

1. Introduction

Afin de comprendre mieux l'intérêt que présente ASP.NET dans le développement de site Web dynamique, nous allons aborder un exemple de programmation événementiel en utilisant des scripts côté client écrits en Javascript (cela pourrait être aussi en Vbscript) et côté serveur en ASP et ASP.NET .

Imaginons une page Web comprenant deux listes déroulantes: la première permettra d'effectuer le choix d'un modèle de voiture tandis que la deuxième comprendra les couleurs disponibles pour le type de véhicule sélectionné. Le changement de choix dans la première liste doit provoquer la mise à jour du contenu de la deuxième liste.

Intéressons nous au contenu du fichier html pour la création de ces listes:

```
<html>
  <head>
    <title>Exemple1</title>
  </head>
  <body>
    <form>
      <SELECT id="Voiture" name="Voiture">
        <option selected value="NO">Effectuer un choix</option>
        <option value="V1">Voiture V1</option>
        <option value="V2">Voiture V2</option>
        <option value="V3">Voiture V3</option>
      </SELECT>&nbsp;
      <SELECT id="Couleur" name="Couleur">
        <option value="NO" selected>Pas de sélection</option>
      </SELECT>
    </form>
  </body>
</html>
```

L'objectif n'étant pas la mise en forme de la page, nous nous sommes limités au code le plus simple. Il est clair que tout changement effectué dans le choix de la première liste ne peut provoquer aucun changement dans la deuxième liste. Nous allons insérer du code Javascript pour que notre page soit réactive au changement. Voici le code obtenu:

```
1    <html>
2    <head>
3        <title>Exemple 1</title>
4        <script language="javascript">
5            <!--
6                function Change_color()
7                {
8                    var max=document.Devis.Couleur.length;
9                    for (i=1;i<max;i++)
10                       document.Devis.Couleur.options[1]=null;
11
12                    if (document.Devis.Voiture.selectedIndex==1)
13                    {
14                        document.Devis.Couleur.options[0].innerText="Selectionner
15                        une couleur";
16                        nopt = 1;
17                        opt = new Option("Gris","",true,false);
18                        document.Devis.Couleur.options[nopt++] = opt;
19                        opt = new Option("Beige","",true,false);
20                        document.Devis.Couleur.options[nopt++] = opt;
21                        opt = new Option("Jaune","",true,false);
22                        document.Devis.Couleur.options[nopt++] = opt;
```

```

22     }
23     if (document.Devis.Voiture.selectedIndex==2)
24     {
25     document.Devis.Couleur.options[0].innerText="Selectionner
26     une couleur";
27         nopt=1;
28         opt = new Option("Bleu", "", true, false);
29         document.Devis.Couleur.options[nopt++] = opt;
30         opt = new Option("Rouge", "", true, false);
31         document.Devis.Couleur.options[nopt++] = opt;
32     }
33     if (document.Devis.Voiture.selectedIndex==3)
34     {
35         document.Devis.Couleur.options[0].innerText="Pas de
36         couleur";
37     }
38
39     }
40     //-->
41     </script>
42 </head>
43 <body>
44     <form name="Devis">
45         <SELECT id="Voiture" name="Voiture" onchange="Change_color()">
46             <option selected value="NO">Effectuer un choix</option>
47             <option value="V1">Voiture V1</option>
48             <option value="V2">Voiture V2</option>
49             <option value="V3">Voiture V3</option>
50         </SELECT>&nbsp; &nbsp;
51         <SELECT id="Couleur" name="Couleur">
52             <option value="NO" selected>Pas de sélection</option>
53         </SELECT>
54     </form>
55 </body>
56 </html>

```

Commentaires:

A la ligne 45, nous retrouvons la gestion de l'événement lié à un changement de choix dans la première liste déroulante: il s'agit de l'événement *onchange* . Nous pouvons lier cet événement à l'exécution d'une fonction: *Change_color()*. Cette fonction se trouve définie dans un conteneur identifié par les balises `<script>` et `</script>`. Dans la balise d'ouverture, nous retrouvons l'information sur le langage dont le navigateur devra tenir compte pour pouvoir interpréter le code. Une des particularités des scripts est l'accès à une arborescence d'objets dépendant du navigateur pour lequel ce script est écrit et repris sous la dénomination DOM (Document Object Model), en français Modèle objet du document. Cette dépendance du navigateur et encore plus la version de celui-ci vous oblige à être très prudent dans l'écriture du code faute de quoi votre site ne serait pas fonctionnel pour l'ensemble des internautes voulant s'y connecter. C'est un des aspects débattus dans le cours de javascript.

Un des avantages, c'est que cette façon de travailler ne génère pas de trafic sur votre réseau et donc quelle est plus rapide au niveau temps de réaction par rapport à un événement donné. Nous pouvons remarquer que le code javascript est bien dissocié des balises HTML et de ce fait, peut être facilement réutilisable.

Nous allons maintenant aborder les scripts côté serveur en commençant par l'ASP. L'ASP dispose également d'objet et le langage est laissé au choix du programmeur: soit le VBScript, soit le JScript. Pour la plus grande facilité de compréhension, nous avons choisi le JScript car il se rapproche le plus du Javascript et du C#.

```

1  <%@ Language="JScript" %>
2  <html>
3    <head>
4      <title>Exemple 1</title>
5    </head>
6    <body>
7      <form name="Devis" method=post action="http://127.0.0.1/exemple1.asp">
8        <SELECT id="Voiture" name="Voiture"
9          onchange="document.forms[0].submit()">
10         <option selected value="NO">Effectuer un choix</option>
11         <option value="V1">Voiture V1</option>
12         <option value="V2">Voiture V2</option>
13         <option value="V3">Voiture V3</option>
14       </SELECT>&nbsp; &nbsp;
15       <SELECT id="Couleur" name="Couleur">
16         <option value="NO" selected>pas de selection</option>
17         <% if (Request.Form("Voiture")==="V1")
18           Response.Write(" <option>Bleu</option>");%>
19         <% if (Request.Form("Voiture")==="V1")
20           Response.Write(" <option>Rouge</option>");%>
21         <% if (Request.Form("Voiture")==="V1")
22           Response.Write(" <option>Jaune</option>");%>
23         <% if (Request.Form("Voiture")==="V2")
24           Response.Write(" <option>Gris</option>");%>
25         <% if (Request.Form("Voiture")==="V2")
26           Response.Write(" <option>Beige</option>");%>
27       </SELECT>
28     </form>
29 </body>
30 </html>

```

Commentaires:

Un fichier utilisant la technologie ASP développée pour l'IIS de Microsoft doit posséder l'extension .asp. Il est impératif au début du fichier de renseigner le langage avec lequel le serveur devra interpréter les pages ASP. Nous retrouvons cette information à la ligne 1. Nous retrouvons toutes les parties de code JScript dans des conteneurs délimités par <% et %>.

Un avantage des scripts côté serveur est qu'ils sont exécutés sur le serveur et de ce fait il ne faut pas se soucier de la compatibilité avec un quelconque navigateur; seul le serveur Web est important pour l'interprétation des codes ASP.

Un désavantage de l'ASP est le fait que le code et les balises HTML qui ne doivent subir aucun traitement préalable par le serveur et être donc envoyées sans modification vers le client, sont mélangés. En ce qui concerne la possibilité de réutiliser du code précédemment développé pour certains sites, cela devient un peu compliqué et c'est sans doute le plus grand reproche que l'on puisse faire à cette technologie. Un autre aspect pourrait être l'occupation de bande passante liée à ce trafic entre les clients et le serveur pour l'ensemble des événements à gérer.

Dans une dernière étape, nous allons considérer le même objectif pour la page Web avec un script à exécuter côté serveur mais nous développerons la solution en utilisant l'ASP.NET. Cette technologie nous permet de choisir également un langage tel que le VB.NET ou le C#. Etant donné l'apprentissage du langage C# dans le cours de programmation pour les applications Windows, nous retiendrons ce langage.

```

1  <%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
2  AutoEventWireup="false"
   Inherits="Exemple3.WebForm1" %>
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

```

```

3      <HTML>
4          <HEAD>
5              <title>WebForm1</title>
6              <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
7              <meta name="CODE_LANGUAGE" Content="C#">
8              <meta name="vs_defaultClientScript" content="JavaScript">
9              <meta name="vs_targetSchema"
              content="http://schemas.microsoft.com/intellisense/ie5">
10         </HEAD>
11         <body MS_POSITIONING="GridLayout">
12             <form id="Form1" method="post" runat="server">
13                 <asp:DropDownList id="Voitures" runat="server"
14                     AutoPostBack="True">
15                     <asp:ListItem Value="NO">Effectuer un
16                         choix</asp:ListItem>
17                     <asp:ListItem Value="V1">Voiture V1</asp:ListItem>
18                     <asp:ListItem Value="V2">Voiture V2</asp:ListItem>
19                     <asp:ListItem Value="V4">Voiture V4</asp:ListItem>
20                 </asp:DropDownList>
21                 <asp:DropDownList id="Couleurs" runat="server">
22                     <asp:ListItem Value="NO">Pas de
23                         selection</asp:ListItem>
24                 </asp:DropDownList>
25             </form>
26         </body>
27     </HTML>

```

Commentaires:

La première ligne reprend une syntaxe fort proche de celle rencontrée dans l'exemple précédent puisqu'elle reprend un conteneur délimité par les caractères <% et %>. Les attributs sont malgré tout différents et nous nous limiterons à certains pour ce paragraphe: *language="c#"* indique le langage utilisé pour notre script côté serveur, dans ce cas ci le c#. *Codebehind="WebForm1.aspx.cs"* Cet attribut permet de séparer le code du script de la partie HTML, ce qui était possible dans les scripts côté client. Nous retrouverons le code dans un fichier séparé appelé WebForm1.aspx.cs. Nous en donnerons le contenu juste après. *Inherits="Exemple3.WebForm1"* Cet attribut indique le nom de la classe à utiliser lors de la mise en place du code behind.

Nous retrouvons l'intérêt de l'ASP dans le sens où les scripts sont exécutés sur le serveur et de ce fait, il ne faut pas s'inquiéter d'une quelconque incompatibilité des scripts avec des navigateurs différents. D'autre part, nous retrouvons l'intérêt des scripts côté client qui permettaient un regroupement et donc une réutilisation aisée du code dans d'autres pages. Un des aspects négatifs est le trafic réseau que chaque événement devra générer.

Il est important de signaler que le compilateur n'est appelé que lors du premier accès à la page et qu'à ce moment du code MSIL est créé. Le compilateur ne sera rappelé que lorsque le contenu de la page aura été modifié par le programmeur.

Tout comme l'ASP ou le Javascript, l'essentiel se résume en une bonne connaissance des différents objets et de leur propriété. Les paragraphes suivant vont traiter ces objets.

Dans le code ASP.NET généré dans notre exemple précédent, nous retrouvons la méthode suivante:

```

override protected void OnInit(EventArgs e)
{
    InitializeComponent();
    base.OnInit(e);
}

```

Nous retrouvons dans la classe Page un événement appelé Init qui se chargera d'appeler la méthode de notre classe OnInit dans laquelle notre méthode personnelle InitializeComponent est appelée. Cet événement est déclenché avant tout chargement d'un contrôle. Cette fonction est donc la première à être exécutée.

Voici le contenu de la méthode InitializeComponent:

```
private void InitializeComponent()
{
    this.Voitures.SelectedIndexChanged += new
        System.EventHandler(this.ListeChange);
    this.Load += new System.EventHandler(this.Page_Load);
}
```

Cette méthode comprend la gestion des événements des différents contrôles placés sur notre page. Exemple: `this.Load += new System.EventHandler(this.Page_Load);`

Load est un événement qui est lié à la classe Page et qui est déclenché au chargement de la page lorsque tous les contrôles ont été chargés. Nous avons associé à cet événement l'exécution de la méthode Page_Load.

A la ligne 14, nous retrouvons l'attribut `AutoPostBack="True"` dans la balise select indiquant de ce fait que tout événement généré pour cet objet doit faire l'objet d'un envoi vers le serveur Web.

Exercice1: comment pouvons nous faire en sorte d'initialiser le contenu de notre liste déroulante de façon dynamique? Nous pourrions en effet imaginer que la gamme des véhicules disponibles figure dans une base de données.

Nous placerons notre code dans la méthode *Page_Load*.

Solution:

```
1     private void Page_Load(object sender, System.EventArgs e)
2     {
3     if (!this.IsPostBack)
4     {
5         ListItem v1 = new ListItem("Voiture V1", "V1");
6         ListItem v2 = new ListItem("Voiture V2", "V2");
7         ListItem v3 = new ListItem("Voiture V3", "V3");
8         Voitures.Items.Add(v1);
9         Voitures.Items.Add(v2);
10        Voitures.Items.Add(v3);
11    }
12 }
```

A la ligne 3, nous retrouvons le test `if(!this.IsPostBack)` permettant de savoir si le rechargement de la page est lié à une réponse envoyée par le serveur suite à un événement. Là également, nous retrouvons une différence notable avec l'ASP puisque notre exemple repris ci-dessus provoquait un rechargement des pages sans tenir compte du choix effectué dans la première liste déroulante. La technologie ASP.NET prévoit la sauvegarde des données présentes dans un formulaire et de ce fait, si nous n'effectuons pas ce test, nous risquons de retrouver une copie supplémentaire des items dans la liste déroulante.

Près le chargement de notre page HTML, nous pouvons en visualiser le contenu et nous retrouvons des champs cachés (attribut hidden) permettant d'indiquer au serveur le contenu des différents autres champs visibles présents dans votre formulaire pour les y replacer lors de la réponse envoyée suite à un événement.

Voici le contenu d'une partie du fichier:

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtOTAzNTMwODg0O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+Oz47bDx0PH
Q8O3A8bDxpPDE+O2k8Mj47aTwzPjs+O2w8cDxWb2l0dXJIIFYxO1YxPjtwPFZvaXR1cm
UgVjI7VjI+O3A8Vm9pdHVyZSBWMztWMz47Pj47Pjs7Pjs+Pjs+Pjs+TghyoZavyqGovDAg
GHdsyslFAXM=" />
```

2. L'utilisation des contrôles serveur.

2.1. Syntaxe.

Les contrôles serveur ASP.NET sont identifiés au sein d'une page à l'aide de balises dont la forme est semblable à celle-ci `<asp:ListItem runat="server">`. Tout comme pour les codes Javascript côté client avec le modèle d'objet de documents, chacune de ces balises peut recevoir un identificateur `id` permettant par programme la manipulation du modèle objet correspondant au moment de l'exécution.

Exemple repris du code précédent:

```
- dans le code HTML:
<asp:DropDownList id="Couleurs" runat="server">.
-dans le script c sharp:
Couleurs.Items.Insert(0,"Effectuer un choix");
```

2.2. La gestion des événements

Ces contrôles peuvent exposer et déclencher des événements que nous pouvons gérer dans notre code. Si nous prenons la première liste déroulante, nous avons pu lui associer un événement sous la forme du code suivant:

```
this.Voitures.SelectedIndexChanged += new
System.EventHandler(this.ListeChange);
```

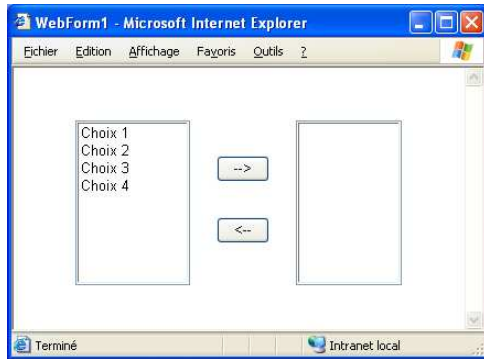
`Voitures` est le l'objet associé à la première liste déroulante.

`SelectedIndexChanged` est l'événement que l'on désire gérer. Cet événement est déclenché lorsque un changement de sélection est effectué dans la première liste déroulante.

`ListeChange` correspond au nom de la méthode devant être appelée lorsque l'événement survient. Cette méthode doit comprendre dans sa définition des arguments imposés qui seront identiques quelle que soit la méthode associée à un événement donné:

```
private void ListeChange(object sender, System.EventArgs e).
```

Exercice: Soit deux listes et deux boutons permettant de déplacer un item d'une liste à l'autre.



Les items dans la liste de gauche sont placés de façon statique dans le code.

Nous devons gérer les événements sur les deux boutons. Ils correspondent à des contrôles serveur que nous retrouvons sous la forme suivante:

```
<asp:Button id="Button1" style="Z-INDEX: 103; LEFT: 184px; POSITION: absolute; TOP: 80px" runat="server" Width="48px" Text="-->"></asp:Button>
<asp:Button id="Button2" style="Z-INDEX: 104; LEFT: 184px; POSITION: absolute; TOP: 136px" runat="server" Width="48px" Text="<--"></asp:Button>
```

Nous retrouvons dans les balises des feuilles de style permettant de positionner de façon absolue les ressources sur le page web. Les événements liés à ces ressources se trouvent dans le code csharp sous la forme suivante:

```
this.Button1.Click += new System.EventHandler(this.MoveLtoR);
this.Button2.Click += new System.EventHandler(this.MoveRtoL);
```

Voici le contenu d'une des méthodes permettant lors d'un click sur le bouton supérieur de transférer l'item sélectionné dans la liste de gauche vers la liste de droite.

```
1     private void MoveLtoR(object sender, System.EventArgs e)
2     {
3         int x = List1.SelectedIndex;
4         if (x >= 0)
5         {
6             List2.Items.Add(List1.Items[x].Text);
7             List1.Items.RemoveAt(x);
8         }
9     }
```

Commentaires:

La ligne 3 permet de récupérer l'index de l'item sélectionné. Dans le cas où aucune sélection n'a été effectuée, cette propriété renvoie la valeur -1.

List1.Items[x].Text permet de récupérer le texte correspondant à l'index [x] dans la liste. List2.Items.Add(...) permet d'ajouter un nouvel item dans la liste de droite dont le nom est renseigné sous la forme d'un string passé en paramètre.

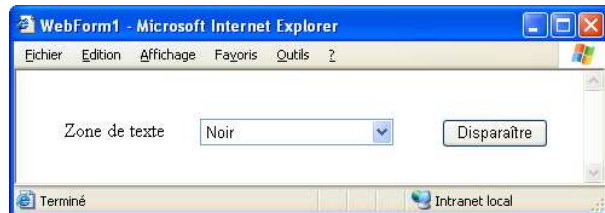
List1.Items.RemoveAt(x) permet de supprimer l'item venant d'être transféré dans la liste de droite.

2.3. L'utilisation des attributs.

Au même titre que les balises HTML, les balises ASP disposent d'attributs permettant de modifier l'apparence d'une ressource ou de texte. Prenons le cas d'un label que l'on désire positionner dans une de nos pages. Nous pouvons de façon dynamique modifier l'un de ces attributs en passant par l'objet associé à la balise correspondant à ce label. On peut donc

imaginer que l'on puisse paramétrer l'apparence d'une page en fonction d'un utilisateur donné et ainsi envisager la personnalisation de pages.

Exemple: Imaginons un label et une liste déroulante proposant des choix de couleur. Lorsque l'on change la sélection dans la liste, la couleur du label doit changer en fonction de la sélection. Nous ajouterons également un bouton permettant de faire disparaître ou réapparaître le label.



Nous devons retrouver deux événements: celui correspondant à un changement d'index dans la liste déroulante et celui de cliquer sur le bouton. Nous retrouvons la gestion de ces événements définie dans la partie de code suivante:

```
this.Couleurs.SelectedIndexChanged += new  
System.EventHandler(this.Change_color);  
this.Visibilite.Click += new System.EventHandler(this.Appdisp);
```

Voici le contenu des deux méthodes `Change_color` assurant le changement de la couleur du texte dans le label et `Appdisp` assurant l'apparition ou la disparition du bouton.

```
1 private void Appdisp(object sender, System.EventArgs e)  
2 {  
3     Couleurs.Visible = ! Couleurs.Visible;  
4     if (Couleurs.Visible == true) Visibilite.Text="Disparaitre";  
5     else Visibilite.Text="Apparaitre";  
6 }
```

Commentaires:

A la ligne 3, nous accédons à la propriété *Visible* de l'objet *Couleurs* associé à la liste déroulante. Cette propriété retourne un booléen correspondant à la valeur *true* si la ressource est visible sur la page ou *false* si elle n'est pas visible.

`Visibilite.Text` aux lignes 4 et 5 permet de modifier le texte qui figure sur le bouton. Cette propriété de l'objet *Visibilité* est liée à l'attribut `Text` de la balise correspondante. Lorsque la liste n'est pas visible c'est le texte 'apparaître' qui doit figurer sur le bouton tandis que quand la liste est visible, c'est le texte 'disparaître' qui doit être renseigné.

```
1 private void Change_color(object sender, System.EventArgs e)  
2 {  
3     int x = Couleurs.SelectedIndex;  
4     switch(x)  
5     {  
6         case 0:  
7             Label1.ForeColor=Color.Black;  
8             break;  
9         case 1:  
10            Label1.ForeColor=Color.Red;  
11            break;  
12        case 2:  
13            Label1.ForeColor=Color.Green;
```

```

14         break;
15     case 3:
16         Label1.ForeColor=Color.Blue;
17         break;
18     }
19 }

```

Commentaires:

Les lignes 7, 10 et 13 permettent de modifier la couleur du texte dans le label en jouant sur la propriété ForeColor de l'objet Label1.

Exercice: Imaginons le même style d'exercice ou nous retrouvons un label et des cases à cocher correspondant aux attributs de texte souligné, gras et italique. Le fait de cocher ou décocher une des cases modifie l'apparence du texte dans le label.



3. Validation des formulaires.

Un des objectifs des pages aspx est de pouvoir récupérer le contenu de formulaires et d'assurer la validité des données entrées par les différents champs. Des contrôles de validation sont ajoutés à un formulaire au même titre que les autres contrôles serveur. Nous retrouverons par exemple une vérification de plage ou le fait qu'un utilisateur doit nécessairement entrer une donnée dans un champ pour que le formulaire soit valide.

Nous pouvons attacher plusieurs contrôles de validation à un contrôle d'entrée. Pour chaque contrôle, nous retrouverons une propriété spécifique contenant la valeur à valider. Nous retrouvons dans le tableau ci après les différents contrôles d'entrée pouvant être validés ainsi que leur propriété.

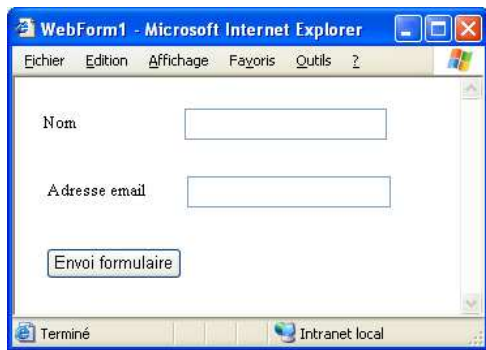
Contrôle	Propriété
HtmlInputText	Value
HtmlTexteArea	Value
HtmlSelect	Value
TextBox	Text
ListBox	SelectedItem.Value
DropDownList	SelectedItem.Value
RadioButtonList	SelectedItem.Value

Un des cas les plus fréquents est l'obligation demandée à un utilisateur d'entrer une information dans un des champs. Prenons comme exemple un formulaire dans lequel

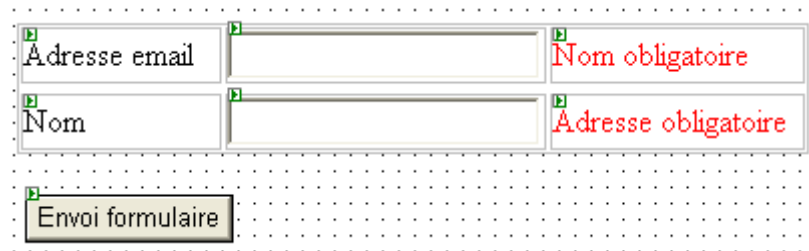
l'utilisateur doit introduire un nom et une adresse email obligatoires. Nous pouvons envisager sous ASP.NET que cette vérification se fasse en javascript c-à-d côté client ou uniquement côté serveur.

Exemple 1:

Nous analyserons dans une première étape la création de ce formulaire avec des scripts exécutés uniquement côté serveur. Voici le contenu de la page web:



Nous retrouvons deux contrôles de type label, deux contrôles de type TextBox ainsi qu'un bouton permettant l'envoi du formulaire. L'ensemble de ces contrôles sont des contrôles de type serveur. Analysons maintenant le code:



Du côté design, nous retrouvons deux contrôles supplémentaire que nous avons placés grâce à la boîte d'outils en sélectionnant *RequiredFieldValidator*. Ces contrôles possèdent la propriété *ControlToValidate* nous permettant grâce à une liste déroulante de sélectionner le contrôle devant faire l'objet d'une validation, la propriété *ErrorMessage* qui correspond au message d'erreur devant apparaître à l'emplacement du contrôle de validation si le champ à vérifier est vide et la propriété *EnableClientScript* permettant de renseigner si nous désirons une exécution de script pour la validation des champs côté client ou côté serveur.

Si nous visualisons le code HTML envoyé vers le navigateur, nous pouvons constater l'absence de javascript, tous les scripts de validation étant réalisés côté serveur. Comme la vérification se fait côté serveur, il est important de pouvoir déterminer si la validation des champs a renvoyé une erreur pour savoir si nous pouvons ou pas récupérer le contenu du formulaire en vue d'un traitement.

Nous allons donc gérer l'événement du clic sur le bouton d'envoi et ajouter en en tête du tableau un label dans lequel nous provoquerons l'affichage d'un message de validation ou de non validation de la page.

Voici le code correspondant à la fonction liée à cet événement:

1	<code>private void InitializeComponent()</code>
2	<code>{</code>
3	<code> this.Envoi.Click += new System.EventHandler(this.Envoi_Click);</code>
4	<code> this.Load += new System.EventHandler(this.Page_Load);</code>
5	
6	<code>}</code>
7	
8	
9	<code>private void Envoi_Click(object sender, System.EventArgs e)</code>
10	<code>{</code>
11	<code> if (this.IsValid)</code>
12	<code> Validation.Text="Page valide";</code>
13	<code> else</code>
14	<code> Validation.Text="Page non valide";</code>
15	<code>}</code>

Commentaires: la ligne 4 correspond à la mise en place de la gestion de l'événement Click sur le bouton Envoi. La ligne 11 permet de tester la validité de la page. Si un des contrôles n'est pas valide, la propriété *IsValid* renvoie une valeur *false* tandis que si tous les contrôles sont valides, nous retrouvons la valeur *true*.

Exemple 2: reprenons l'exemple précédent et adaptons le de sorte que nous puissions avoir une gestion par scripts côté client. Cette opération peut être rendue possible en modifiant la propriété *EnableClientScript* à *true*.

Lorsque nous laissons un champ incomplet et que nous cliquons sur le bouton envoi, les mêmes messages d'erreur apparaissent mais le label n'est pas mis à jour, la preuve que le contenu du formulaire n'est pas envoyé vers le serveur. Si nous éditons le contenu de la page html chargée, nous retrouvons les parties de code suivantes:

```
<input type="submit" name="Envoi" value="Envoi formulaire" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate();"
language="javascript" id="Envoi" />
```

Lorsque l'on clique sur le bouton, la fonction *Page_ClientValidate()* est exécutée. Cette fonction se situe dans le fichier *WebUIValidation.js* que l'on retrouve renseigné dans la ligne suivante:

```
<script language="javascript"
src="/aspnet_client/system_web/1_1_4322/WebUIValidation.js"></script>
```

L'objectif de cette partie de cours n'étant pas l'étude du javascript, nous nous limiterons à cette constatation sans aller plus loin dans l'analyse. C'est ce script qui, exécuté localement, se chargera d'analyser le contenu des contrôles et d'envoyer ou pas le contenu du formulaire vers le serveur.

Le tableau suivant reprend les différents contrôles de validation avec lesquels nous pouvons travailler.

Nom du contrôle	Description
RequiredFieldValidator	Garantit que l'utilisateur n'ignore pas d'entrée.
CompareValidator	Compare une entrée utilisateur avec une valeur de constante ou une valeur de propriété d'un autre contrôle à l'aide d'un opérateur de comparaison (inférieur à, égal à, supérieur à, etc).

RangeValidator	Vérifie qu'une entrée utilisateur est comprise entre les limites inférieures et supérieures. Vous pouvez vérifier les plages indiquées sous forme de paires de nombres, de caractères alphabétiques ou de dates. Les limites peuvent être exprimées sous la forme de constantes.
RegularExpressionValidator	Vérifie que l'entrée correspond à un modèle défini par une expression régulière. Ce type de validation vous permet de vérifier des séquences de caractères prévisibles, telles que celles figurant dans les numéros de sécurité sociale, les adresses de messagerie, les numéros de téléphone, les codes postaux, etc.
CustomValidator	Vérifie l'entrée de l'utilisateur à l'aide d'une logique de validation que vous codez personnellement. Ce type de validation vous permet de vérifier les valeurs dérivées au moment de l'exécution.
ValidationSummary	Affiche les erreurs de validation sous une forme récapitulative pour tous les validateurs d'une page.

Exercice 1: prévoir un formulaire comprenant une zone de texte dans laquelle on demande d'introduire un numéro de carte bancaire devant faire l'objet d'un test de validité.

Exercice 2: prévoir un formulaire dans lequel l'utilisateur doit entrer une adresse email dont il faut tester la validité.

4. Les objets serveur.

4.1. Introduction.

Si nous reprenons les exemples du paragraphe précédent, si nos formulaires ne présente pas d'erreur dans l'encodage des champs, le même formulaire est renvoyé vers l'utilisateur alors que nous souhaiterions que l'utilisateur soit redirigé vers une autre page. De plus, il serait intéressant que l'on puisse faire passer des paramètres de la page contenant le formulaire vers la page vers laquelle nous sommes redirigés. Pour ce, nous pouvons envisager l'utilisation d'objets propres au serveur pour assurer la gestion de variables ou la gestion de cookies fréquemment utilisés dans la gestion du "panier à provision" sur les sites commerciaux. Nous allons nous intéresser aux objets suivants:

- l'objet Request,
- l'objet Response,
- l'objet Session,
- l'objet Application
- l'objet Cache.

Nous consacrerons un paragraphe à chacun de ces objets. Nous y listerons l'ensemble des propriétés de l'objet ainsi qu'un exemple détaillé pour les propriétés que nous jugerons les plus utiles à développer.

4.2. L'objet *HttpRequest*.

4.2.1. Les propriétés et méthodes de l'objet.

Les propriétés de l'objet seront accédées au moyen de la propriété *Request* de l'objet *Page*. *HttpRequest* est implémenté dans l'espace de nom *System.Web*. Cet objet permet de récupérer des éléments postés par le client au travers du serveur.

Propriété	Description
<i>AcceptTypes</i>	Retourne un tableau de caractères contenant tous les types MIME supportés par le client.
<i>ApplicationPath</i>	Renvoie le chemin virtuel de l'application
<i>Browser</i>	Renvoie un objet <i>HttpBrowserCapabilities</i> qui contient des informations concernant le navigateur client.
<i>ClientCertificate</i>	Renvoie un objet <i>HttpClientCertificate</i> contenant des informations sur le certificat client.
<i>ContentLength</i>	Renvoie la taille en octets de la requête
<i>ContentEncoding</i>	Renvoie le jeu de caractères utilisé dans le corps
<i>Cookies</i>	Renvoie une collection <i>HttpCookiesCollection</i> contenant les cookies déposés sur le client
<i>CurrentExecutionFilePath</i>	Renvoie le chemin virtuel de la requête. Lorsque des redirections de pages ont été effectuées, le chemin renvoyé est l'original.
<i>FilePath</i>	Renvoie le chemin virtuel de la requête
<i>Files</i>	Renvoie une collection <i>HttpFileCollection</i> contenant la liste des fichiers à charger.
<i>Filter</i>	Retourne le filtre à utiliser lors de la lecture du flux entrant courant
<i>Form</i>	Retourne une collection <i>NameValueCollection</i> de variables postées d'un formulaire à l'aide de la méthode POST
<i>Headers</i>	Renvoie une collection <i>NameValueCollection</i> contenant les en-têtes de la requête
<i>HttpMethod</i>	Renvoie sous forme d'une chaîne de caractères la méthode qui a été utilisée pour transférer les données. typiquement, cette propriété peut renvoyer les valeurs GET, POST ou HEAD
<i>InputStream</i>	Renvoie un objet <i>Stream</i> (flux d'entrée) contenant le corps de la requête
<i>IsAuthenticated</i>	Renvoie une valeur booléenne indiquant qu'un utilisateur est authentifié par le serveur (true)
<i>IsSecureConnection</i>	renvoie une valeur booléenne indiquant que la requête est envoyée sur un espace sécurisé https(true) ou non (false)
<i>Params</i>	Renvoie une collection <i>NameValueCollection</i> contenant à la fois l'ensemble des données issues des collections <i>QueryString</i> , <i>Form</i> , <i>ServerVariables</i> , et les cookies.

Path	Renvoie sous forme de chaîne de caractères, le répertoire virtuel de la requête.
PathInfo	Renvoie des informations supplémentaires sur le chaîne de caractères contenant la requête URL envoyée
PhysicalApplicationPath	Renvoie le répertoire physique de la racine de l'application qui exécute la requête. Typiquement, cette racine est celle du site.
PhysicalPath	Renvoie le chemin physique de la page qui exécute la requête.
QueryString	Renvoie une collection NameValueCollection qui contient les paramètres envoyés dans la chaîne de la requête c-à-d également envoyés d'un formulaire via la méthode GET.
RawUrl	Renvoie l'URL virtuelle complète de la page ayant initié la requête
RequestType	Renvoie ou définit une chaîne de caractères contenant le type de méthode utilisée (GET ou POST)
ServerVariables	Renvoie une collection NameValueCollection contenant une liste de toutes les variables serveurs
TotalBytes	Renvoie le nombre total d'octets envoyés au serveur
Url	renvoie un objet URL contenant, par ses propriétés, des informations sur l'url de la requête: Port pour le port, Host pour le nom d'hôte et Scheme le type de protocole, etc.
UrlReferrer	Renvoie un objet Url contenant des informations sur la requête referant à la requête en cours
UserAgent	Renvoie une chaîne de caractères identifiant le navigateur utilisé
UserHostAddress	Renvoie l'adresse IP de l'utilisateur à l'origine de la requête
UserHostName	Renvoie le nom d'hôte de l'utilisateur à l'origine de la requête
UserLanguages	Renvoie un tableau de chaînes de caractères contenant la liste des langues supportées par le client.

L'objet Request comprend quelques méthodes:

MapImageCoordinate	Renvoie un tableau à deux dimensions contenant les positions x et y de la souris sur un champ image
MapPath	Renvoie le chemin physique d'un répertoire à partir de son emplacement virtuel
SaveAs	Enregistre la requête dans un fichier sur le disque du serveur.

Exemple 1: Nous allons envisager une page contenant un formulaire dont les champs seront initialisés sur base d'informations envoyées par l'url.

Prenons comme exemple la requête Websolution.aspx. Si nous désirons faire passer des paramètres par l'url, nous utiliserons la syntaxe suivante:

`http://localhost/Websolution.aspx?nom=dubois?prenom=damien`

Dans cette url, nous retrouvons deux champs facilement reconnaissables du fait qu'ils sont chaque fois précédés du caractère '?'. Nous retrouvons donc le champ nom et le champ prenom. A chacun de ces champs est affectée une valeur sous la forme champ=valeur. Pour le champ nom nous retrouvons la valeur dubois et pour le champ prenom la valeur damien.

Si nous souhaitons pouvoir récupérer les valeurs associées à chaque champ et les faire apparaître dans deux zones de texte présentes dans un formulaire, nous devons utiliser la propriété QueryString de l'objet Request.

Nous allons dans la méthode liée à l'événement de chargement de la page, ajouter le code suivant:

```
this.TextBox1.Text=Request.QueryString["nom"];  
this.TextBox2.Text=Request.QueryString["prenom"];
```

QueryString correspond à une collection et de ce fait, on peut accéder aux différents éléments en utilisant la notion d'indexeurs rencontrés dans la théorie du Csharp. Nous rencontrons deux surcharges: l'une réclamant un entier, l'autre réclamant le nom du champ duquel nous désirons récupérer la valeur.

Si nous analysons les requêtes de type GET envoyées par certains moteurs de recherche, nous pouvons nous apercevoir que tous les caractères spéciaux sont remplacés par le signe % suivi du code hexadécimal du caractère. Si nous effectuons une recherche sur le mot hésitation, nous retrouverons le code suivant dans l'url: q=h%C3%A9sitation. Si nous souhaitons transmettre des caractères accentués vers la page appelée, il nous faudra transcoder notre chaîne de caractères. Cette opération peut s'effectuer en utilisant la méthode `System.Web.HttpUtility.UrlEncode(chaîne)`.

Exemple 2: Supposons que l'on désire enregistrer dans un fichier d'archive l'ensemble des accès à notre formulaire. Nous désirons y renseigner le nom d'hôte, l'adresse IP ainsi que langue utilisée par le client sur son navigateur. L'accès à un fichier peut se réaliser au moyen de l'objet System.IO. Nous placerons le code dans la méthode associée à l'événement de chargement de la page.

```
1  if (!this.IsPostBack)  
2  {  
3      System.IO.FileStream FS = new  
        System.IO.FileStream(MapPath("suivi.txt"),System.IO.FileMode.Append,  
        System.IO.FileAccess.Write);  
4  
5      System.IO.StreamWriter fichier = new System.IO.StreamWriter(FS);  
6  
7      fichier.WriteLine("{0},{1},{2}",Request.UserHostName,Request.UserHostAddress,  
        Request.UserLanguages[0]);  
8  
9      fichier.Close();  
10     FS.Close();  
11 }
```

Commentaires:

La ligne 1 permet d'éviter l'ajout de ligne dans le fichier lorsqu'il s'agit de recharger la page à chaque réception du formulaire.

Les lignes 3 et 4 permettent d'ouvrir un flux sur un fichier et de définir un flux en écriture sur celui-ci. La ligne 4 permet l'utilisation d'écritures formatées tandis que l'objet FS seul ne permet que l'écriture sous forme d'un tableau de bytes. La ligne 7 permet d'écrire sur le flux en envoyant le nom d'hôte, l'adresse IP ainsi que la langue définis pour le navigateur client. Les lignes 9 et 10 permettent de fermer le flux et le fichier.

4.3. L'objet Response.

4.3.1. Les propriétés et méthodes de l'objet.

Les propriétés de l'objet seront accédées au moyen de la propriété Response de l'objet Page. HttpResponse est implémenté dans l'espace de nom System.Web. Cet objet permet d'envoyer des éléments vers le client au travers du serveur. Une des méthodes connue de cet objet est la méthode Write utilisée pour envoyer une chaîne de caractères vers le client.

Propriétés	Description
BufferOutput	Renvoie une valeur booléenne indiquant que la page renvoyée est mise en cache ou non
Cache	Renvoie un objet HttpCachePolicy donnant des détails sur la mise en cache
CacheControl	Permet de définir un cache public ou privé. Cette méthode est définie pour une raison de compatibilité avec asp. Il est préférable d'utiliser Cache
Charset	Renvoie ou définit le type de caractère utilisé à renvoyer
ContentType	Renvoie ou définit le type de contenu MIME de la réponse
Cookies	Renvoie une collection HttpCookieCollection de cookies dans la réponse
Expires	Permet de définir la durée de la mise en cache de la réponse. Cette méthode est utilisée pour des raisons de compatibilité avec asp. Nous utiliserons la méthode Cache.
ExpireAbsolute	Permet de définir la durée absolue de mise en cache de la réponse. Cette méthode est utilisée pour des raisons de compatibilité avec asp. Nous utiliserons la méthode Cache.
Filter	Renvoie ou définit un filtre à appliquer sur l'objet stream de sortie.
IsClientConnected	Renvoie une valeur booléenne indiquant que le client est toujours connecté à la page.
OutputStream	Renvoie un objet Stream représentant le corps de la réponse
Status	Définit l'état de la page à renvoyer vers le client
StatusCode	Définit le code d'état de la page à renvoyer vers le client
StatusDescription	Définit une description de l'état de la page à renvoyer vers le client
SuppressContent	Valeur booléenne indiquant que le contenu de la page doit être renvoyé au client

L'objet Response comprend les méthodes suivantes:

AddCacheItemDependencies	Permet d'ajouter une dépendance sous la forme d'un tableau d'items dans la cache. Si un de ces items est modifié, alors la mise en cache sera invalidée et la page sera rechargée.
AddCacheItemDependency	Permet d'ajouter un item sous la forme d'un lien dans la cache. Si cet item est modifié, alors la mise en cache sera invalidée et la page sera rechargée.
AddFileDependencies	Permet d'ajouter une dépendance sous la forme d'un groupe de fichiers. Si un des fichiers est modifié, alors la mise en cache sera invalidée et la page sera rechargée.
AddFileDependency	Permet d'ajouter une dépendance sous la forme d'un fichier. Si ce fichier est modifié, alors la mise en cache sera invalidée et la page sera rechargée.
AddHeader	Cette méthode a été conservée pour des raisons de compatibilité avec asp. Il faut utiliser à la place AppendHeader.
AppendHeader	Ajoute un en-tête personnalisé sous la forme clef, valeur.
AppenToLog	Ajoute une chaîne de caractère au fichier de log d'IIS
ApplyAppPathModifier	Ajoute une chaîne de caractères identifiant le numéro de session de l'utilisateur au chemin virtuel spécifié. Cette méthode s'avère utile lorsque l'on désire travailler sans cookies.
BinaryWrite	Ecrit des données binaires dans le flux de réponse.
Clear	Vide l'ensemble de la réponse
ClearContent	Vide le contenu de la réponse
ClearHeaders	Vide les en-têtes de la réponse
End	Renvoie tous les éléments du flux vers le client et stoppe l'exécution du script. La procédure Application_EndRequest est appelée à l'issue.
Flush	Renvoie tous les éléments du flux vers le client.
Pics	Ajoute une propriété PICS-LABEL à l'en-tête du flux sortant.
Redirect	Effectue une redirection de page vers l'URL spécifiée
Write	Ecrit dans le flux de sortie
WriteFile	Ecrit le contenu dans fichier spécifié dans le flux de sortie.

Exemple 1:

Nous avons dans les exemples précédents expliquer comment nous pouvions en utilisant la méthode GET envoyer des données vers une nouvelle page.

Lorsque un formulaire sans erreur est envoyé vers le serveur, nous pouvons donc envisager la redirection vers une autre page en attachant à son url des données.

Nous allons au niveau de notre code assurer la gestion de l'événement du clic sur le bouton d'envoi du formulaire. Nous retrouvons le code suivant:

```
1  private void InitializeComponent()  
2  {  
3      this.Button1.Click += new System.EventHandler(this.Button1_Click);  
4  }  
5  
6  private void Button1_Click(object sender, EventArgs e)  
7  {  
8      if (this.IsValid)  
9      {  
10         string adresse="http://localhost/websolution/webform1.aspx?nom=";  
11         adresse+= System.Web.HttpUtility.UrlEncode(this.TextBox1.Text);  
12         Response.Redirect(adresse);  
13     }  
14     else  
15     {  
16         this.Label1.Text="page non valide";  
17     }  
18 }
```

Commentaires:

La ligne 3 permet la mise en place de la gestion de l'événement dans la méthode InitializeComponent. On retrouve l'événement Click ainsi que la méthode associée qui est Button1_Click.

A partir de la ligne 6 nous retrouvons le contenu de la méthode qui sera donc appelée lors du clic sur le bouton d'envoi. A la ligne 8, nous testons si le formulaire est valide par rapport aux tests de validation mis en place sur la page. Dans le cas où nous n'avons aucune erreur, nous créons à la ligne 10 une chaîne de caractère contenant l'url de la page vers laquelle nous désirons nous brancher. Nous allons lui ajouter les données à transmettre par la méthode GET, c-à-d à travers l'url. La méthode UrlEncode va permettre de transformer les caractères spéciaux tels que les caractères accentués au format url.

4.3.2. La gestion des cookies.

Nous retrouvons dans les objets Response et Request la propriété Cookie permettant d'envoyer un cookie vers un client ou de récupérer un cookie qui provient d'un client. Un cookie est en fait un fichier texte qui est créé sur l'ordinateur client et dans lequel des informations sont stockées. La gestion des cookies vous permettra notamment d'enregistrer des informations concernant votre dernière visite sur un site donné, d'enregistrer vos différentes commandes sur un site commercial avant l'envoi global vers le serveur: on parle alors de panier à provision etc. Un cookie peut être entièrement défini par son nom, par sa valeur et sa date d'expiration.

Exemple 1: Soit un cookie devant enregistrer la date de votre dernier login sur le site ainsi que le nom ayant été utilisé pour vous authentifier. A chaque nouvelle visite, le serveur doit faire apparaître sur la page ces informations.

Pour la création des cookies ou leur modification, nous allons gérer l'événement du clic sur le bouton d'envoi. Nous trouverons dans la fonction associée à cet événement le code suivant:

```
1 private void Button1_Click(object sender, System.EventArgs e)
2 {
3     System.Web.HttpCookie nom = new
4         System.Web.HttpCookie("Nom", this.TextBox1.Text);
5     System.Web.HttpCookie last = new
6         System.Web.HttpCookie("Last", System.DateTime.Now.ToLongDateString());
7     nom.Expires=System.DateTime.Now.AddDays(30);
8     last.Expires=System.DateTime.Now.AddDays(30);
9     Response.Cookies.Add(nom);
10    Response.Cookies.Add(last);
11 }
```

Commentaires:

Les lignes 3 et 4 comprennent la création des deux cookies avec dans chaque constructeur le nom de la variable et la valeur affectée. Nous retrouverons donc:
Nom qui contient le nom entré par l'utilisateur
Last qui contient la date courante correspondant à l'envoi du formulaire
Les lignes 5 et 6 permettent de définir la date d'expiration des cookies. Nous ajouterons 30 jours à la date courante.
Ces deux entrées sont ajoutées au cookie qui sera envoyé vers le client.

Pour faire apparaître les informations sur la page chargée, nous allons devoir tester si les entrées dans le cookie sont présentes. Dans la positive, les valeurs seront récupérées et serviront à initialiser certains champs dans notre formulaire.

```
1 private void Page_Load(object sender, System.EventArgs e)
2 {
3     System.Web.HttpCookie nom = Request.Cookies.Get("Nom");
4     System.Web.HttpCookie last = Request.Cookies.Get("Last");
5     if (nom != null)
6         this.TextBox1.Text=nom.Value;
7     if (last != null)
8         this.Label1.Text=last.Value;
9 }
```

Commentaires:

Les lignes 3 et 4 permettent de récupérer le cookie du client. Nous utilisons pour cela la méthode Get avec comme paramètre le nom de la variable de laquelle nous devons récupérer la valeur associée.
Les lignes 5 et 7 permettent de tester si les variables existaient dans le cookie. Si ce n'est pas le cas, les références nom et last comprendront la valeur null. Dans le cas où ces variables existent, nous en récupérons les valeurs associées pour initialiser le contenu des champs TextBox1 et Label1.

4.3.3. La mise en cache de page.

Etant donné que la plupart des pages aspx sont des pages créées dynamiquement, il est intéressant pour des pages ne subissant que rarement des modifications d'être placées

en cache. Nous distinguons trois types de mise en cache à la disposition des applications Web:

- La mise en cache de sortie qui met en cache la réponse dynamique générée par une demande. Utile lorsque la page complète doit être cachée.
- La mise en cache par fragments, qui met en cache certaines parties d'une réponse générée par une demande. Utile lorsque une partie de page doit être cachée.
- La mise en cache de données qui permettant la mise en cache des objets arbitraires par programme.

Pour pouvoir activer la mise en cache de sortie, nous devons utiliser dans notre document la directive `@OutputCache` dont voici les principales propriétés.

```
<%@ OutputCache Duration="Nombre de secondes" Location="Any | Client ||  
Downstream | Server | None" VaryByControl="controlname"  
VaryByCustom="browser | Chaîne" VaryByHeader="headers"  
VaryByParam="NomParametres" %>
```

L'attribut `Location` spécifie l'endroit où les pages seront mises en cache. L'attribut `VaryByParam` permet de créer des mises en cache en fonction des paramètres envoyés à la page aussi bien avec la méthode GET qu'avec la méthode POST. Les valeurs possibles pour cet attribut sont `none`, `*`, `POST` et `GET`.

Exemple 1: mise en cache de sortie. Pour mettre en évidence cette cache, nous allons initialiser un label sur base de l'heure courante. Si la mise en cache se déroule bien, nous devrions à chaque demande de rafraîchissement du navigateur client obtenir la même heure.

```
< % @ OutputCache Duration="60" VaryByParam="none" %>
```

La durée de mise en cache est de 60 secondes et aucun paramètre GET ou POST n'a d'effet sur la mise en cache.

Nous avons repris un exemple d'envoi de données à une page via la méthode GET. Nous pourrions provoquer une mise en cache en fonction des paramètres passés. Soit l'url suivante comprenant des paramètres: `http://localhost/WebSol?nom=test`. Nous devons alors modifier la directive pour qu'elle contienne les attributs suivants:

```
< % @ OutputCache Duration="60" VaryByParam="nom" %>
```

Exercice: veuillez reprendre l'exemple précédent et mettre en évidence la mise en cache sur base des paramètres passés.

Exemple 2: mise en cache avec mise en place d'une dépendance par rapport à un fichier. Nous pouvons imaginer facilement l'utilité d'un tel mécanisme: un fichier xml duquel sont tirées de données pour la création dynamique d'une page. Tant que le fichier xml n'est pas modifié, la page html correspondante peut être maintenue en cache. Nous allons définir cette dépendance dans la méthode liée à l'événement du chargement de la page. En voici son contenu:

```

1 private void Page_Load(object sender, System.EventArgs e)
2 {
3     this.Label1.Text=System.DateTime.Now.ToLongTimeString();
4     System.Collections.ArrayList liste= new System.Collections.ArrayList(1);
5     liste.Add(MapPath("test.txt"));
6     Response.AddFileDependencies(liste);
7 }

```

Commentaires:

Les lignes 3, 4 et 5 permettent de créer une collection contenant les différents fichiers devant faire partie de la dépendance. La ligne 6 permet de mettre la dépendance en place. Il suffit maintenant de modifier le contenu du fichier pour s'apercevoir que l'heure se remet à jour dans le label présent dans le formulaire.

Nous évoquerons la mise en cache de contrôles utilisateurs et de données ultérieurement dans notre cours lorsque nous aurons abordé les contrôles utilisateurs et la gestion des bases de données.

4.4. L'objet Session

4.4.1. Introduction

Une session identifie la période de temps entre laquelle un utilisateur entre sur un site et le moment où il quitte son navigateur. Chaque utilisateur connecté sur un site a son propre identifiant unique sous forme d'une clef de session unique (SessionID). Cette clef est envoyée sous forme de cookie et est transmise à chaque requête envoyée vers le serveur dans la partie d'en-tête du message http. Ce cookie expire lorsque le navigateur se ferme.

Le seul inconvénient de ce système est lié à l'utilisation des cookies du fait que tout client peut avoir bloqué les cookies sur son navigateur. ASP.NET propose une solution alternative nommée cookieless permettant de gérer un état de session sans cookie.

4.4.2. Les propriétés et méthodes de l'objet.

Les propriétés de l'objet seront accédées au moyen de la propriété Session de l'objet Page.

Propriétés	Description
CodePage	Renvoie ou définit le numéro du code de page utilisé pour la session en cours.
Contents	Renvoie une référence vers l'objet Session en cours
Count	Renvoie le nombre d'éléments définis dans la session en cours
IsCookieless	Renvoie une valeur booléenne indiquant que l'état de session est géré par les cookies (false) ou non (true).
IsNewSession	Renvoie une valeur booléenne indiquant que la session vient d'être créée par la requête courante (true) ou non (false).
IsReadOnly	Renvoie une valeur booléenne indiquant que la session en cours est en lecture seule (true) ou non (false).

IsSynchronized	Renvoie une valeur booléenne indiquant que l'accès à la collection d'éléments de la session est synchronisé.
Item	Renvoie un objet de la session en cours identifié par son nom ou son index.
Keys	Renvoie une collection contenant la liste de toutes les clefs des objets de la session en cours.
LCID	Renvoie un identifiant du code de langue utilisé par la session en cours.
Mode	Renvoie une valeur indiquant comment l'état de session est enregistré.
SessionID	Renvoie un numéro unique de session identifiant une session donnée.
StaticObjects	Renvoie une collection d'objets créés par la balise OBJET dans le fichier global.asax
SyncRoot	Renvoie un objet pouvant être utilisé pour parcourir en accès synchrone les objets de la collection session
Timeout	Renvoie ou définit le temps maximal en minutes avant que la session ne soit détruite.

Les méthodes de l'objet Session sont les suivantes:

Méthodes	Description
Abandon	Détruit la session en cours
Add	Ajoute un nouvel élément à la session en cours.
Clear	Supprime tous les éléments de la session en cours.
CopyTo	Copie le contenu de la collection Session dans un tableau auquel vous pouvez spécifier l'index de départ
GetEnumerator	Renvoie une énumération contenant toutes les valeurs de la collection Session
Remove	Supprime un élément de la session sur base de son nom.
RemoveAll	Supprime tous les éléments de la session
RemoveAt	Supprime un élément de la session à partir de son index.

Exemple 1: nous avons dans une exemple précédent envisager l'échange de données entre page par l'utilisation de l'url par la méthode GET. Une autre méthode consiste à utiliser les variables de session pour autant que les pages appartiennent au même projet du fait que tout cookie ne peut être commun qui si on se trouve dans le même chemin virtuel. Dans le formulaire de départ, nous allons ajouter dans la méthode liée à l'événement du clic sur le bouton d'envoi, le code suivant:

```

1  if (this.IsValid)
2  {
3      if (Session["Nom"]==null)
4          Session["Nom"]=this.TextBox1.Text;
5      string adresse= "webform2.aspx";
6      Response.Redirect(adresse);
7  }

```

Commentaires:

La ligne 3 comprend le test d'existence de la variable d'environnement appelée Nom. Elle n'existe pas du fait que l'on récupère une référence sur l'objet de type null. Si la variable n'existe pas, elle est créée lors de son initialisation. On lui affecte une donnée de type string qui correspond au contenu du champ TextBox1.

La syntaxe de la ligne 4 est identique à la suivante:

```
Session.Add("Nom", this.TextBox1.Text);
```

Dans le code de la page webform2.aspx, nous retrouverons dans la méthode associée à l'événement du chargement de la page, le code suivant:

```
if (Session["Nom"]!=null)
    this.TextBox1.Text=(string) Session["Nom"];
else
    Response.Redirect("WebForm1.aspx");
```

Si la variable de session n'existe pas, nous redirigeons l'utilisateur vers la page WebForm1.aspx: généralement c'est situation est liée à un utilisateur qui a essayé d'accéder à la page directement sans passer par la page d'accueil.

4.5. L'objet Application.

Alors que l'objet session est propre à un utilisateur donné, l'objet application contient des méthodes et des propriétés partagées par tous les utilisateurs. Attention: lorsque l'application est arrêtée (nous évoquons ici le service IIS), tous les objets de la collection application sont détruits.

Lorsque l'on crée un site en utilisant l' IIS; une application est automatiquement associée au site et la racine de l'application est, dans ce cas, identique à celle du site. Lorsque nous prenons une des applications développées au cours, celle-ci correspond toujours à un répertoire virtuel associé au serveur pour lequel nous retrouvons:

- Un répertoire /bin dans lequel sont ajoutés les composants dll
- Un fichier web.config dérivé du fichier machine.config (propre au serveur) contenant des informations de configuration de l'application
- Un fichier global.asax incluant des méthodes liées à certains événements au niveau du serveur.

Nous envisagerons lors d'un des exercices de mettre en place un compteur reprenant l'ensemble des visiteurs connectés sur le site. C'est un cas de figure où la variable associée au comptage doit être accessible quel que soit l'utilisateur pour l'incrémenter lors d'une connexion ou la décrémenter lors d'une déconnexion.

4.5.1. Les propriétés et méthodes de l'objet.

Propriétés	Description
AllKeys	Renvoie un tableau contenant tous les éléments de la collection Application
Contents	Renvoie une référence vers l'objet Application en cours
Count	Renvoie le nombre d'éléments contenu dans la collection Application
Item	Renvoie un objet de l'application en cours identifié par

	son nom ou son index.
StaticObjects	Renvoie une collection d'objets créés au niveau Application par la balise OBJET dans le fichier global.asax

Les méthodes de l'objet Application sont les suivantes:

Méthodes	Description
Add	Ajoute un nouvel élément à la collection Application
Clear	Supprime tous les éléments de la collection Application
Get	Renvoie un élément de la collection application spécifié par son index ou son nom
GetKey	Renvoie le nom d'un élément de la collection Application en spécifiant son index
Lock	Verrouille l'objet Application en cours
Remove	Supprime un élément de la collection Application en spécifiant son nom
RemoveAll	Supprime tous les éléments de la collection Application
RemoveAt	Supprime un élément de la collection Application en spécifiant son index
Unlock	Déverrouille l'objet Application en cours.

Exercice 1: compteur de visiteurs connectés sur un site donné.

Pour mettre en place ce dispositif, nous devons utiliser le fichier global.asax. Si nous souhaitons créer ce compteur, nous devons créer une variable lors du démarrage de l'application. Cette variable sera incrémentée lorsque une session est ouverte et décrémentée lorsque une session se ferme. Il faudra naturellement tenir compte du Timeout: temps à partir duquel une session se ferme automatiquement lorsque un utilisateur n'a plus envoyé de requêtes sur le serveur.

Voici le code que nous retrouvons dans le fichier global.asax:

```

1  protected void Application_Start(Object sender, EventArgs e)
2  {
3      Application["compteur"]=0;
4  }
5  protected void Session_Start(Object sender, EventArgs e)
6  {
7      Application.Lock();
8      int temp = (int) Application["compteur"];
9      temp++;
10     Application["compteur"]=temp;
11     Application.UnLock();
12 }
13 protected void Session_End(Object sender, EventArgs e)
14 {
15     Application.Lock();
16     int temp = (int) Application["compteur"];
17     temp--;
18     Application["compteur"]=temp;
19     Application.UnLock();
20 }

```

Commentaires:

Nous retrouvons des méthodes associées à certains événements dans notre serveur:
`protected void Application_Start(Object sender, EventArgs e)` appelée lors du démarrage de l'application. C'est dans cette méthode que nous créons notre variable compteur initialisée à zéro.

`protected void Session_Start(Object sender, EventArgs e)` qui est appelée lors d'une ouverture de session par un utilisateur. C'est dans cette méthode que nous incrémentons la valeur de la variable d'application compteur.

`protected void Session_End(Object sender, EventArgs e)` qui est appelée lors de la fermeture d'une session. La valeur par défaut du timeout est de 20 minutes.

Comme la variable d'application compteur peut à la fois être accédée alors qu'un utilisateur quitte sa session et qu'un autre en ouvre une, il est indispensable que la modification du contenu de cette variable se fasse dans une zone protégée par un verrouillage. C'est la raison de la présence des deux lignes `Application.Lock()` et `Application.Unlock()`.

Exercice 2: insertion d'un pied de page de façon automatique. Si nous désirons qu'un pied de page identique soit inséré au bas de chacune des pages de notre site, nous pouvons jouer du copier/coller ce qui nécessitera un peu de travail lorsque nous désirerons y effectuer des modifications. L'idée est donc à nouveau d'utiliser le fichier `global.asax`. Nous allons dans un premier temps créer un fichier `PiedPage.htm` correspondant à notre pied de page et nous ferons en sorte de l'insérer en bas de chacune de nos pages accédée grâce à la méthode

`protected void Application_EndRequest(Object sender, EventArgs e)`

Nous retrouvons régulièrement cette façon de pratiquer chez les fournisseurs d'accès pour pouvoir insérer systématiquement des en-têtes ou des pieds de page au niveau de vos pages.

L'idée est d'ouvrir le fichier html comme un fichier texte et d'en envoyer le contenu vers le client en utilisant la propriété `Write` de l'objet `Response`. Voici le code:

```
1  protected void Application_EndRequest(Object sender, EventArgs e)
2  {
3      System.IO.FileStream FS = new
        System.IO.FileStream(Server.MapPath("PiedPage.htm"), System.IO.FileMode.
        Open);
4      System.IO.StreamReader SR = new System.IO.StreamReader(FS);
5      Response.Write(SR.ReadToEnd());
6      SR.Close();
7      FS.Close();
7  }
```

Exercice 3: compteur du total des visiteurs ayant déjà accédé au site.

Cet exercice est fort semblable à celui développé précédemment excepté qu'il ne faut pas décrémenter le compteur lors de la fermeture d'une session et qu'il faut veiller à sauvegarder cette valeur dans un fichier lors de la fermeture de l'application pour en récupérer la valeur lors du démarrage de l'application.

```
1  protected void Application_Start(Object sender, EventArgs e)
2  {
3      System.IO.FileStream FS = new
        System.IO.FileStream(Server.MapPath("Info.txt"), System.IO.FileMo
```

```

        de.Open);
4      System.IO.StreamReader SR = new System.IO.StreamReader(FS);
5      Application["total"] = Int32.Parse(SR.ReadLine());
6      SR.Close();
7      FS.Close();
8  }
9  protected void Session_Start(Object sender, EventArgs e)
10 {
11     int tmp = (int) Application["total"];
12     tmp++;
13     Application.Lock();
14     Application["total"] = tmp;
15     Application.UnLock();
16 }
17 protected void Application_End(Object sender, EventArgs e)
18 {
19     System.IO.FileStream FS = new
        System.IO.FileStream(Server.MapPath("Info.txt"), System.IO.FileMo
        de.Create);
20     System.IO.StreamWriter SW = new System.IO.StreamWriter(FS);
21     SW.WriteLine(Application["total"].ToString());
22     SW.Close();
23     FS.Close();
24 }

```

Commentaires:

La méthode `protected void Application_Start(Object sender, EventArgs e)` est appelée lors du démarrage de l'application. Nous y retrouvons aux lignes 3 et 4 l'accès à un fichier appelé `info.txt` dans lequel la première ligne contient la valeur correspondant au nombre clients connectés lors de la dernière sauvegarde. Cette sauvegarde s'effectue lors de l'arrêt de l'application. La donnée est enregistrée dans une variable d'application qui sera utilisée par la suite dans notre code.

La méthode `protected void Session_Start(Object sender, EventArgs e)` est appelée lors de l'ouverture d'une session. Nous y retrouvons l'incrément du compteur.

La méthode `protected void Application_End(Object sender, EventArgs e)` est appelée lors de l'arrêt de l'application. Nous sauvegardons dans le fichier `info.txt` le contenu de la variable d'environnement associée au nombre de visiteurs connectés.

5. La sécurité.

5.1. Introduction.

Un des points important dans la conception de sites Web pour certaines applications est certainement de pouvoir identifier les utilisateurs et de contrôler l'accès aux ressources. L'action de déterminer l'identité d'un utilisateur au moyen par exemple d'un nom et d'un mot de passe est connue sous le vocable authentification. Une fois un utilisateur authentifié, nous pouvons restreindre les accès à certaines ressources: on parle alors d'autorisation. Ces mécanismes peuvent être gérés au niveau du serveur IIS mais également au travers de l'ASP.NET. Certaines applications doivent être capables d'adapter le contenu de façon dynamique en fonction de l'identité à l'origine de la demande.

L'authentification se fait conjointement avec le serveur IIS. Elle peut être paramétrée dans le fichier `web.config` se situant dans la racine du répertoire où se situe votre application. Nous

retrouverons un format de fichier XML intégrant des balises dont celle permettant de configurer ce paramètre. Il s'agit de <authentication > dont l'attribut mode peut prendre une des valeurs suivant :

Forms	Pour une authentification par formulaire
Windows	Pour une authentification avec une fenêtre de type Windows
Passport	Avec une authentification par un serveur de passeports permettant via une authentification unique de pouvoir accéder à plusieurs sites.

En fonction d'un de ces choix, les codes d'accès (nom et mot de passe) se trouveront définis au niveau du même fichier XML pour l'authentification par formulaire, au niveau des comptes utilisateurs pour l'authentification Windows et au niveau du serveur de passeports pour le dernier.

5.2. Authentification par formulaire.

L'authentification basée sur des formulaires permet aux applications de proposer leur propre interface utilisateur qui peut s'intégrer au mieux dans un portail d'accueil et d'effectuer leur propre vérification des informations d'authentification. C'est une des techniques les plus courantes sur la plupart des sites internet. Pour l'utiliser dans notre application, nous devons choisir la valeur Forms pour l'attribut mode de la balise XML <authentication>.

Dans le conteneur <authentication> </authentication>, nous allons placer une balise <forms> dont voici la liste des différents attributs:

Attribut	Description
loginUrl	URL de connexion vers laquelle les utilisateurs non authentifiés sont redirigés. Elle peut se trouver sur le même ordinateur ou sur un ordinateur distant. Si elle se situe sur un ordinateur distant, les deux ordinateurs doivent utiliser la même valeur pour l'attribut decryptionkey .
name	Nom du cookie qui sera placé chez le client à des fins d'authentification. Si plusieurs applications souhaitent utiliser les services d'authentification basés sur des formulaires sur un ordinateur unique, elles doivent configurer chacune une valeur de cookie unique.
timeout	Durée d'expiration du cookie exprimée en minutes. La valeur par défaut est de 30.
path	Chemin d'accès à utiliser pour les cookies émis. La valeur par défaut est "/".
protection	Méthode utilisée pour protéger les cookies. Pour rappel, ces cookies contiennent vos codes d'accès et peuvent présenter un problème de sécurité si vous n'êtes pas le seul à utiliser votre ordinateur. <ul style="list-style-type: none">• All: utilise à la fois la validation et le cryptage pour protéger le cookie. C'est la valeur par défaut proposée.• None: le cryptage et la validation sont désactivés• Encryption: crypte le cookie à l'aide de tripleDES ou DES.• Validation: ne crypte pas le contenu du cookie, mais valide le fait que les données du cookie n'ont pas été altérées en chemin. Pour créer le cookie, la clé de

	validation est concaténée dans une mémoire tampon avec les données du cookie et un MAC est calculé, puis ajouté au cookie sortant.
--	--

Exercice 1: mise en place d'un formulaire d'authentification.

Nous allons dans une première phase configurer correctement le fichier web.config de notre application. Celui-ci contiendra entre autres, les balises suivantes:

```
<authentication mode="Forms" >
<forms name=".ASPXUSERDEMO" loginUrl="WebForm1.aspx" protection="All" timeout="60"
/>
</authentication>
<authorization>
<deny users="?" />
</authorization>
```

Nous créons dans une deuxième étape le formulaire de login qui sera donc accédé par le nom WebForm1.aspx. Attention si vous changez de nom de mettre à jour le fichier web.config au niveau de l'attribut loginUrl dans la balise forms.



La page Web comprend deux zones de texte servant à saisir le nom de l'utilisateur et son mot de passe ainsi qu'un bouton permettant de demander l'authentification. Nous nous sommes limités dans un premier temps à vérifier le nom et le mot de passe de façon statique dans le code. Voici le contenu de la méthode associée à l'événement du clic sur le bouton.

```
1 private void Verify_Click(object sender, System.EventArgs e)
2 {
3     if ((this.TextNom.Text=="test") && (this.TextPassword.Text=="password"))
4     {
5         FormsAuthentication.RedirectFromLoginPage(this.TextNom.Text, false);
6     }
7 }
```

Commentaires:

Nous utilisons à la ligne 5 la classe FormsAuthentication permettant de gérer l'authentification par formulaire. La méthode RedirectFromLoginPage permet de rediriger l'utilisateur vers la page d'origine vers laquelle il voulait se connecter avant d'être redirigé vers la page de login. Cette méthode permettra également la mise en place d'un cookie qui pourrait être temporaire ou permanent.

Nous avons dans notre exemple créé un deuxième formulaire dont l'accès est limité aux seuls utilisateurs authentifiés. Nous allons y placer un bouton de sorte que l'on puisse se

déconnecter. Dans cet exemple, l'authentification se base sur des données présentes dans le code. Nous envisagerons d'intégrer la gestion de bases de données plus tard dans le cours mais pour ceux qui ne désirent pas mettre en place un serveur SQL, il est possible de coder les noms et mots de passe dans le fichier XML web.config. Voici la syntaxe utilisée:

```
<authentication>
  <credentials passwordFormat="clear" >
    <user name="Mary" password="password"/>
    <user name="John" password="password"/>
  </credentials>
</authentication>
```

Dans la balise <credentials>, nous pouvons renseigner le format sous lequel le mot de passe est enregistré: clear, SHA1 ou MD5. Nous devons naturellement modifier le code utilisé dans la méthode `Verify_Click`.

```
1 private void Verify_Click(object sender, EventArgs e)
2 {
3     if (FormsAuthentication.Authenticate(this.TextNom.Text, this.TextPassword.Text)
4     {
5         FormsAuthentication.RedirectFromLoginPage(this.TextNom.Text, false);
6     }
7 }
```

Commentaires:

La méthode `FormsAuthentication.Authenticate` permet de vérifier la validité du mot de passe introduit sur base des informations présentes dans le fichier web.config. Elle retournera une valeur vraie si l'utilisateur est authentifié et fausse dans le cas contraire.

5.3. Authentification par une fenêtre Windows.

L'authentification Windows est fortement liée au serveur IIS avec lequel nous travaillons. Nous allons donc mettre en place cette technique en paramétrant le serveur et en adaptant des paramètres particuliers dans notre application.

Exercice 1: authentification par fenêtre Windows.

Soit un projet Weblogin2. Dès que le projet est créé, nous allons dans une première étape modifier les paramètres de notre serveur IIS en respectant les différentes étapes suivantes:

- 1- Démarrer – Panneau de configuration – outils d'administration
- 2- Sélectionner dans la fenêtre Services Internet IIS
- 3- Choisir l'application WebLogin2 et cliquer sur le bouton droit de la souris pour sélectionner les propriétés.
- 4- Choisir sécurité de répertoire – connexions anonymes et contrôles d'authentification, cliquer sur le bouton modifier
- 5- Choisir Authentification intégrée Windows et laisser les autres choix décochés.

Dans notre projet, nous allons modifier le fichier web.config de la façon suivante:

```
<authentication mode="Windows" />
<authorization>
    <deny users="willy\emmanuel" />
</authorization>
```

Nous devons modifier la propriété users de sorte d'y placer le nom d'utilisateur sous lequel nous sommes authentifié par le système d'exploitation par notre login.

Nous pouvons dès lors exécuter notre application. Pour l'accès à la page WebForm1.aspx, nous devons fournir un nom et un mot de passe qui peuvent être ceux de l'administrateur.

Nous pouvons modifier notre configuration pour interdire tous les utilisateurs sauf celui par exemple correspondant à willy\administrateur. Il est impératif de renseigner que la vérification se fait séquentiellement dans le fichier XML, ce qui signifie que nous devons en premier placer les balises <allow> et seulement ensuite la balise <deny> permettant d'interdire tout le monde.

```
<authorization>
<allow users="willy\administrateur"/>
<deny users="*" />
</authorization>
```

6. L'accès aux données côté serveur.

6.1. Introduction.

Lorsque l'on parle d'accès aux données sous .NET, on évoque souvent la technologie ADO.NET qui est le nom donné à un des composants .NET permettant d'accéder à une source de données. L'intérêt de ce composant est de proposer une interface indépendante du choix de la base de données et de rendre portable l'expérience acquise en programmation Windows vers la programmation Web puisque les mêmes objets existent.

Pour accéder à une base de données, nous devons respecter le canevas suivant:

- 1- Réaliser une connexion vers la source de données
- 2- Créer des commandes permettant, grâce à la connexion, d'envoyer des sélections, des suppressions, des modifications et des insertions d'enregistrements dans la source.
- 3- Création d'un objet servant de conteneur aux données sélectionnées par la commande.

La création des commandes se base sur la connaissance préalable du langage SQL.

L'utilisation de bases de données nous permet d'entrevoir la possibilité de création dynamique de pages mais également l'authentification ou la sauvegarde de formulaires envoyées par des utilisateurs.

6.2. Connexion vers une source de données.

Comme nous l'avons présenté dans notre introduction, une des premières étapes dans l'utilisation de données côté serveur est de créer un objet permettant la connexion vers la source de donnée. Nous disposons pour cela de deux familles d'objets:

- Les objets System.Data.SQL permettant l'accès à une base de données de type MS-SQL

- Les objets System.Data.OleDb permettant d'accéder par des serveurs OLE à d'autres types de bases de données différentes de MS-SQL.

Nos exemples se baseront sur des bases de données de type Access. Nous considérerons comme exemple la gestion du cercle des étudiants au niveau du bar. Nous créerons donc une table client comprenant les champs suivants:

- 1- Un champ de type caractère appelé *Nom* (50 caractères, non indexé)
- 2- Un champ de type caractère appelé *Prenom* (50 caractères, non indexé)
- 3- Un champ de type numérique auto incrémenté appelé *Reference*.

Une fois cette table créée, nous allons pouvoir envisager la création de notre projet WebClient dans lequel nous créerons notre connexion. Pour initialiser une nouvelle connexion, nous utiliserons un objet *Connection*. Les méthodes et les propriétés sont les suivantes:

Propriétés	Description
ConnectionString	Renvoie ou définit la chaîne de connexion à utiliser pour l'ouverture de la base.
ConnectionTimeout	Renvoie le temps maximal de réponse avant lequel une exception sera générée si l'ouverture de la connexion ne s'est pas faite
Database	Renvoie le nom de la base de données associée à l'objet.
DataSource	Renvoie la position de la base de données.
Provider	Renvoie le nom du fournisseur OleDb utilisé.
ServerVersion	Version du serveur auquel le client est connecté
State	Renvoie l'état de la connexion. Les valeurs possibles sont Broken, Closed, Connecting, Executing, Fetching, Open

Méthodes	Description
BeginTransaction	Commence une nouvelle transaction.
ChangeDatabase	Change la base de données en cours.
Close	Ferme la connexion.
CreateCommand	Crée ou renvoie un objet OleDbCommand associé à la connexion en cours.
GetOleDbSchemaTable	Renvoie le schéma de la base de données sous forme d'objet DataTable.
Open	Ouvre la connexion à la base de données définie dans le constructeur de l'objet ou par la propriété ConnectionString

Exercice 1: ouverture d'une connexion vers une base de données.

Nous pouvons créer les objets manuellement dans le code ou utiliser les assistants. Nous opterons pour la deuxième solution en mettant en analysant le code ayant été inséré dans notre projet. Nous ouvrons la boîte d'outils, Data et nous double cliquons sur la ressource OleDbConnection. Un objet est automatiquement instancié et nous pouvons en éditer les propriétés. Celles-ci comprennent la propriété ConnectionString pour laquelle nous cliquons sur la flèche pour finalement choisir dans la liste déroulante 'nouvelle connexion'.

Un assistant apparaît alors sous forme de quatre onglets:

Le premier onglet 'Fournisseur' nous permet de choisir le serveur ole correspondant à la source de données à laquelle nous désirons accéder.

Le deuxième onglet nous permettra de choisir dans la structure de notre disque dur la base de données avec laquelle nous désirons travailler.

Le troisième onglet nous permet de paramétrer les propriétés avancées telles que les paramètres réseau.

Le dernier onglet reprend un aperçu de toutes les propriétés de l'objet et leur valeur.

Voici le code inséré dans notre projet:

```
this.oleDbConnection1 = new System.Data.OleDb.OleDbConnection();
this.oleDbConnection1.ConnectionString =
@"Provider=Microsoft.Jet.OLEDB.4.0;Password="";User ID=Admin;Data
Source=C:\DatabaseSample\BAR.MDB;Mode=Share Deny None;Extended Properties="";Jet
OLEDB:System database="";Jet OLEDB:Registry Path="";Jet OLEDB:Database
Password="";Jet OLEDB:Engine Type=5;Jet OLEDB:Database Locking Mode=1;Jet
OLEDB:Global Partial Bulk Ops=2;Jet OLEDB:Global Bulk Transactions=1;Jet OLEDB:New
Database Password="";Jet OLEDB>Create System Database=False;Jet OLEDB:Encrypt
Database=False;Jet OLEDB:Don't Copy Locale on Compact=False;Jet OLEDB:Compact
Without Replica Repair=False;Jet OLEDB:SFP=False";
```

6.3. Lecture des données.

Il existe deux façons de pouvoir accéder aux données à travers la connexion ainsi réalisée. Soit nous laissons la connexion ouverte et nous récupérerons le flux d'informations au travers d'un autre objet de type DataReader. Comme cet objet fonctionne en mode connecté, il est impératif de le fermer à l'aide de la méthode Close.

Soit nous prenons le contenu de la table que nous transférons en mémoire dans un objet que l'on appelle un DataSet. Une fois cet objet rempli, la connexion vers la base de données peut être fermée. Que l'on passe par l'une ou l'autre des techniques, nous devons créer des commandes permettant la sélection, la suppression, l'insertion ou la mise à jour d'enregistrements dans notre base de données. Ces commandes sont mises en place par l'objet OleDbCommand et leur propriété CommandText nous permet de définir la requête SQL associée à la commande.

Exercice 1: Nous allons dans une première étape utiliser l'objet DataReader pour accéder à notre table Clients et ajouter dans notre formulaire des boutons permettant de nous déplacer dans les différents enregistrements ainsi que deux zones de texte permettant d'afficher le nom et le prénom du client.

Près avoir inséré dans notre projet un objet OleDbConnection, nous pouvons ajouter en utilisant la boîte d'outils, un objet OleDbCommand. Nous utiliserons la fenêtre des propriétés pour initialiser la propriété Connection de notre objet OleDbCommand1 ainsi que la propriété CommandText nous permettant de définir la requête SQL permettant d'effectuer la sélection des enregistrements dans la source de données. Voici le code inséré automatiquement dans notre projet:

```
this.oleDbCommand1.CommandText = "SELECT Clients.* FROM Clients";
this.oleDbCommand1.Connection = this.oleDbConnection1;
```

Nous plaçons sur notre formulaire deux zones d'éditations dont le nom de l'objet correspondant serait TextBoxNom et TextBoxPrenom. Nous initialiserons les deux zones dans la méthode appelée lors du chargement de la page.

```

1  this.oleDbConnection1.Open();
2  System.Data.OleDb.OleDbDataReader test = this.oleDbCommand1.ExecuteReader();
3  test.Read();
4  this.TextBoxNom.Text=test["Nom"].ToString();
5  this.TextBoxPrenom.Text=test["Prenom"].ToString();
6  test.Close();
7  this.oleDbConnection1.Close();

```

Commentaires:

Le code de la ligne 2 nous permet de récupérer un objet de type OleDbDataReader grâce à l'objet de commande. Nous appelons à la ligne 3 la méthode Read() nous permettant de demander la lecture de l'enregistrement courant et le positionnement sur l'enregistrement suivant. Une fois la ligne lue, nous pouvons récupérer le contenu de chacune des colonnes de la ligne lue pour initialiser les deux zones d'édition.

Un des problèmes que nous pouvons rencontrer dans ce type d'exercice est la nécessité de mémoriser la position que l'on occupe dans la table par rapport aux valeurs affichées dans notre formulaire. Il ne faut pas perdre de vue que la classe Form1 n'est instanciée que lorsque l'on va demander l'affichage de la page HTML correspondante. Une fois affichée, l'objet est détruit et de ce fait tout membre utilisé pour ce but de mémorisation de position n'aurait pas d'utilité puisque détruit entre deux rafraîchissements de la page.

Nous pouvons envisager deux solutions:

- 1- utilisation d'une variable liée à l'objet ViewState.
- 2- utilisation d'une variable de session permettant la sauvegarde de la donnée.

```

1  private void Prev_Click(object sender, System.EventArgs e)
2  {
3      int tmp;
4      tmp=Int32.Parse(this.ViewState["Position"].ToString());
5      if (tmp >1) tmp-=1;
6      this.ViewState["Position"]=tmp;
7      this.UpdateForm();
8  }
9
10 private void Next_Click(object sender, System.EventArgs e)
11 {
12     int tmp;
13     tmp=Int32.Parse(this.ViewState["Position"].ToString());
14     tmp+=1;
15     this.ViewState["Position"]=tmp;
16     this.UpdateForm();
17 }
18
19 public void UpdateForm()
20 {
21     int pos=1;
22     int i=0;
23     bool result=false;
24
25     if (this.ViewState["Position"]==null)
26         this.ViewState.Add("Position",1);
27     Else
28         pos=Int32.Parse(this.ViewState["Position"].ToString());
29
30     this.TextBox1.Text=pos.ToString();
31     this.oleDbConnection1.Open();
32     System.Data.OleDb.OleDbDataReader test =
33         this.oleDbCommand1.ExecuteReader();

```

```

34     while ((result=test.Read())&& (i++<pos))
35     {
36         if (result)
37         {
38             this.TextBoxNom.Text=test["Nom"].ToString();
39             this.TextBoxPrenom.Text=test["Prenom"].ToString();
40         }
41         this.ViewState["Position"]=i;
42     }
43     test.Close();
44     this.oleDbConnection1.Close();
45
46 }
47 }

```

Commentaires:

`this.ViewState.Add("Position",1)` permet d'ajouter une clef dans l'objet ViewState et de l'initialiser avec la valeur 1. Si la clef existe, la valeur de la clef est modifiée.

`if (this.ViewState["Position"]!=null)` permet de récupérer la valeur mémorisée dans la clef Position située dans le ViewState. Si cette clef n'existe pas encore, l'indexeur renvoie null.

Exercice 2: Nous allons maintenant développer le code permettant d'utiliser l'objet OleDbAdapter. Alors que l'objet OleDbDataReader nous offre une technique d'accès orientée connexion, l'objet OleDbDataAdapter nous permet de travailler sans connexion à savoir que le DataSet va contenir une copie en mémoire des enregistrements envoyés par la commande au travers de la connexion.

Nous pouvons avoir la même réflexion de savoir comment nous allons faire pour maintenir le DataSet présent en mémoire tout au long de la session qu'un utilisateur aura avec le serveur. La solution est de créer une variable que l'on utilisera dans le fichier global.asax. Bien qu'il était possible en ASP de disposer de fonctions permettant de se déplacer dans un RecordSet, avec asp.net et DataSet, cette possibilité a disparu assez curieusement du fait que sous Windows Forms, il existe le BindingContext permettant cette opération.

Nous allons donc ajouter une variable permettant de connaître le record sur lequel un utilisateur se trouve. Nous pouvons là aussi, intégrer une variable de session qui permettra d'individualiser la position par rapport à un utilisateur donné.

```

1     protected void Session_Start(Object sender, EventArgs e)
2     {
3         System.Data.DataSet Test = new System.Data.DataSet();
4         Session["MyDataSet"]=Test;
5         Session["Position"]=0;
6     }

```

Commentaires:

La méthode Session_Start est présente dans le fichier global.asax et est appelée lors de la création d'une nouvelle session utilisateur. Cette méthode comprend la création d'un objet de type DataSet et la création de deux variables de session: la première correspondant à une référence sur l'objet DataSet créé précédemment et l'autre permettant de déterminer la position de l'enregistrement qui est en cours de visualisation sur la page.

```

1 private void Page_Load(object sender, System.EventArgs e)
2 {
3     if (!this.IsPostBack)
4     {
5         System.Data.DataSet MyDataSet = (DataSet)Session["MyDataSet"];
6         this.oleDbConnection1.Open();
7         this.oleDbDataAdapter1.Fill(MyDataSet);
8         this.TextBox1.Text=MyDataSet.Tables["Clients"].Rows[0]
9         ["Nom"].ToString();
10        this.TextBox2.Text=MyDataSet.Tables["Clients"].Rows[0]
11        ["Prenom"].ToString();
12        this.oleDbConnection1.Close();
13    }
14 }

```

Commentaires:

Lors du chargement de la page correspondant à l'ouverture de la session, nous pouvons récupérer l'objet DataSet qui a été créé dans le fichier global.asax et ensuite le remplir sur base de l'objet OleDbDataAdapter. Les deux zones d'éditations sont initialisées avec le contenu des deux colonnes de la première ligne.

```

1 private void Next_Click(object sender, System.EventArgs e)
2 {
3     System.Data.DataSet MyDataSet = (DataSet)Session["MyDataSet"];
4     int i = Int32.Parse(Session["Position"].ToString());
5     if (i<MyDataSet.Tables["Clients"].Rows.Count-1)
6     {
7         i++;
8         Session["Position"]=i;
9     }
10    this.TextBox1.Text=MyDataSet.Tables["Clients"].Rows[i]["Nom"].ToString();
11    this.TextBox2.Text=MyDataSet.Tables["Clients"].Rows[i]["Prenom"].ToString();
12
13 }

```

Commentaires:

Cette méthode est liée à l'événement du simple clic sur le bouton du formulaire permettant de se déplacer sur l'enregistrement suivant.

Le code de la ligne 3 nous permet de récupérer l'objet DataSet créée dans le fichier global.asax. Nous pouvons à la ligne 4 récupérer également le contenu de la variable de session Position qu'il ne nous reste plus qu'à incrémenter pour autant que l'on ne dépasse pas le nombre maximum d'enregistrements présents dans le DataSet.

Les lignes 10 et 11 comprennent la mise à jour des deux zones d'édition avec le contenu des colonnes de la ligne d'indice i.

Exercice 3: utilisation des contrôles DataGrid et DataList.

Lorsque nous devons présenter l'ensemble des enregistrements renvoyés par une requête SQL et ce sous la forme d'un tableau dans lequel on devrait associer à chaque colonne un champ particulier et à chaque ligne un enregistrement, nous pouvons utiliser le DataGrid. Nous reprendrons notre exercice en supposant que l'ensemble des clients doit être présenté dans un DataGrid.

Nous devons tout comme les exercices précédents, ajouter un objet *Connection* ainsi qu'un objet *OleDbDataAdapter*. Une fois ce dernier créé, nous choisirons dans son menu contextuel l'option générer le groupe de données (DataSet) et nous verrons alors apparaître

un objet supplémentaire étant par défaut dans notre projet *dataset11*. Nous choisirons dans le menu contextuel l'option afficher le schéma nous permettant ainsi de faire apparaître la structure de la base de données soit sous forme graphique, soit sous forme de code XML. Nous verrons ultérieurement comment utiliser cette interface graphique pour créer des relations entre tables appartenant au même DataSet.

Partie 1: liaison du DataGrid avec notre source de données au travers d'un DataSet. Lorsque le DataGrid est placé dans notre formulaire à partir de la boîte d'outils, nous pouvons en éditer les propriétés:

Propriétés de l'objet DataGrid.

Propriétés	Description
AllowCustomPaging	Renvoie ou définit un booléen indiquant que la pagination personnalisée est active.
AllowPaging	Renvoie ou définit un booléen indiquant que la pagination des données est active. Permet de limiter le nombre d'enregistrements par page défini par la propriété PageSize.
AllowSorting	Renvoie ou définit un booléen indiquant que le tri des données est actif. Les titres des colonnes apparaissent alors comme des liens permettant ainsi de pouvoir prendre en charge l'événement du clic sur le lien.
AlternatingItemStyle	Renvoie ou définit le style des lignes alternées.
AutoGenerateColumns	Renvoie ou définit un booléen indiquant si la génération des colonnes est automatique c-à-d liée aux champs présents dans la base de données.
Columns	Renvoie une collection d'objets DataGridViewColumnCollection correspondant aux colonnes du tableau.
CurrentPageIndex	Renvoie ou définit l'index de la page de données affichée.
EditItemIndex	Renvoie ou définit le numéro de la ligne sélectionnée pour l'édition dans un tableau.
EditItemStyle	Renvoie ou définit le style (TableItemStyle) d'une ligne affichée dans le mode édition.
FooterStyle	Renvoie ou définit le style (TableItemStyle) du pied du tableau.
HeaderStyle	Renvoie ou définit le style (TableItemStyle) de l'entête du tableau.
Items	Renvoie une collection d'objets DataGridViewItems des éléments du tableau.
ItemStyle	Renvoie ou définit le style (TableItemStyle) des éléments du tableau.
PageCount	Renvoie le nombre total de pages (lorsque la pagination est active).
PageSize	Renvoie ou définit la taille d'une page en nombre d'éléments lorsque la pagination est active.
PagerStyle	Renvoie ou définit un objet DataGridViewPagerStyle définissant le style des sélecteurs de pages.
SelectedIndex	Renvoie ou définit l'index de l'élément qui est sélectionné.

SelectedItem	Renvoie un objet DataGridViewItem contenant des informations sur l'élément sélectionné.
SelectedItemStyle	Renvoie ou définit le style (TableItemStyle) des éléments du tableau.
ShowFooter	Renvoie ou définit une valeur booléenne indiquant que le pied du tableau est visible ou pas.
ShowHeader	Renvoie un booléen indiquant que l'en-tête du tableau doit être affiché ou pas.
VirtualItemCount	Renvoie le nombre total d'éléments lorsque la pagination personnalisée (CustomPaging) est activée.

L'ensemble de ces propriétés peut être accédé au travers d'un assistant graphique. Pour la liaison avec notre base de données, nous devons pratiquer de la sorte:

- 1- Editer la propriété *DataSource* et choisir l'objet DataSet que nous venons de créer.
- 2- Editer la propriété *DataMember* et choisir la table présente dans notre DataSet devant être utilisée pour remplir notre tableau.

Nous devons ajouter le code suivant dans la méthode appelée lors du chargement de la page:

```

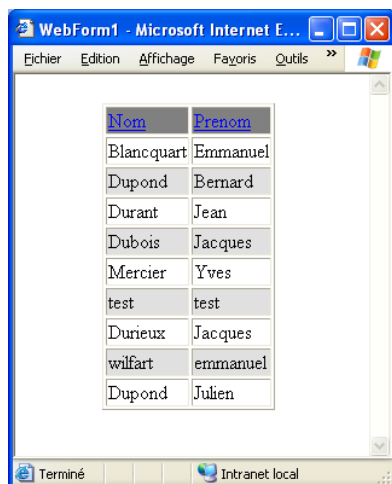
1     private void Page_Load(object sender, System.EventArgs e)
2     {
3         this.oleDbDataAdapter1.Fill(this.dataSet11, "Clients");
4         this.DataGrid1.DataBind();
5     }

```

Commentaires:

Le code de la ligne 3 permet de remplir le DataSet et le code de la ligne 4 permet de lier le contrôle à la source de données spécifiée par la propriété DataSource.

Partie 2: Nous reprendrons notre application précédente en ajoutant dans notre DataGridView la possibilité de tri en fonction d'un champ donné. Nous choisirons dans le menu contextuel du contrôle DataGridView l'option 'générer les propriétés' et nous obtenons alors un assistant. Nous validerons dans les propriétés générales l'option du comportement 'Autoriser le tri' ce qui générera dans l'en-tête du tableau des liens.



Nous pouvons envisager de cliquer sur le lien en espérant voir l'ordre des enregistrements changer. Cette action n'aura aucun effet, nous devons lier l'événement du clic sur ce lien avec l'exécution d'une méthode et c'est dans cette méthode que nous choisirons le type de tri avec lequel travailler. Voici le code correspondant:

```
1 private void DataGrid1_SortCommand(object source,
   System.Web.UI.WebControls.DataGridSortCommandEventArgs e)
2 {
3     this.oleDbDataAdapter1.SelectCommand.CommandText="select * from Clients
   order by "+e.SortExpression;
4     this.oleDbDataAdapter1.Fill(this.dataSet11,"Clients");
5     this.DataGrid1.DataBind();
6 }
```

Commentaires:

La solution envisageable pour modifier l'ordre de présentation des données est de changer la requête SQL associée à la sélection d'enregistrements pour y placer une clause Order By. Nous récupérons comme paramètre la référence 'e' nous permettant d'obtenir par la propriété SortExpression la chaîne de caractère associée à une colonne que nous avons définie dans l'assistant des propriétés de notre contrôle DataGrid (option colonnes Expression de tri).

Partie 3: Nous allons maintenant adapter notre exercice précédent pour limiter le nombre d'enregistrements qu'on peut visualiser sur une page avec notre contrôle DataGrid. Nous utiliserons pour paramétrer la pagination le menu contextuel du DataGrid dans lequel nous choisirons l'option Générateur de propriétés – pagination. Nous pourrons alors activer l'option de pagination et choisir le nombre d'enregistrements présents dans chacune des pages ainsi que les liens présents dans le pied de page nous permettant de nous déplacer d'une page à l'autre.

Lorsque ce choix est fait, il ne nous reste plus qu'à intégrer le code lié à l'événement. Pour ce, nous visualisons les événements liés au DataGrid et nous choisissons PageIndexChanged. Un double clic sur la zone correspondante permet l'insertion automatique de la méthode dans laquelle nous ajoutons le code suivant:

```
private void DataGrid1_PageIndexChanged(object source,
1 System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
2 {
3     this.DataGrid1.CurrentPageIndex=e.NewPageIndex;
4     this.oleDbDataAdapter1.Fill(this.dataSet11,"Clients");
5     this.DataGrid1.DataBind();
6 }
```

6.4. Modification des données.

Nous nous limiterons à l'emploi d'un contrôle DataGrid dans lequel nous envisagerons l'ajout d'une colonne supplémentaire offrant les possibilités d'édition, de modification, de validation ou d'annulation.

Nous sélectionnons dans le menu contextuel l'option générer les propriétés et nous nous positionnons sur la gestion des colonnes. Nous ajouterons une colonne de boutons de type 'Modifier, mettre à jour, annuler'. Une fois cette colonne ajoutée, nous pouvons éditer les événements liés au DataGrid et nous ajouterons une méthode pour la gestion des événements suivants: CancelCommand, EditCommand et UpdateCommand. Nous retrouverons le code suivant:

```

1 private void DataGrid2_EditCommand(object source,
  System.Web.UI.WebControls.DataGridCommandEventArgs e)
2 {
3     this.DataGrid2.EditItemIndex=e.Item.ItemIndex;
4     this.oleDbDataAdapter1.Fill(this.dataSet11);
5     this.DataGrid2.DataBind();
6 }
7
8 private void DataGrid2_CancelCommand(object source,
  System.Web.UI.WebControls.DataGridCommandEventArgs e)
9 {
10    this.DataGrid2.EditItemIndex=-1;
11    this.oleDbDataAdapter1.Fill(this.dataSet11);
12    this.DataGrid2.DataBind();
13 }
14
15 private void DataGrid2_UpdateCommand(object source,
  System.Web.UI.WebControls.DataGridCommandEventArgs e)
16 {
17     TextBox NomText = (TextBox)e.Item.Cells[0].Controls[0];
18     TextBox PrenomText = (TextBox)e.Item.Cells[1].Controls[0];
19     String Nom = NomText.Text;
20     String Prenom = PrenomText.Text;
21     this.oleDbDataAdapter1.Fill(this.dataSet11);
22     this.dataSet11.Tables["Clients"].Rows[e.Item.ItemIndex].BeginEdit();
23     this.dataSet11.Tables["Clients"].Rows[e.Item.ItemIndex]["Nom"]=Nom;
24     this.dataSet11.Tables["Clients"].Rows[e.Item.ItemIndex]["Prenom"]=Prenom;
25     this.dataSet11.Tables["Clients"].Rows[e.Item.ItemIndex].EndEdit();
26     this.oleDbDataAdapter1.Update(this.dataSet11,"Clients");
27     this.DataGrid2.EditItemIndex=-1;
28     this.DataGrid2.DataBind();
29 }

```

Commentaires:

Nous analyserons l'option de mise à jour. D'un point de vue visuel, nous pouvons nous apercevoir qu'une zone d'édition vient se superposer à chacune des cases de notre grille se trouvant dans un état d'édition. Nous pouvons récupérer le contenu de ces zones par le code utilisé aux lignes 17 et 18. Dès que ces données sont récupérées, nous pouvons placer notre ligne active de notre DataSet dans un mode d'édition (ligne 22: BeginEdit), ensuite affecter les différents champs de notre ligne avec ces données et finalement fermer le mode d'édition.

A ce stade, notre DataSet est modifié. Il ne reste qu'à forcer la mise à jour de notre base de données en exécutant la méthode Update de notre objet oleDbDataAdapter1.

6.5. Liaison entre plusieurs tables.

Nous allons envisager le même exercice mais en ajoutant un deuxième DataGrid dans notre formulaire: le premier associé aux clients et le deuxième devant contenir les différentes dettes d'un client donné correspondant à l'item sélectionné dans le premier DataGrid.

Nous retrouvons dans notre base de données deux tables: la table Clients et la table Dettes, ces deux tables étant liées par le champ 'Reference'.

Exercice : Utilisation des paramètres dans la requête de sélection des enregistrements de la table Dettes. Nous allons ajouter une colonne supplémentaire aux colonnes associées aux champs de notre table dette qui sera une colonne de boutons de type 'sélection'.

Dès que ce bouton est ajouté, nous allons définir une nouvelle méthode liée à l'événement de changement d'index dans la DataGrid contenant la liste des clients.

Voici le code associé à cette méthode:

```
1 private void DataGrid1_SelectedIndexChanged(object sender, System.EventArgs e)
2 {
3     this.oleDbDataAdapter1.Fill(this.dataSet11);
4     this.oleDbDataAdapter2.SelectCommand.Parameters["Reference"].Value=
5     this.dataSet11.Tables["Clients"].Rows[this.DataGrid1.SelectedIndex]["Reference"];
6     this.oleDbDataAdapter2.Fill(this.dataSet11,"Dettes");
7     this.DataGrid3.DataBind();
8 }
```

Commentaires:

Le code de la ligne 4 permet de récupérer l'index de la ligne sélectionnée permettant, dans notre DataSet, de récupérer le contenu de la ligne correspondante et d'ainsi pouvoir modifier le paramètre 'Reference' de la requête de sélection dans notre table Dettes et ensuite de pouvoir réinitialiser le contenu de notre DataGrid.

Nous modifierons notre code dans la méthode associée au chargement de la page pour qu'au démarrage, la DataGrid 'Dettes' soit correctement remplie.

```
1 private void Page_Load(object sender, System.EventArgs e)
2 {
3     if (!this.IsPostBack)
4     {
5         this.oleDbDataAdapter1.Fill(this.dataSet11,"Clients");
6         this.oleDbDataAdapter2.SelectCommand.Parameters["Reference"].Value=1;
7         this.oleDbDataAdapter2.Fill(this.dataSet11,"Dettes");
8         this.DataGrid1.DataBind();
9         this.DataGrid2.DataBind();
10        this.DataGrid3.DataBind();
11    }
12 }
```

Exercice 2: Nous allons modifier notre exemple précédent en mettant en place une relation entre tables dans notre DataSet typé. Nous utiliserons l'interface graphique pour mettre cette relation en place en sélectionnant le menu contextuel – ajouter – relation. Nous choisirons comme table parent la table 'Clients' et comme table enfant la table 'Dettes' en utilisant dans chacune des tables le champ Reference.

Nous supprimons dans notre DataGrid 'Clients' le lien entre la propriété DataSource et une éventuelle table, cette initialisation étant effectuée dans notre code et nous supprimons l'option de pagination pour la DataGrid 'Clients'.

```
1 private void DataGrid1_SelectedIndexChanged(object sender, System.EventArgs e)
2 {
3     this.oleDbDataAdapter1.Fill(this.dataSet11);
4     this.oleDbDataAdapter2.Fill(this.dataSet11,"Dettes");
5     this.DataGrid3.DataSource=this.dataSet11.Tables["Clients"].
6     DefaultView[this.DataGrid1.SelectedItem.ItemIndex].
7     CreateChildView("ClientsDettes");
8     this.DataGrid3.DataBind();
9 }
```

```
1 private void Page_Load(object sender, System.EventArgs e)
2 {
3     if (!this.IsPostBack)
```

```

4      {
5          this.oleDbDataAdapter1.Fill(this.dataSet11,"Clients");
6          this.oleDbDataAdapter2.Fill(this.dataSet11,"Dettes");
7          this.DataGrid3.DataSource=this.dataSet11.Tables["Clients"].
DefaultView[0].CreateChildView("ClientsDettes");
8          this.DataGrid1.DataBind();
9          this.DataGrid2.DataBind();
10         this.DataGrid3.DataBind();
11     }
12 }

```

7. Les services Web sous .NET.

Les services Web offrent la possibilité de partager des fonctions programmables sur Internet, celles-ci pouvant être utilisées par des abonnés au service au sein de leurs programmes ou de leur site web.

7.1. Création d'un service

Nous pouvons par l'interface de développement Visual Studio .NET, créer des services Web: il suffit pour cela, lors de la création d'un nouveau projet de choisir l'icône correspondante. Nous pouvons remarquer que la particularité du fichier est de porter l'extension asmx. Et de faire référence à l'espace de nom `System.Web.Services`. Nous retrouverons dans un service, une classe dérivée de la classe `System.Web.Services.WebService` et l'ensemble des méthodes qui doivent être rendues accessibles dans le service sont précédées de la syntaxe suivante:

```
[WebMethod (Description="Ceci est mon premier service")]
```

Pour `webMethod`, les attributs suivants peuvent être utilisés:

Attributs	Description
BufferResponse	Quand il est initialisé à true, cet attribut indique que la réponse générée par le service Web doit être bufférisée sur le serveur avant d'être envoyée vers le client. Cette technique permet de minimiser les communications entre le processus de travail et le processus IIS et de ce fait d'améliorer les performances. La valeur par défaut est true.
CacheDuration	Cette propriété autorise la mise en cache des résultats renvoyés par le service Web en spécifiant le nombre de secondes pendant lesquelles cette mise en cache reste effective. La valeur par défaut 0 permet de désactiver cette mise en cache.
Description	Cette propriété permet, sous la forme d'une chaîne de caractères, de fournir une description dans la page d'aide du service pour la méthode correspondante. Par défaut, cette chaîne est vide.
EnableSession	Cet attribut permet d'accéder directement à une collection d'états de session de <code>HttpContext.Current.Session</code> ou

	WebService.Session.
MessageName	Cette propriété permet de surcharger le nom de la méthode en utilisant un alias. Par défaut, c'est le nom de la méthode qui est employée.
TransactionOption	Cette propriété permet au service Web de participer comme objet racine à la transaction. Nous utiliserons comme valeur une énumération de type TransactionOption reprenant les valeurs suivantes: Disable, NotSupported, Required, Requires, Supported.

Une fois nos méthodes créées, nous pouvons tester notre service directement au travers de son Url. Par notre interface Visual Studio .NET, il suffit de démarrer le service comme s'il s'agissait d'une application.