés (avec un peu de JavaScript/VBScript si besoin).

Les contrôles "Zone de Texte" et autres "Boutons" que vous voyez ne sont pas les contrôles HTML courant. Ce sont des contrôles serveurs ASP.NET ! La preuve, vous les manipulez comme des contrôles windows. Regardez à droite la fenêtre des propriétés/événements. Même la page en elle même est un objet ASP.NET. Vous manipulez ses propriétés comme vous le feriez avec ses contrôles.




*Les propriétés de la page principale*



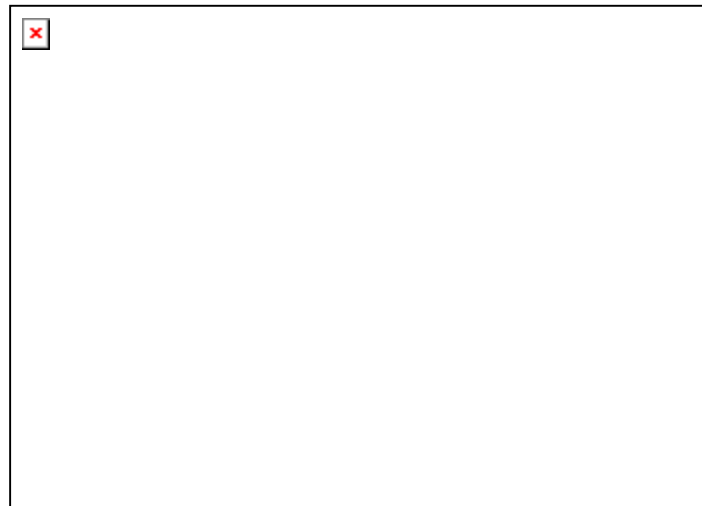
*Les propriétés d'un bouton ASP.NET*

Vous pouvez bien entendu inclure dans votre page des contrôles HTML standard, mais à ce moment là, vous ne pourrez utiliser les avantages de ASP.NET. Réservez les à des fonctions décoratives, ou à des tâches où ils sont suffisants. Vous reconnaîtrez les contrôles serveurs des contrôles HTML standard, car

les contrôles serveurs ont tous un petit  en haut à gauche.

### 3/ Exemple d'une page

Voici comme dit plus haut une page ASPX faite avec VS.NET. J'y ai ajouté une zone de texte pour prendre un nom de personne, ainsi qu'un calendrier pour une date. Oui, un calendrier. Pas fait main, mais fait ASPX ! Entièrement customisable sur l'apparence, il remplit son rôle à merveille. Outre cela, un bouton de remise à zéro, un de validation, une bannière de pub, une zone de feed-back utilisateur et des jolies couleurs.

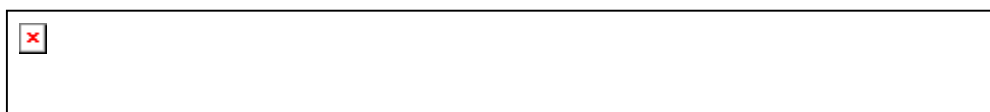


*Une page de prise de rendez-vous*

*Vous pourrez trouver le source [ici](#) mais pas la voir tourner pour l'instant ... pas encore uploadé :(*

### 4/ Des contrôles magiques

Magique est peut-être un peu fort, mais qui facilitent la vie, c'est clair. Par exemple les *Validators*. Ils fonctionnent du côté client, et permettent d'interdire l'envoi de la page tant qu'un certain critère n'est pas rempli. Bénéfice immédiat : vous réduisez la charge réseau de votre serveur ! La basse besogne de vérification est effectuée côté client par du javascript (généralisé automatiquement). Dans mon exemple, j'ai ajouté un **RequiredFieldValidator** qui fait apparaître un certain message si l'utilisateur essaye d'envoyer le formulaire sans que ce champ contienne une valeur.



*En mode design, on voit le RequiredFieldValidator*

Pour lui indiquer quel contrôle serveur surveiller, il suffit de changer la propriété idoine.



*Choix du contrôle à surveiller*

Autre exemple : le calendrier *Calendar*. Impossible de faire plus simple ou plus utilisable. C'est simplement LA solution dès qu'on veut que l'utilisateur saisisse une date. ET évidemment on le customise à notre guise ...

### 5/ Du code

Comment marche mon bouton "RAZ" ? En effet, ce n'est pas comme je l'ai dit un bouton HTML "classique" qui efface le contenu des contrôles d'un formulaire. Et bien comme promis, c'est du C# ... On sélectionne l'événement "Click" du bouton, et on saisie du code :




*Le code du bouton RAZ*

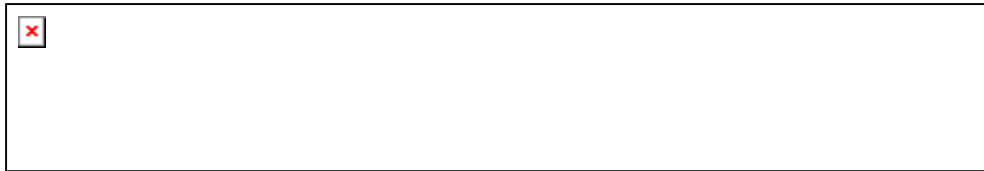
---

## ASP.NET dans la page

### Cohabitation ASP.NET et HTML

NB : Avec Visual Studio .NET vous pouvez à tout instant voir le code HTML de votre page en cliquant sur le bouton idoine . Il est très déconseillé d'essayer de trifouiller soi-même le code HTML dans VS.NET, car vos modifs pourraient passer à la trappe bien vite.

Voici à quoi ressemble une partie du code HTML de la Web Form. Vous remarquerez la présence de chevrons nouveaux, de la forme `<asp:le-type-du-contrôle id="le-nom-du-contrôle" style="les-infos-du-contrôle..." runat="server" >...</asp:le-type-du-contrôle>`



*Quelques lignes d'ASP.NET d'un fichier .aspx*

Vos contrôles ASP.NET sont dans un formulaire (chevrons <form ...>). Normal, ils doivent causer des *post-backs* (un "retour-d'information" au serveur), et c'est la seule solution. Eh oui, car c'est côté serveur où se trouve le code disant quoi faire. (comme en HTML, ne mettez pas plus d'un formulaire par page, l'ASP.NET est "tatillon" là-dessus)

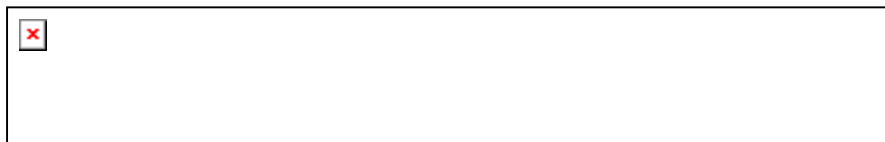
Ayez toujours en tête qu'un aller retour client-serveur est coûteux, et que moins vous en faites, mieux tout le monde se porte (surtout le serveur !). Ainsi, évitez de définir un callback pour l'évènement "TextChanged" d'une zone de texte ... Votre page ferait un aller retour à chaque nouvelle lettre ! Imaginez le trafic !

#### **ASP.NET vers HTML et réciproquement**

Lors de l'envoi au client, votre page est transformée en son équivalent HTML (le seul langage à peu près universel du Web). Ainsi, votre client ne recevra JAMAIS de chevrons genre "<asp:Button ..." ! Voici une partie de notre page de réservation, côté client ...



*Côté SERVEUR*

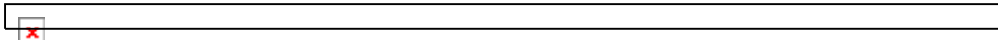


*Les mêmes lignes côté CLIENT*

ASP.NET a "sérialisé la page en HTML" et l'a envoyée au client. Vous retrouvez les mêmes objets, sous leur forme HTML. On peut se poser des questions : mais cette transformation ne peut être complète ? Où passe le reste des données "non HTML" ?

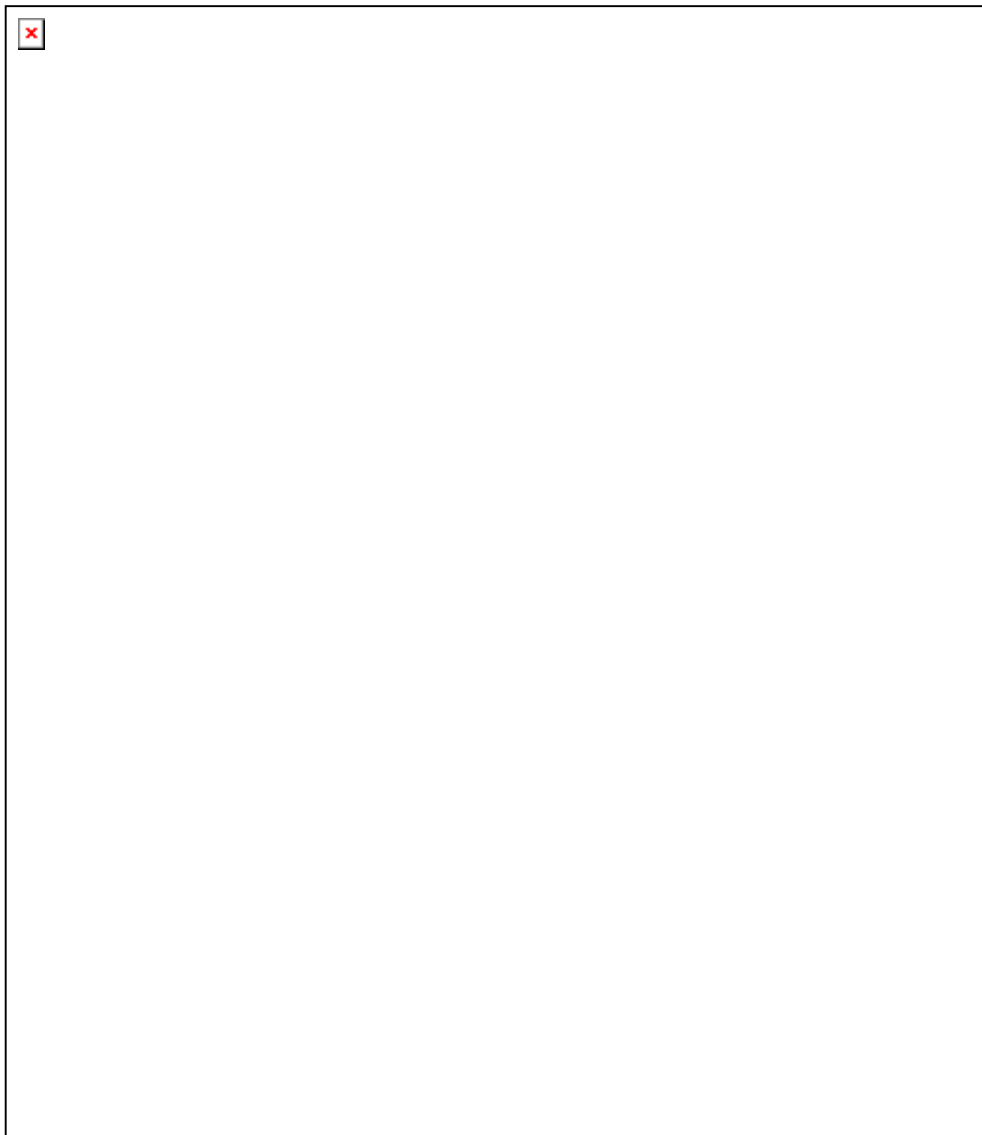
Ce qui n'est pas HTML va dans un champ invisible, dont vous ne connaissez pas l'existence et que vous n'aurez pas à utiliser (directement en tout cas).

C'est ASP.NET qui le génère pour sauvegarder (de manière cryptée) les données ne pouvant être codées de manière HTML. Ce contrôle est **\_\_VIEWSTATE**.



*Le contrôle \_\_VIEWSTATE visible seulement côté client*

C'est à partir de la page HTML et de \_\_VIEWSTATE que le serveur "déséréalise" ces données en un objet .NET (votre page).



*Un aller-retour d'une page ASPX*

**NB :** vous pouvez désactiver le fait qu'un contrôle sauvegarde son état dans le champ \_\_VIEWSTATE en mettant sa propriété **EnableViewState** à *false*. C'est très utile pour les contrôles gourmands genre DataGrid, etc... Inutiles de sauvegarder leur état chez le client et d'alourdir ainsi ET la page ET le volume

à envoyer/retourner.

## Débuguer

Casse tête, grand plaisir de notre métier : la chasse aux bugs. Avec l'ASP, je dois avouer que c'est MON PIRE SOUVENIR. Je n'ai jamais autant perdu de temps dans ma vie qu'en débuggant des pages ASP. Mais ASP.NET m'a réconcilié (pas avec les bugs, mais avec leur chasse). Déjà, on tape du code d'un côté et on fait une interface de l'autre. On peut compiler le code de la page sans même avoir IIS : on élimine ainsi toutes les fautes de codage (mais pas celles d'exécution). Ensuite, j'ai rarement vu des pages d'erreurs aussi bien faites et belles que celles générées par ASP.NET quand on essaye de voir une page que l'exécution à fait aller dans les choux (ça marche aussi avec "fraise"). Et ça continue ...

ASP.NET vous permet d'obtenir pour chacune de vos pages un résumé très détaillé des sessions, cookies, en-têtes, variables serveurs et ceci très simplement.



*Une page d'information générée par ASP.NET*

Pour cela, il vous suffit d'affecter la propriété **trace** à *true* dans l'en-tête de votre page ASPX.

```
<%@ Page Language="C#" Inherits="System.Web.UI.Page" trace="true" %>
```

Vous pouvez mixer vos propres messages avec ceux générés automatiquement par ASP.NET. Ainsi, remplacez vos traces genre "Response.Write ("Variable A =" + A.ToString())" par des "**Trace**.Write("...")" (ou encore des *Trace.Warn()* qui sont pareil MAIS en rouge donc plus visibles).



*Vos traces dans le rapport d'ASP.NET*

Et même, vous pouvez inclure une trace au niveau de votre application (et non simplement d'une page). Editez pour cela votre fichier *web.config* de votre application, et définissez dans le noeud `configuration -> system.web` le noeud suivant :

```
<trace
enabled="true"
localOnly="true"
/>
```

## Partie 3 : ADO.NET

*Il sera ici question de base de données. Si vous n'êtes pas intéressé, sautez à la section suivante.*

*Je ne traiterai pas ici de l'utilisation des bases de données "comme en ASP mais avec ASP.NET" vu que c'est pareil !!!! Je ne parlerai ici que de la nouveauté : **ADO.NET**.*

*Je supposerai ici que mon lecteur a les notions de base concernant les bases de données, leur fonctionnement et qu'il ne panique pas devant une ligne de SQL ...*

**Notez que tout ce qui est dit ici est aussi entièrement valable pour une application autre qu'une WebForm : ADO.NET n'est pas exclusif à ASP.NET, mais fait partie du système .NET.**

Les bases de données sont des outils indispensables de l'informaticien. Apprendre à les maîtriser est indispensable, quel que soit votre "arme" : génie logiciel, webmestre, réseau, ... Le monde étant hétérogène (fort heureusement), les bases de données n'échappent pas à la règle. Résultat pour tenter d'harmoniser tout ça, Microsoft à développé différentes technologies, descendantes les unes des autres. La dernière en date était ADO (**ActiveX Data Objects**). Son évolution est **ADO.NET**. Elle fonctionne comme ADO mais rajoute encore une couche d'abstraction. Ceci fait, certain codes qui pouvait être différent suivant les bases de données manipulées se trouvent être identiques (facilité d'implémentation, maintenance, ...). ADO.NET est aussi plus adapté aux applications web gourmandes en trafic et ainsi bien souvent en connexions BD. Il comble même certaines lacunes d'ADO (nous n'en parlerons pas ici).



Mais la différence principale tient en le fait qu'ASP.NET fonctionne en mode "asynchrone" ou "déconnecté". Vous possédez sur le serveur WEB une partie de la base de données que vous souhaitez modifier, vous effectuez votre tâche (tri, ajout, suppression, ...), puis soumettez le résultat au serveur BD. Le transfert de donnée s'effectuant au format XML (au coeur de .NET) SAUF en local où on utilise la méthode normale (et plus efficace).

Deux namespaces vont être utiles : `System.Data.OleDb` et `System.Data.SqlClient`. ADO.NET propose avec *OleDb* de pouvoir utiliser tout type de base de données mais en passant par plusieurs wrappers intermédiaires. Par contre *SqlClient* permet de "court-circuiter" et de parler de manière plus directe au driver, mais ceci seulement pour SQLServer. Une nouvelle fois, revoici le choix entre universalité et performance...

### Pour changer, un exemple

Avec Access, j'ai créé une petite base de données n'ayant qu'une table. La base est "alanvsnetdb.mdb" et la table se nomme *Numeros*. Elle associe à un chiffre sa représentation textuelle française et anglaise (que jusqu'à 7, je suis flemmard). Voici son schéma relationnel :

Numeros (Numero, Fr, En)

La base est stockée dans le sous répertoire *db/* de mon application. Une nouvelle page est présente : *accesbd.aspx* qui (ô surprise) accède à notre BD.

Dans la page *accesbd.aspx* il n'y a rien d'autre qu'une zone de texte et un *DataGrid*. Je n'ai pas indiqué le nombre de colonnes, lignes, types .... etc au *DataGrid*. Je n'ai rien fait d'autre que le créer et choisir ses couleurs d'affichage. (j'insiste car c'est important plus tard)



*Le design de la page dans VS.NET*



*La page vue sous IE 6*

### Du code !

Et bien pas tant de code que cela. Notez toutefois qu'utilisant une base Access, j'utilise des *OleDb...* et non des *SqlClient...* Voici le code complet générant l'affichage.

```

OleDbConnection maConnexion = null;
try
{
    maConnexion = new OleDbConnection (
    @"Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=""C:\inetpub\wwwroot\AlanVSNET\bd\alanvsnet.mdb""");
    maConnexion.Open();

    OleDbDataAdapter monAdapter = new OleDbDataAdapter ("SELECT * FROM
    Numeros", maConnexion);
    DataSet monDataSet = new DataSet ("monDataSet");

    monAdapter.Fill(monDataSet);

    dg.DataSource = monDataSet;
    dg.DataBind();

}
finally
{
    if (maConnexion != null)
        maConnexion.Close();
}

```

[csharpindex.com/colorCode](http://csharpindex.com/colorCode)

Découpons le en partie logique.

### La connexion

Pour accéder à la base de données, on a besoin d'une connexion. C'est le rôle des lignes suivantes (peu importantes ici, car idem qu'en ASP).

```

maConnexion = new OleDbConnection ("...");
maConnexion.Open();

maConnexion = new SqlConnection ("...");
maConnexion.Open();

```

La section [OleDbConnection](#) et [SqlConnection](#) dans .NET Documentation.

### Le DataAdapter

```

OleDbDataAdapter monAdapterOleDb = new OleDbDataAdapter ("SELECT * FROM
Numeros", maConnexion);

SqlDataAdapter monAdapterSqlSrv = new SqlDataAdapter ("SELECT * FROM
Numeros", maConnexion);

```

C'est un **ensemble de requêtes SQL** qui vous servira à **peupler un(des) DataSet(s)**.

C'est lui qui prends en charge la copie, gestion, exécution des requêtes, etc. Je ne rentrerai pas dans les détails, mais il regroupe les tables que vous lui fournissez à la création, et peut stocker vos différentes opérations dans un

DataSet.

La section [OleDbDataAdapter](#) et [SqlDataAdapter](#) dans .NET Documentation.

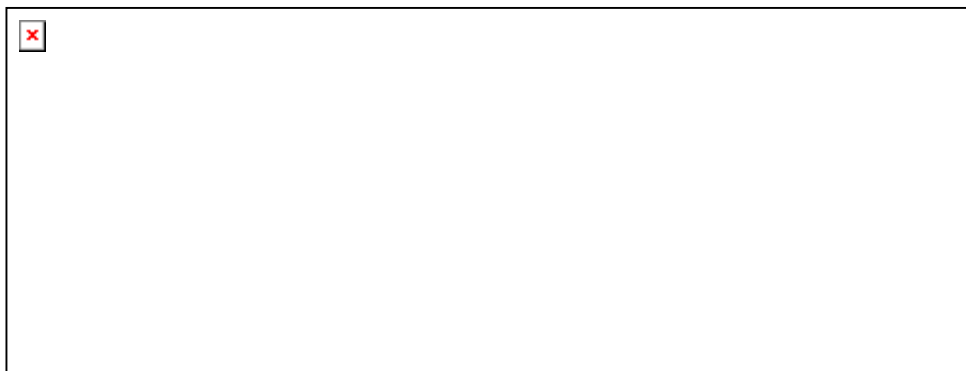
### Le DataSet

```
DataSet monDataSet = new DataSet ("monDataSet");
```

C'est le "coeur" d'ADO.NET. Ce n'est plus à la base de données que vous accédez mais à cet objet qui est une copie locale d'une partie de la base réelle. C'est en quelque sorte une base locale à votre application et qui vous permet de travailler **en mode déconnecté**.

Ainsi, une fois votre DataSet peuplé avec votre requête (ou ensemble de requêtes) vous pouvez vous déconnecter de la base de donnée ! Vous utilisez le DataSet comme vous l'auriez fait avec la base, mais sans trafic ! C'est de l'accès direct car la partie de la base que vous utilisez est en mémoire centrale.

Et pour répercuter les modifications sur la base, appelez la méthode *Update(DataSet)* de votre connexion (une fois reconnecté, évidemment !).



*Un DataSet est une image locale d'une partie de la base distante*

Vous pouvez évidemment utiliser un curseur sur un DataSet (comme en mode connecté), mais aussi grâce aux méthodes *DataBind()* et *DataSource()* des différents contrôles .NET, avoir le travail mâché pour vous ! (comme dans l'exemple) Une partie des composants supportant ce traitement sont :

- DataGrid
- DataList
- ListBox
- DropDownList
- CheckBoxList
- RadioButtonList
- ...

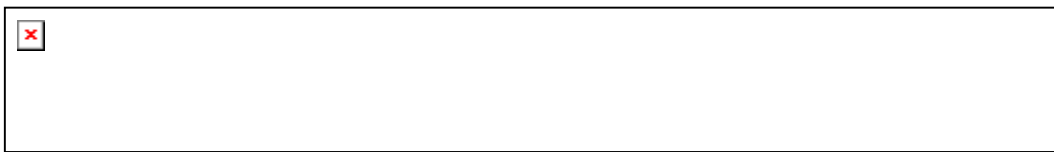
Cela ne s'arrête pas là. Votre DataSet est nativement exportable en XML. Il peut vous donner à la demande les schémas relationnels de son contenu (au format XSD) et les données contenues (au format XML). Ainsi, c'est 100% de

compatibilité avec tout autre système supportant XML et base de données. C'est pour cela que les DataSet sont universels quel que soit vote BD (il n'existe pas de ~~OleDbDataSet, SqlDataSet, ...~~)

Exporter une structure d'un DataSet et/ou ses données est EXTREMEMENT simple. Voici les deux lignes s'acquittant de cette tâche...



*Pour écrire une structure de DataSet*



*Pour écrire les données du DataSet*

Le résultat obtenu est deux fichiers XML complètement universels ! Partageable avec toute application de tout OS. (hé non, pas de tentative hégémoniste de Microsoft ici, n'en déplaise à certains)



*La structure en XML*



*Les données sous forme XML*

La section DataSet dans [.NET Documentation](#).

**La Liaison**

Pour afficher les données après la requête, prenez un des contrôles supportant DataSource/DataBind. Affectez lui comme source votre DataSet, et faites votre DataBind.

Notez qu'en ASP.NET, si vous avez plusieurs contrôles qui utilisent des DataSet, faites un DataBind de la page (`Page.DataBind();`) plutôt qu'un DataBind par contrôle : le contrôle Page répercutera le message à ses contrôles enfant.

## Partie 4 : ASP.NET sur le Web

### Les hosts ASP.NET sur le Web

Pour faire profiter le monde de vos pages ASP.NET, soit vous ouvrez votre propre site Web, soit vous vous faites hoster. Evidement, je vais vous parler de la seconde option ici (la première se limitant à installer IIS, ASP.NET et trouver un fournisseur de nom de domaine). Beaucoup de ces sites proposent evidement les deux formules (gratuit/payant), la première étant généralement une restriction de la seconde. Toutefois regardez bien ce qui vous est offert : accès base de données, espace disque offert, composants ASP/ASP.NET installés sur le serveur, extensions Front Page (pour faciliter les màj), accès FTP ou Web (sachant que les interfaces Web sont souvent TRÈS agaçantes), ...

Brinkster    
msugs   sponsorisé par  et 

[www.prosygma.com](http://www.prosygma.com) \$\$\$  
[www.ikoula.fr](http://www.ikoula.fr) \$\$\$  
[www.phidji.com](http://www.phidji.com) \$\$\$ ou gratuit 15 jours :(  
[www.netenligne.com](http://www.netenligne.com) \$\$\$  
[www.dotnetisp.com](http://www.dotnetisp.com) \$\$\$  
[www.linkbynet.com](http://www.linkbynet.com) \$\$\$

### Ma page chez Brinkster.com

Un host gratuit de ASP.NET est [Brinkster.com](http://Brinkster.com). Il propose 30 Mo d'espace et accès à Access (dire 10x rapidement). Pour l'instant on ne peut faire de *code behind* et on doit taper son code inline. C'est pas très cool, mais ça devrait bientôt changer ...



# Partie 5 : Le mot de la fin

## Conclusion

Personnellement, ASP.NET m'a réconcilié avec les applications web. Ayant contribué à en développer une dans un récent emploi, j'en avais gardé une idée de complexité, de code pas propre et de difficulté à gérer et maintenir. De plus je devais m'arracher les cheveux avec des considérations HTMLesques, qui ne faisaient que me compliquer ma tâche de développeur. Mais c'est fini : on programme dans des langages propres, Objet, de notre choix (C#, VB.NET,...), déboguable aisément. Le code que l'on produit est compilé donc plus rapide et .NET dépasse 1000x les possibilités de VB. ADO.NET quand à lui apportera son lot de satisfactions aux développeurs travaillant avec plusieurs bases (d'une manière ou d'une autre). Et la possibilité de travailler en mode déconnecté soulagera tout les serveurs (et les admin, chasseurs de bande passante).

## Remerciements

Merci à Laurent ELLERBACH pour sa sympathie, ses idées, et les opportunités qu'il m'a offert et continue à m'offrir. Et aussi pour sa relecture !!!  
Merci aussi à Céline pour sa très gentille relecture, d'autant plus pénible qu'elle ne comprends pas ce que je dis ... ;)

## Sources



**Formation à ASP.NET** (poche, pas cher, très bien) ISBN : 2-84082-865-0 chez MSPress (et sur le web [www.c2i.fr/press](http://www.c2i.fr/press))

**Designing Microsoft ASP.NET Applications** (cher) ISBN : 0-7356-1348-6 chez MSPress

**Applied Microsoft.NET Framework Programming** (cher, très complet, mais ne parle pas d'ADO.NET ni d'ASP.NET) ISBN : 0-7356-1422-9 chez MSPress