Développement WEB avec ASP.NET 1.1

- Volume 2 -

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

Introduction

Le volume 1 du cours ASP.NET a présenté les notions fondamentales de la programmation web, celles que l'on retrouve tout le temps, quelque soit la technologie de développement utilisée (Java, Php, Asp.net). Nous avons en particulier présenté la méthode MVC (Modèle, Vue, Contrôleur) préconisée pour tout développement web. Le volume 1 s'est attardé essentiellement sur les parties Modèle et Contrôleur et peu sur les vues. Celles-ci étaient construites sans fioritures avec le langage HTML. ASP.NET amène avec lui une bibliothèque de composants permettant :

- la conception d'interfaces web riches et obtenues sans développement excessif
- la conception du contrôleur de l'application comme un ensemble de gestionnaire d'événements liés aux composants des vues

L'écriture d'une application web devient similaire à celle d'une application windows où on dessine des interfaces avant d'écrire les gestionnaires des événements que celles-ci peuvent générer. Cependant, lorsqu'on essaie de raccrocher ce mode de conception au modèle MVC, on s'aperçoit qu'ils ne vont pas ensemble. Il n'est pas possible d'avoir un unique contrôleur par lequel passent toutes les requêtes des clients, si l'application utilise des formulaires construits avec des composants ASP.NET. C'est fâcheux car il nous faut alors choisir entre deux modes de conception ayant chacun un intérêt. On peut cependant montrer qu'il est possible d'écrire des application utilisant les composants riches d'ASP.NET si on accepte d'avoir plusieurs contrôleurs au lieu d'un seul. Cette solution est souvent acceptable.

Ce document expose comment construire des vues à l'aide des composants ASP.NET appelés composants serveur. C'est un long travail. Il existe beaucoup de composants et certains sont très complexes. Ce volume 2 consacré aux seules vues est ainsi aussi important en taille que celui consacré aux notions fondamentales. C'est de plus complètement propriétaire puisque on ne retrouve pas les techniques qui vont être exposées ici, ni en Java ni en PHP. C'est pourquoi nous avons choisi de les isoler dans un volume à part, considérant qu'il fallait d'abord maîtriser les notions fondamentales de tout développement web avant de s'intéresser aux problèmes de construction de vues.

Le thème des composants serveur ASP.NET est vaste et ce volume 2 n'arrive pas à l'épuiser. Les composants HTML serveur n'ont pas été exposés. Ils le seront peut-être ultérieurement.

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

Serge Tahé

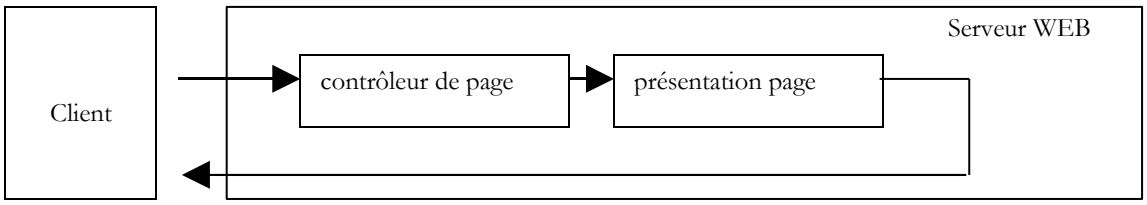
Juin 2004

1 Composants serveur ASP - 1

1.1 Introduction

Nous décrivons dans ce chapitre, la technologie préconisée dans ASP.NET pour construire l'interface utilisateur. Nous savons qu'il y a deux phases bien distinctes dans le traitement d'une page .aspx par le serveur web :

1. il y a tout d'abord exécution du contrôleur de la page. Celui-ci est constitué par du code situé soit dans la page .aspx elle-même (solution WebMatrix) soit dans un fichier à part (solution Visual Studio.NET).
2. puis le code de présentation de la page .aspx est exécuté pour être transformé en code HTML envoyé au client.



ASP.NET offre trois bibliothèques de balises pour écrire le code de présentation de la page :
1. les balises HTML classiques. C'est ce que nous avons utilisé jusqu'à maintenant.
2. les balises HTML serveur
3. les balises webforms

Quelque soit la bibliothèque de balises utilisée, le rôle du contrôleur de page ne change pas. Il doit calculer la valeur des paramètres dynamiques apparaissant dans le code de présentation. Jusqu'à maintenant, ces paramètres dynamiques étaient simples : c'étaient des objets de type [String]. Ainsi si dans le code de présentation on a une balise <%=nom%> :

- le contrôleur de page déclare une variable [nom] de type [String] et calcule sa valeur
- lorsque le contrôleur de page a terminé son travail et que le code de présentation est exécuté pour générer la réponse HTML, la balise <%=nom%> est remplacée par la valeur calculée par le code de contrôle

On sait que le découpage contrôleur/présentation est arbitraire et qu'on peut mélanger code contrôleur et code de présentation dans la même page. Nous avons expliqué pourquoi cette méthode était déconseillée et nous continuerons à respecter le découpage contrôle/présentation.

Les balises [HTML serveur] et [WebForms] permettent d'introduire dans le code de présentation, des objets plus complexes que le simple objet [String]. Cela présente parfois un réel intérêt. Prenons l'exemple d'un formulaire avec liste. Celle-ci doit être présentée au client avec un code html qui ressemble à ceci :

```
<select name="uneListe" size="3">
 <option value="opt1">option1</option>
 <option value="opt2">option2</option>
 <option value="opt3" selected>option3</option>
</select>
```

Le contenu de la liste et l'option à sélectionner sont des éléments dynamiques et doivent donc être générés par le contrôleur de page. Nous avons déjà rencontré ce problème et l'avons résolu en mettant dans le code de présentation la balise

```
<%=uneListeHTML%>
```

Cette balise va être remplacée par la valeur [String] de la variable [uneListeHTML]. Cette valeur calculée par le contrôleur devra être le code HTML de la liste, c.a.d. "<select name=..>...</select>". Ce n'est pas particulièrement difficile à faire et cela paraît une solution élégante qui évite de mettre du code de génération, directement dans la partie présentation de la page. Ici il y aurait une boucle, avec des tests à insérer dans celle-ci qui serait ainsi fortement "polluée". Néanmoins, cette méthode présente un inconvénient. Le découpage contrôle/présentation d'une page sert aussi à délimiter deux domaines de compétence :

- celui du développeur .NET qui s'occupe du contrôleur de page
- celui de l'infographiste qui s'occupe de la partie présentation de celle-ci

Ici, on voit qu'on a transféré la génération du code HTML de la liste dans le contrôleur. L'infographiste peut vouloir agir sur ce code HTML pour modifier l'aspect "visuel" de la liste. Il sera obligé de travailler dans la partie [contrôleur] et donc de sortir de son domaine de compétence avec le risque d'introduire malencontreusement des erreurs dans le code.

Les bibliothèques de balises serveur résolvent ce problème. Elles proposent un objet représentant une liste HTML. Ainsi la bibliothèque [WebForms] offre la balise suivante :

```
<asp:ListBox id="uneListe" runat="server"></asp:ListBox>
```

Cette balise représente un objet de type [ListBox] qui peut être manipulé par le contrôleur de page. Cet objet a des propriétés pour représenter les différentes options de la liste HTML et désigner l'option sélectionnée. Le contrôleur de page va donc donner les valeurs adéquates à ces propriétés. Lorsque la partie présentation va être exécutée, la balise

```
<asp:ListBox id="uneListe" runat="server"></asp:ListBox>
```

va être remplacée par le code HTML représentant l'objet [uneListe], c.a.d. le code "<select ..>...</select>". Pour l'instant, pas de différence fondamentale avec la méthode précédente sinon une façon de coder orientée objet, ce qui est intéressant. Revenons à notre infographiste qui a besoin de modifier le "look" de la liste. Les balises serveur ont des attributs de style (BackColor, Bordercolor, BorderWidth, ...) qui permettent de fixer l'aspect visuel de l'objet HTML correspondant. Ainsi on pourra écrire :

```
        <asp:ListBox id="ListBox1" runat="server" BackColor="#ffff99"></asp:ListBox></P>
```

L'intérêt est que l'infographiste reste dans le code de présentation pour faire ces modifications. C'est un avantage certain vis à vis de la méthode précédente. Les bibliothèques de balises serveur amènent donc des facilités dans la construction de la partie présentation des pages, ce qu'on a appelé dans les chapitres précédents, l'interface utilisateur. Ce chapitre a pour but de les présenter. On verra qu'elle propose parfois des objets complexes tels que des calendriers ou des tables liées à des sources de données. Elles sont extensibles, c.a.d. que l'utilisateur peut créer sa propre bibliothèque de balises. Il peut ainsi réaliser une balise générant un bandeau dans une page. Toutes les pages utilisant cette balise auront alors le même bandeau.

La génération HTML faite pour une balise s'adapte au type du navigateur client. Lorsque celui-ci fait une requête au serveur web, il envoie parmi ses entêtes HTTP, un entête [User-Agent: xx] où [xx] identifie le client. Voici un exemple :

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

Avec cette information, le serveur web peut connaître les capacités du client, notamment le type de code HTML qu'il sait gérer. Il y a eu en effet, au fil du temps, plusieurs versions du langage HTML. Les navigateurs récents comprennent les versions les plus récentes du langage, ce que ne savent pas faire des navigateurs plus anciens. En fonction de l'entête HTTP [User-Agent:] que le client lui a envoyé, le serveur lui enverra une version HTML qu'il saura comprendre. C'est une idée intéressante et utile car le développeur n'a alors pas à se préoccuper du type de navigateur client de son application.

Enfin, des IDE évolués comme Visual Studio.NET, WebMatrix, ... permettent une conception "à la windows" de l'interface Web. Ces outils non indispensables, apportent cependant une aide décisive au développeur. Celui-ci dessine l'interface web à l'aide de composants graphiques qu'ils déposent sur cette interface. Il a un accès direct aux propriétés de chacun des composants de l'interface qu'il peut ainsi paramétrer à sa guise. Ces propriétés seront traduites dans le code HTML de présentation de l'interface en attributs de la balise <asp:> du composant. L'intérêt pour le développeur est qu'il n'a pas à se rappeler ni la liste ni la syntaxe des attributs de chaque balise. C'est un avantage appréciable lorsqu'on ne connaît pas parfaitement les bibliothèques de balises serveur offertes par ASP.NET. Lorsque cette syntaxe est acquise, certains développeurs pourront préférer coder directement les balises dans le code de présentation de la page sans passer par la phase conception graphique. Un IDE n'est plus alors utile. Un simple éditeur de texte suffit. Selon la façon dont on travaille, l'accent est alors mis sur les composants (utilisation d'un IDE) ou les balises (utilisation d'un éditeur de texte). Il y a équivalence entre ces deux termes. Le composant est l'objet qui va être manipulé par le code de contrôle de la page. L'IDE nous donne accès à ses propriétés en phase de conception. Les valeurs données à celles-ci sont traduites immédiatement dans les attributs de la balise du composant dans le code de présentation. En phase d'exécution, le code de contrôle de la page va manipuler le composant et affecter des valeurs à certaines de ses propriétés. Le code de présentation va lui générer le code HTML du composant en utilisant d'une part les attributs fixés à la conception pour la balise serveur correspondante, d'autre part les valeurs des propriétés du composant calculées par le code de contrôle.

1.2 Le contexte d'exécution des exemples

Nous allons illustrer la conception des interfaces web à base de composants serveur avec des programmes dont le contexte d'exécution sera la plupart du temps le suivant :

1. l'application web sera composée d'une unique page P contenant un formulaire F,
2. le client fera sa première requête directement à cette page P. Cela consistera à demander l'url de la page P avec un navigateur. C'est donc une requête GET qui sera faite sur cette url P. Le serveur délivrera la page P et donc le formulaire F qu'elle contient,
3. l'utilisateur remplira celui-ci et le postera, c.a.d. qu'il fera une action qui forcera le navigateur à poster le formulaire F au serveur. L'opération POST du navigateur sera toujours à destination de la page P. Le serveur délivrera de nouveau la page P avec le formulaire F, le contenu de ce dernier ayant pu être modifié par l'action de l'utilisateur.

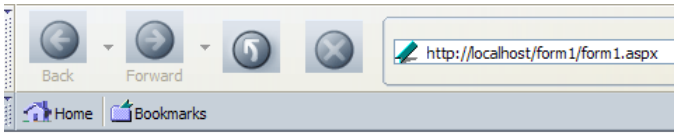
4. puis les étapes 2 et 3 reprendront.

C'est un processus d'exécution bien particulier en-dehors duquel, certains concepts exposés ci-après ne fonctionnent plus. Nous ne sommes plus dans un contexte d'architecture MVC dans lequel une application multi-pages est contrôlée par une page particulière que nous avons appelée "contrôleur d'application". Dans l'architecture MVC, le POST des formulaires a pour cible le contrôleur et non les formulaires eux-mêmes. Or nous allons voir que construire un formulaire avec des composants serveur, implique que ce formulaire soit posté à lui-même.

1.3 Le composant Label

1.3.1 Utilisation

La balise <asp:label> permet d'insérer un texte dynamique dans le code de présentation d'une page. Ca ne fait donc pas davantage que la balise <%=variable%> utilisée jusqu'à maintenant. L'étude de cette première balise va nous permettre de découvrir le mécanisme des balises serveur. Nous créons une page avec une partie contrôle [form1.aspx.vb] et une partie présentation [form1.aspx] . Il s'agit d'afficher l'heure :



Il est 18:55:13

Ce problème a déjà été traité dans la partie 1 de ce document et le lecteur est invité à s'y reporter s'il souhaite savoir comment il avait été traité. Le code de présentation [form1.aspx] est le suivant :

```
<%@ page src="form1.aspx.vb" inherits="form1" AutoEventWireup="false" %>
<HTML>
 <HEAD>
   <title>Webforms</title>
 </HEAD>
 <body>
   <asp:Label Runat="server" ID="lblHeure" />
 </body>
</HTML>
```

Nous introduisons la balise <asp:label>. Dans les bibliothèque de balises, l'attribut [runat="server"] est obligatoire. L'attribut ID identifie le composant. Le contrôleur devra le référencer avec cet identifiant. Le code du contrôleur [form1.aspx.vb] est le suivant :

```
Imports System.Web.UI.WebControls

Public Class form1
 Inherits System.Web.UI.Page

 Protected lblHeure As Label

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

 Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' met l'heure dans lblHeure
   lblHeure.Text = "Il est " + Date.Now.ToString("T")
 End Sub

End Class
```

Le contrôleur doit donner une valeur à l'objet [lblHeure] de type [System.Web.UI.WebControls.Label]. Tous les objets affichés par les balises <asp:> appartiennent à l'espace de noms [System.Web.UI.WebControls]. Aussi pourra-t-on importer systématiquement cet espace de noms :

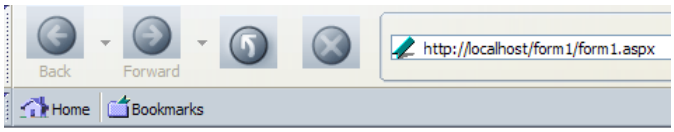
```
Imports System.Web.UI.WebControls
```

L'objet [Label] a différentes propriétés dont la propriété [Text] qui représente le texte qui sera affiché par la balise <asp:label> correspondante. Ici, nous mettons dans cette propriété l'heure courante. Nous le faisons dans la procédure [Form_Load] du contrôleur qui est systématiquement exécuté. Dans la procédure [Form_Init] qui elle aussi est toujours exécutée mais avant la procédure [Form_Load], nous mémorisons la requête du client dans un fichier [request.txt] du dossier de l'application. Nous aurons l'occasion d'examiner ce fichier pour comprendre certains aspects du fonctionnement des pages utilisant des balises serveur.

L'objet [Label] possède de nombreuses propriétés, méthodes et événements. Le lecteur est invité à lire la documentation sur la classe [Label] pour les découvrir. Dans la suite, il en sera toujours ainsi. Pour chaque balise, nous ne présentons que les quelques propriétés dont nous avons besoin.

1.3.2 Les tests

Nous plaçons les fichiers (form1.aspx, form1.aspx.vb) dans un dossier <application-path> et lançons Cassini avec les paramètres (<application-path>,/form1). Puis nous demandons l'url [http://localhost/form1/form1.aspx]. Nous obtenons le résultat suivant :



Il est 19:39:37

Le code HTML reçu par le navigateur est le suivant :

```
<HTML>
 <HEAD>
   <title>Webforms</title>
 </HEAD>
 <body>
   <span id="lblHeure">Il est 19:39:37</span>
 </body>
</HTML>
```

On voit que la balise serveur

```
   <asp:Label Runat="server" ID="lblHeure" />
```

a été transformée en le code HTML suivant :

```
   <span id="lblHeure">Il est 19:39:37</span>
```

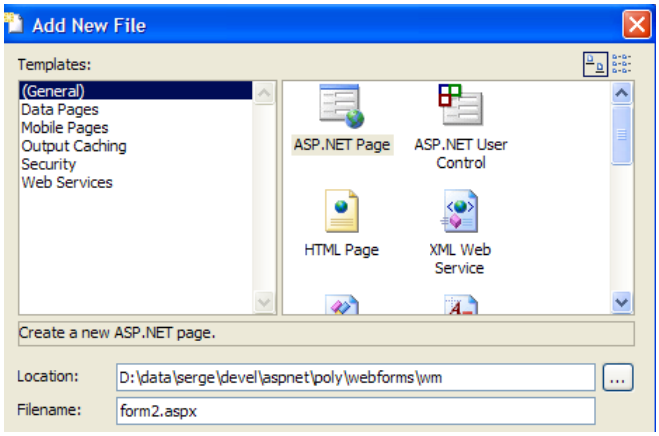
C'est la propriété [Text] de l'objet [lblHeure] qui a été placé entre les balises et . La requête faite par le client et mémorisée dans [request.txt] est la suivante :

```
GET /form1/form1.aspx HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp
eg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

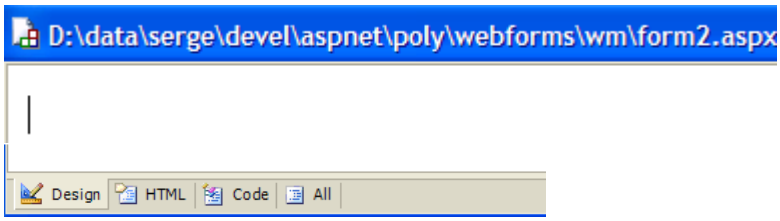
Rien que de très normal.

1.3.3 Construire l'application avec WebMatrix

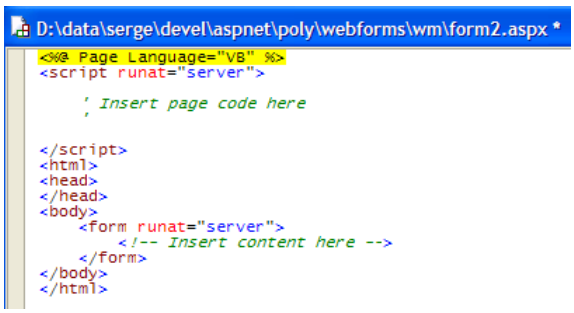
Nous avons construit le code de présentation de la page [form1.aspx] à la main. Cette méthode est utilisable si on connaît les balises. Un simple éditeur de texte suffit alors pour construire l'interface utilisateur. Pour débuter, un outil de conception graphique allié à une génération de code automatique est souvent nécessaire parce qu'on ne connaît pas la syntaxe des balises dont on a besoin. Nous construisons maintenant la même application avec l'outil WebMatrix. Une fois WebMatrix lancé, nous choisissons l'option [File/New File] :



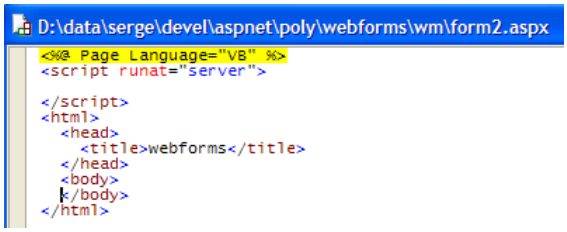
Nous créons une page ASP.NET appelée [form2.aspx]. Une fois validé l'assistant précédent, nous obtenons la fenêtre de conception de la page [form2.aspx] :



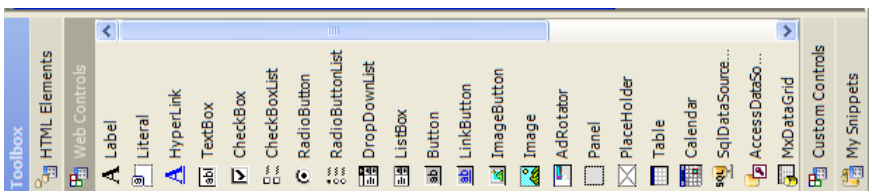
On rappelle que WebMatrix met le code de contrôle de la page et celui de présentation dans un même fichier, ici [form2.aspx]. L'onglet [All] présente le contenu de ce fichier texte. On peut constater dès maintenant qu'il n'est pas vide :



L'inconvénient de ce type d'outils est que souvent ils génèrent du code inutile. C'est le cas ici, où WebMatrix a généré une balise HTML <form>, alors qu'on ne va pas construire de formulaire... Par ailleurs, on peut constater que le document n'a pas de balise <title>. Nous remédions à ces deux problèmes sur le champ, pour obtenir la nouvelle version suivante :



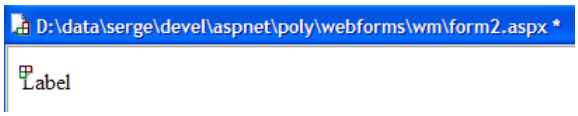
Ce que nous appelons le code contrôleur va venir s'insérer entre les balises <script> et </script>, ce qui assure une séparation au moins visuelle entre les deux types de code : contrôle et présentation. Nous revenons sur l'onglet [Design] pour dessiner notre interface. Une liste de composants est disponible dans une fenêtre d'outils à gauche de la fenêtre de conception :



La fenêtre d'outils offre l'accès à deux types de composants :

- les composants [WebControls] qui se traduisent par des balises <asp:>
- les composants [HTML Elements] qui se traduisent par des balises HTML classiques. Cependant, on peut ajouter aux attributs d'une balise HTML, l'attribut [runat="server"]. Dans ce cas, la balise HTML et ses attributs sont accessibles au contrôleur via un objet ayant pour propriétés celles de la balise HTML qu'il représente. On a appelé précédemment ces balises, les balises HTML serveur.

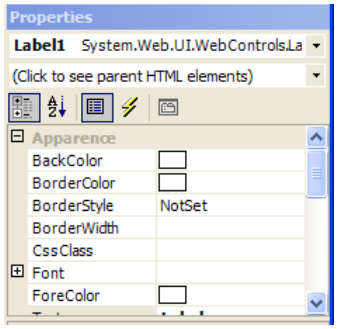
Double-cliquons sur le composant [Label] de la liste de contrôles [WebControls]. Dans l'onglet [Design] on obtient le résultat suivant :



Dans l'onglet [All] le code est devenu le suivant :

```
<%@ Page Language="VB" %>
<html>
<head>
    <title>webforms</title>
</head>
<body>
    <asp:Label id="Label1" runat="server">Label</asp:Label>
</body>
</html>
```

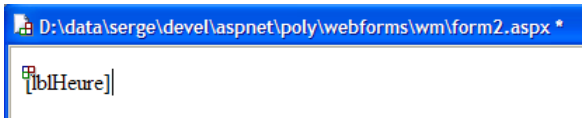
On peut remarquer tout d'abord qu'on a perdu la balise <script>. Une balise <asp:label> a été générée. Elle a un nom [Label1] et une valeur [Label]. Revenons sur l'onglet [Design] pour modifier ces deux valeurs. Nous y cliquons une fois sur le composant [Label] pour faire apparaître la fenêtre de propriétés de ce composant, en bas à droite :



Le lecteur est invité à consulter les propriétés de l'objet [Label]. Deux d'entre-elles nous intéressent ici :

- Text : c'est le texte que doit afficher le label - nous mettons la chaîne vide (c.a.d. rien)
- ID : c'est son identifiant - nous mettons **lblHeure**

L'onglet [Design] devient ceci :



et le code de [All] devient :

```
<%@ Page Language="VB" %>
<html>
<head>
    <title>webforms</title>
</head>
<body>
    <asp:Label id="lblHeure" runat="server"></asp:Label>
</body>
</html>
```

La partie présentation de la page est finie. Il nous reste à écrire le code de contrôle chargé de mettre l'heure dans la propriété [Text] de [lblHeure]. Nous ajoutons dans l'onglet [All] le code suivant :

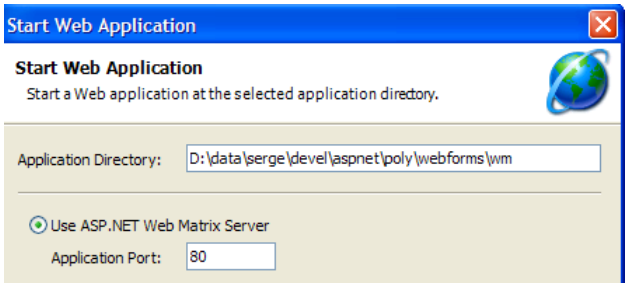
```
<%@ Page Language="VB" %>
```

```
<script runat="server">
 Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' met l'heure dans lblHeure
   lblHeure.Text = "Il est " + Date.Now.ToString("T")
 End Sub
</script>

<html>
<head>
    <title>webforms</title>
</head>
<body>
    <asp:Label id="lblHeure" runat="server"></asp:Label>
</body>
</html>
```

On remarquera que dans le code contrôleur, on ne déclare pas l'objet [lblHeure] comme il avait été fait précédemment :

```
 Protected lblHeure As New System.Web.UI.WebControls.Label
```

En effet, tous les composants serveur <asp:> de la partie présentation font l'objet d'une déclaration implicite dans la partie code de contrôle. Aussi les déclarer provoque une erreur de compilation, celle-ci indiquant que l'objet est déjà déclaré. Nous sommes prêts pour l'exécution. Nous prenons l'option [View/Start] ou le raccourci [F5]. Cassini est automatiquement lancé avec les paramètres suivants :



Nous acceptons ces valeurs. Le navigateur par défaut du système est automatiquement lancé pour demander l'url [http://localhost/form2.aspx]. Nous obtenons le résultat suivant :

Il est 20:56:38

Par la suite, nous utiliserons principalement l'outil WebMatrix afin de faciliter la construction et les tests des courts programmes que nous écrirons.

1.4 Le composant Literal

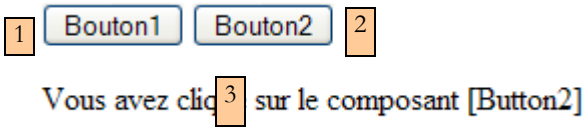
1.4.1 Utilisation

La balise <asp:literal> permet d'insérer un texte dynamique dans le code de présentation d'une page comme le fait la balise <asp:label>. Son principal attribut est [Text] qui représente le texte qui sera inséré tel quel dans le flux HTML de la page. Cette balise est suffisante si on n'a pas l'intention de mettre en forme le texte que l'on veut insérer dans le flux HTML. En effet si la classe [Label] permet de mettre en forme grâce à des attributs tels que [BorderColor, BorderWidth, Font, ...], la classe [Literal] n'a aucun de ces attributs. Le lecteur pourra reprendre intégralement l'exemple précédent en remplaçant le composant [Label] par un composant [Literal].

1.5 Le composant Button

1.5.1 Utilisation

La balise <asp:Button> permet d'insérer un bouton de type [Submit] dans un formulaire qui amène avec lui une gestion d'événements semblable à celle que l'on trouve dans les applications windows. C'est ce point qu'on veut approfondir ici. Nous créons la page [form3.aspx] suivante :



Cette page construite avec WebMatrix a trois composants :

n°	nom	type	propriétés	rôle
1	Button1	Button	text=Bouton1	bouton submit
2	Button2	Button	text=Bouton2	bouton submit
3	lblInfo	Label	text=	message d'information

Le code généré par WebMatrix pour cette partie est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">
</script>
<html>
<head>
    <title>asp:button</title>
</head>
<body>
    <form runat="server">
        <p>
            <asp:Button id="Button1" runat="server" Text="Bouton1"></asp:Button>
            <asp:Button id="Button2" runat="server" Text="Bouton2"></asp:Button>
        </p>
        <p>
            <asp:Label id="lblInfo" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

Nous retrouvons les composants [Button] et [Label] utilisés lors de la conception graphique de la page dans des balises <asp:>. Notons la balise <form runat="server"> qui a été générée automatiquement. C'est une balise HTML serveur, c.a.d. une balise HTML classique représentée néanmoins par un objet manipulable par le contrôleur. Le code HTML de la balise <form> sera généré à partir de la valeur que le contrôleur donnera à cet objet.

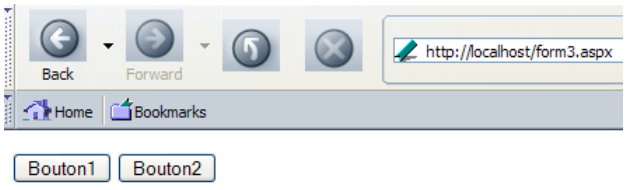
Ajoutons dans la partie contrôleur du code, la procédure [Page_Init] qui gère l'événement [Init] de la page. Nous y plaçons le code qui sauvegarde la requête du client dans le fichier [request.txt]. Nous en aurons besoin pour comprendre le mécanisme des boutons.

```
<script runat="server">
 Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
   ' sauve la requête courante dans request.txt du dossier de la page
   Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
   Me.Request.SaveAs(requestFileName, True)
 End Sub
</script>
```

On notera ci-dessus que nous n'avons pas mis la clause [Handles MyBase.Init] derrière la déclaration de la procédure [Page_Init]. En effet, l'événement [Init] de l'objet [Page] a un gestionnaire par défaut qui s'appelle [Page_Init]. Si on utilise ce nom de gestionnaire, la clause [Handles Page.Init] devient inutile. La mettre ne provoque cependant pas d'erreur.

1.5.2 Tests

Nous lançons l'application sous WebMatrix par [F5]. Nous obtenons la page suivante :



Le code HTML reçu par le navigateur est le suivant :

```
<html>
```

```
<head>
    <title>asp:button</title>
</head>
<body>
    <form name="_ctl0" method="post" action="form3.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwxNTY0NjIwMjUwOzs+2mcnJczeuvF2PEfvmtv7uiUhWUw=" />

        <p>
            <input type="submit" name="Button1" value="Bouton1" id="Button1" />
            <input type="submit" name="Button2" value="Bouton2" id="Button2" />

        </p>
        <p>
            <span id="lblInfo"></span>
        </p>
    </form>
</body>
</html>
```

Notons les points suivants :

- la balise <form runat="server"> s'est transformée en balise HTML

  ```
  <form name="_ctl0" method="post" action="form3.aspx" id="_ctl0">
  ```

  Deux attributs ont été fixés [method="post"] et [action="form3.aspx"]. Peut-on donner des valeurs différentes à ces attributs ? Nous essaierons d'éclaircir ce point un peu plus loin. On retiendra ici que le formulaire sera posté à l'url [form3.aspx]. Deux autres attributs [name, id] ont été également fixés. La plupart du temps, ils sont ignorés. Cependant si la page contient du code Javascript exécuté côté navigateur, l'attribut [name] de la balise <form> est utile.

- les balises <asp:button> sont devenues des balises HTML de boutons [submit]. Un clic sur l'un quelconque de ces boutons provoquera donc un "post" du formulaire [_ctl10] à l'url [form3.aspx].

- la balise <asp:label> est devenue une balise HTML

- un champ caché [__VIEWSTATE] a été généré avec une valeur bizarre :

  ```
  <input type="hidden" name="__VIEWSTATE" value="dDwxNTY0NjIwMjUwOzs+2mcnJczeuvF2PEfvmtv7uiUhWUw="
  />
  ```

  Ce champ représente sous une forme codée l'état du formulaire envoyé au client. Cet état représente la valeur de tous les composants de celui-ci. Comme [__VIEWSTATE] fait partie du formulaire, sa valeur sera postée avec le reste au serveur. Cela permettra à celui-ci de savoir quel composant du formulaire. Il déclenchera alors un événement du genre "le TextBox untel a changé de valeur". Cet événement sera traité par du code écrit par le développeur.

1.5.3 Les requêtes du client

Une fois obtenue la page [form3.aspx] dans le navigateur, redemandons-la puis regardons la requête qui a été envoyée par le navigateur pour l'obtenir. On se rappelle que notre application la mémorise dans le fichier [request.txt] dans le dossier de l'application :

```
GET /form3.aspx HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Accept: application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp
eg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

On a là un GET classique. Maintenons cliquons sur le bouton [Bouton1] de la page dans le navigateur. Il ne se passe apparemment rien. Pourtant nous savons que le formulaire a été posté. C'est le code HTML de la page qui le dit. Cela est confirmé par le nouveau contenu de [request.txt] :

```
POST /form3.aspx HTTP/1.1
Connection: keep-alive
Keep-Alive: 300
Content-Length: 80
Content-Type: application/x-www-form-urlencoded
```

```
Accept:                                                                                application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp
eg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/form3.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316

__VIEWSTATE=dDwxNTY0NjIwMjUwOzs%2B2mcnJczeuvF2PEfvmtv7uiUhWUw%3D&Button1=Bouton1
```

Le premier entête HTTP indique bien que le client a fait un POST vers l'url [/form3.aspx]. La dernière ligne montre les valeurs postées :

- la valeur du champ caché __VIEWSTATE
- la valeur du bouton qui a été cliqué

Si nous cliquons sur [Bouton2], les valeurs postées par le navigateur sont les suivantes :

```
__VIEWSTATE=dDwxNTY0NjIwMjUwOzs%2B2mcnJczeuvF2PEfvmtv7uiUhWUw%3D&Button2=Bouton2
```

La valeur du champ caché est toujours la même mais c'est la valeur de [Button2] qui a été postée. Le serveur peut donc savoir quel bouton a été utilisé. Il va s'en servir pour déclencher un événement qui pourra être traité par la page, une fois que celle-ci aura été chargée.

1.5.4 Gérer l'événement Click d'un objet Button

Rappelons le fonctionnement d'une page .aspx. Celle-ci est un objet dérivé de la classe [Page]. Appelons la classe dérivée [unePage]. Lorsque le serveur reçoit une requête pour une telle page, un objet de type [unePage] est instancié par une opération [new unePage(...)]. Puis ensuite, le serveur génère deux événements appelés [Init] et [Load] dans cet ordre. L'objet [unePage] peut les gérer en fournissant les gestionnaires d'événements [Page_Init] et [Page_Load]. Ensuite d'autres événements seront générés. Nous aurons l'occasion d'y revenir. Si la requête du client est un POST, le serveur générera l'événement [Click] du bouton qui a provoqué ce POST. Si la classe [unePage] a prévu un gestionnaire pour cet événement, celui-ci sera appelé. Voyons ce mécanisme avec WebMatrix. Dans l'onglet [Design] de [form3.aspx], double-cliquons sur le bouton [Bouton1]. Nous sommes alors amenés automatiquement dans l'onglet [Code], dans le corps d'une procédure appelée [Button1_Click]. Pour mieux comprendre, allons dans l'onglet [All] et regardons la totalité du code. Les modifications suivantes ont été apportées :

```
<%@ Page Language="VB" %>
<script runat="server">

...
    Sub Button1_Click(sender As Object, e As EventArgs)

    End Sub

</script>
<html>
...
<body>
    <form runat="server">
...
            <asp:Button id="Button1" onclick="Button1_Click" runat="server" Text="Bouton1"></asp:Button>
    </form>
</body>
</html>
```

Un nouvel attribut [onclick="Button1_Click"] a été ajouté à la balise <asp:Button> de [Button1]. Cet attribut indique la procédure chargée de traiter l'événement [Click] sur l'objet [Button1], ici la procédure [Button1_Click]. Il ne nous reste plus qu'à écrire celle-ci :

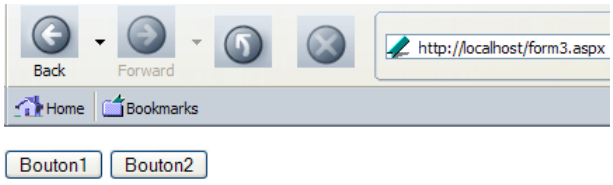
```
    Sub Button1_Click(sender As Object, e As EventArgs)
    ' clic sur bouton 1
    lblInfo.Text="Vous avez cliqué sur [Bouton1]"
    End Sub
```

La procédure met dans le label [lblInfo] un message d'information. Nous procédons de la même façon pour le bouton [Bouton2] pour obtenir la nouvelle page [form3.aspx] suivante :

```
<%@ Page Language="VB" %>
<script runat="server">

    Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
```

```
        ' sauve la requte courante dans request.txt du dossier de la page
        Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
        Me.Request.SaveAs(requestFileName, True)
    End Sub

    Sub Button1_Click(sender As Object, e As EventArgs)
        ' clic sur bouton 1
        lblInfo.Text="Vous avez cliqué sur [Bouton1]"
    End Sub

    Sub Button2_Click(sender As Object, e As EventArgs)
        ' clic sur bouton 2
        lblInfo.Text="Vous avez cliqué sur [Bouton2]"
    End Sub

</script>
<html>
<head>
    <title>asp:button</title>
</head>
<body>
    <form runat="server">
        <p>
            <asp:Button id="Button1" onclick="Button1_Click" runat="server" Text="Bouton1"></asp:Button>
            <asp:Button id="Button2" onclick="Button2_Click" runat="server" Text="Bouton2"></asp:Button>
        </p>
        <p>
            <asp:Label id="lblInfo" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

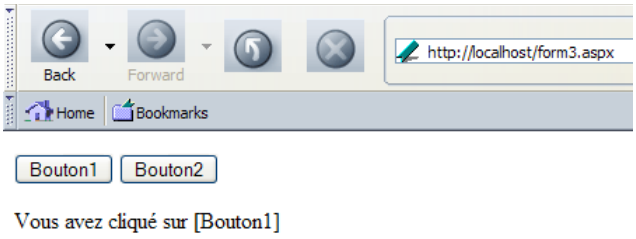
Nous lançons l'exécution par [F5] pour obtenir la page suivante :



Si nous regardons le code HTML reçu, nous constaterons qu'il n'a pas changé vis à vis de celui de la version précédente de la page :

```
<html>
<head>
    <title>asp:button</title>
</head>
<body>
    <form name="_ctl0" method="post" action="form3.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwxNTY0NjIwMjUwOzs+2mcnJczeuvF2PEfvmtv7uiUhWUw=" />

        <p>
            <input type="submit" name="Button1" value="Bouton1" id="Button1" />
            <input type="submit" name="Button2" value="Bouton2" id="Button2" />
        </p>
        <p>
            <span id="lblInfo"></span>
        </p>
    </form>
</body>
</html>
```

Si nous cliquons sur [Bouton1], nous obtenons la réponse suivante :



Le code HTML reçu pour cette réponse est le suivant :

```
<html>
<head>
    <title>asp:button</title>
</head>
<body>
    <form name="_ctl0" method="post" action="form3.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwxNTY0NjIwMjUwO3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDU+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPFZvdXMgYXZleiBjbGlxd
cOpIHN1ciBbQm91dG9uMV07Pj47Pjs7Pjs+Pjs+Pjs+4oO98Vd244kj0lPMXReWOwJ1WW0=" />

        <p>
            <input type="submit" name="Button1" value="Bouton1" id="Button1" />
            <input type="submit" name="Button2" value="Bouton2" id="Button2" />
        </p>
        <p>
            <span id="lblInfo">Vous avez cliqué sur [Bouton1]</span>
        </p>
    </form>
</body>
</html>
```

Nous pouvons constater que le champ caché [__VIEWSTATE] a changé de valeur. Cela reflète le changement de valeur du composant [lblInfo].

1.5.5 Les événements de la vie d'une application ASP.NET

La documentation ASP.NET donne la liste des événements générés par le serveur au cours de la vie d'une application :

- lors de la toute première requête que reçoit l'application, l'événement [Start] sur l'objet [HttpApplication] de l'application va être généré. Cet événement peut être géré par la procédure [Application_Start] du fichier [global.asax] de l'application.

Ensuite nous allons avoir une série d'événements qui vont se répéter pour chaque requête reçue :

- si la requête n'a pas envoyé de jeton de session, une nouvelle session est démarrée et un événement [Start] sur l'objet [Session] associé à la requête est généré. Cet événement peut être géré par la procédure [Session_Start] du fichier [global.asax] de l'application.
- le serveur génère l'événement [BeginRequest] sur l'objet [HttpApplication]. Il peut être géré par la procédure [Application_BeginRequest] du fichier [global.asax] de l'application.
- le serveur charge la page demandée par la requête. Il va instancier un objet [Page] puis générer deux événements sur cet objet : [Init] puis [Load]. Ces deux événements peuvent être gérés par les procédure [Page_Init] et [Page_Load] de la page.
- à partir des valeurs postées reçues, le serveur va générer d'autres événements : [TextChanged] pour un composant [TextBox] qui a changé de valeur, [CheckedChanged] pour un bouton radio qui a changé de valeur, [SelectedIndexChanged] pour une liste qui a changé d'élément sélectionné, ... Nous aurons l'occasion de mentionner les principaux événements pour chacun des composants serveur que nous allons présenter. Chaque événement E sur un objet nommé O peut être géré par une procédure portant le nom O_E.
- l'ordre de gestion des événements précédents n'est pas garanti. Aussi les gestionnaires ne doivent-ils faire aucune hypothèse sur cet ordre. On est cependant assuré que l'événement [Click] du bouton qui a provoqué le POST est traité en dernier.
- une fois la page prête, le serveur va l'envoyer au client. Avant, il génère l'événement [PreRender] qui peut être géré par la procédure [Page_PreRender] de la page.
- une fois la réponse HTML envoyée au client, la page va être déchargée de la mémoire. Deux événements seront générés à cette occasion [Unload] et [Disposed]. La page peut utiliser ces événements pour libérer des ressources.

L'application peut également recevoir des événements en-dehors d'une requête client :

- l'événement [End] sur un objet [Session] de l'application se produit lorsqu'une session se termine. Cela peut se produire sur une demande explicite du code d'une page soit parce que la durée de vie accordée à la session est dépassée. La procédure [Session_End] du fichier [global.asax] gère cet événement. On y libère en général des ressources obtenues dans [Session_Start].
- l'événement [End] sur l'objet [HttpApplication] de l'application se produit lorsque l'application se termine. Cela se produit notamment lorsqu'on arrête le serveur Web. La procédure [Application_End] du fichier [global.asax] gère cet événement. On y libère en général des ressources obtenues dans [Application_Start].

On retiendra les points suivants :

- le modèle événementiel précédent s'appuie sur des échanges HTTP client-serveur classiques. On le voit parfaitement lorsqu'on examine les entêtes HTTP échangés.
- le traitement des événements précédents se fait toujours côté serveur. Le clic sur un bouton peut bien sûr être traité par un script Javascript côté serveur. Mais il ne s'agit pas alors d'un événement serveur et on est là dans une technologie indépendante de ASP.NET.

A quel moment sont traités les événements, qu'ils soient traités du côté serveur (événements liés aux composants serveur) ou du côté navigateur par des scripts Javascript ?

Prenons l'exemple d'une liste déroulante. Lorsque l'utilisateur change l'élément sélectionné dans celle-ci, l'événement (changement de l'élément sélectionné) peut être traité ou non et s'il est traité il peut l'être à différents moments.

- si on souhaite le traiter immédiatement, on a deux solutions :
  - il peut être traité par le navigateur à l'aide d'un script Javascript. Le serveur n'intervient alors pas. Pour que cela soit possible, il faut que la page puisse être reconstruite avec des valeurs déjà présentes dans la page.
  - il peut être traité par le serveur. Pour cela, il n'y a qu'une solution : le formulaire doit être envoyé au serveur pour traitement. On a donc une opération [submit]. On verra que dans ce cas, on utilise un composant serveur appelé [DropDownList] et on fixe son attribut [AutoPostBack] à [true]. Cela veut dire qu'en cas de changement de l'élément sélectionné dans la liste déroulante, le formulaire doit être immédiatement posté au serveur. Le serveur génère dans ce cas, pour l'objet [DropDownList] un code HTML associé à une fonction Javascript chargée de faire un [submit] dès que l'événement "changement de l'élément sélectionné" se produit. Ce [submit] postera le formulaire au serveur et dans celui-ci, des champs cachés seront positionnés pour indiquer que le [post] provient d'un changement de sélection dans la liste déroulante. Le serveur génèrera alors l'événement [SelectedIndexChanged] que la page pourra gérer.
- si on souhaite le traiter mais pas immédiatement, on fixe l'attribut [AutoPostBack] du composant serveur [DropDownList] à [false]. Le serveur génère dans ce cas, pour l'objet [DropDownList] le code HTML classique d'une liste <select> sans fonction Javascript associée. Rien ne se passe alors lorsque l'utilisateur change la sélection de la liste déroulante. Cependant, lorsqu'il va valider le formulaire par un bouton [submit] par exemple, le serveur va pouvoir reconnaître qu'il y a eu un changement de sélection. Nous avons en effet vu que le formulaire envoyé au serveur avait un champ caché [__VIEWSTATE] représentant sous une forme codée l'état de tous les éléments du formulaire envoyé. Lorsque le serveur reçoit le nouveau formulaire posté par le client, il va recevoir ce champ parmi les éléments postés. Il y trouvera l'élément qui était sélectionné dans la liste lorsque le formulaire a été envoyé au navigateur. Parmi les éléments postés par le navigateur dans la requête suivante, il y aura le nouvel élément sélectionné dans la liste. Le serveur Web / ASP.NET va ainsi pouvoir vérifier si la liste déroulante a changé ou non d'élément sélectionné. Si oui, il va générer l'événement [SelectedIndexChanged] que la page pourra alors gérer. Pour distinguer ce mécanisme du précédent, certains auteurs disent que l'événement "changement de sélection" a été mis "en cache" lorsqu'il s'est produit sur le navigateur . Il ne sera traité par le serveur que lorsque le navigateur lui postera le formulaire, à la suite souvent d'un clic sur un bouton [submit].
- enfin si on ne souhaite pas traiter l'événement, on fixe l'attribut [AutoPostBack] du composant serveur [DropDownList] à [false] et on n'écrit pas le gestionnaire de son événement [SelectedIndexChanged].

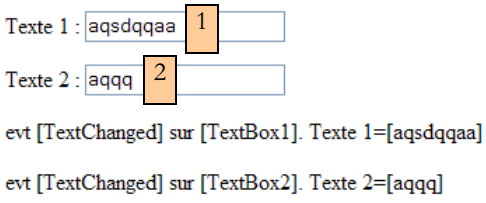
Une fois compris le mécanisme de traitement des événements, le développeur ne concevra pas une application web comme une application windows. En effet, si un changement de sélection dans un combobox d'une application windows peut être utilisé pour changer immédiatement l'aspect du formulaire dans lequel se trouve celui-ci, on hésitera davantage à traiter cet événement immédiatement dans une application web, s'il implique un "post" du formulaire au serveur donc un aller-retour réseau entre le client et le serveur. C'est pourquoi, la propriété [AutoPostBack] des composants serveur est mise à [false] par défaut. Par ailleurs, le mécanisme [AutoPostBack] qui s'appuie sur des scripts javascript générés automatiquement par le serveur web dans le formulaire envoyé au client ne peut être utilisé que si on est sûr que le navigateur client a autorisé l'exécution des scripts javascript sur son navigateur. Les formulaires sont donc souvent construits de la façon suivante :

- les composants serveur du formulaire ont leur propriété [AutoPostBack] à [false]
- le formulaire a un ou des boutons chargés de faire le [POST] du formulaire
- on écrit dans le code contrôleur de la page, les gestionnaires des seuls événements qu'on veut gérer, le plus souvent l'événement [Click] sur un des boutons.

1.6 Le composant TextBox

1.6.1 Utilisation

La balise <asp:TextBox> permet d'insérer un champ de saisie dans le code de présentation d'une page. Nous créons une page [form4.aspx] pour obtenir la présentation suivante :



Cette page construite avec WebMatrix a les composants suivants :

n°	nom	type	propriétés	rôle
1	TextBox1	TextBox	AutoPostback=true Text=(rien)	champ de saisie
2	TextBox2	TextBox	AutoPostback=false Text=(rien)	champ de saisie
3	lblInfo1	Label	Text=(rien)	message d'information sur le contenu de [TextBox1]
3	lblInfo2	Label	Text=(rien)	message d'information sur le contenu de [TextBox2]

Le code généré par WebMatrix pour cette partie est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">
</script>
<html>
<head>
    <title>asp:textbox</title>
</head>
<body>
    <form runat="server">
        <p>
            Texte 1 :
            <asp:TextBox id="TextBox1" runat="server" AutoPostBack="True"></asp:TextBox>
        </p>
        <p>
            Texte 2 :
            <asp:TextBox id="TextBox2" runat="server"></asp:TextBox>
        </p>
        <p>
            <asp:Label id="lblInfo1" runat="server"></asp:Label>
        </p>
        <p>
            <asp:Label id="lblInfo2" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

Dans l'onglet [Design], double-cliquons sur le composant [TextBox1]. Le squelette du gestionnaire de l'événement [TextChanged] de cet objet est alors généré (onglet [All]) :

```
<%@ Page Language="VB" %>
<script runat="server">

    Sub TextBox1_TextChanged(sender As Object, e As EventArgs)
    End Sub
</script>
<html>
...
<body>
...
            <asp:TextBox id="TextBox1" runat="server" AutoPostBack="True"
OnTextChanged="TextBox1_TextChanged"></asp:TextBox>
        </p>
....
    </form>
</body>
</html>
```

L'attribut [OnTextChanged="TextBox1_TextChanged"] a été ajouté à la balise <asp:TextBox id="TextBox1"> pour désigner le gestionnaire de l'événement [TextChanged] sur [TextBox1]. C'est la procédure [TextBox1_Changed] que nous écrivons maintenant.

```
    Sub TextBox1_TextChanged(sender As Object, e As EventArgs)
      ' changement de texte
```

```
      lblInfo1.text=Date.now.Tostring("T") + ": evt [TextChanged] sur [TextBox1]. Texte
1=["+textbox1.Text+"]"
    End Sub
```

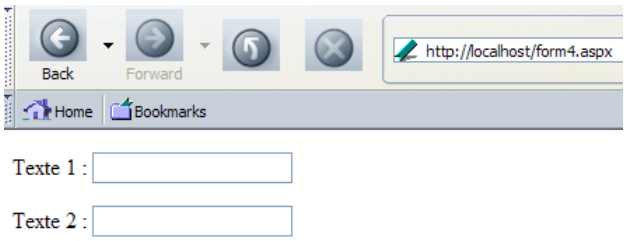
Dans la procédure, nous écrivons dans le label [lblInfo1] un message signalant l'événement et indiquant le contenu de [TextBox1]. On fait de même pour [TextBox2]. Nous indiquons également l'heure pour mieux suivre le traitement des événements. Le code final de [form4.aspx] est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">

    Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
        ' sauve la requte courante dans request.txt du dossier de la page
        Dim requestFileName As String = Me.MapPath(Me.TemplateSourceDirectory) + "\request.txt"
        Me.Request.SaveAs(requestFileName, True)
    End Sub

    Sub TextBox1_TextChanged(sender As Object, e As EventArgs)
      ' changement de texte
      lblInfo1.text=Date.now.Tostring("T") + ": evt [TextChanged] sur [TextBox1]. Texte
1=["+textbox1.Text+"]"
    End Sub

    Sub TextBox2_TextChanged(sender As Object, e As EventArgs)
      ' changement de texte
      lblInfo2.text=Date.now.Tostring("T") + ": evt [TextChanged] sur [TextBox2]. Texte
2=["+textbox2.Text+"]"
    End Sub

</script>
<html>
<head>
    <title>asp:textbox</title>
</head>
<body>
    <form runat="server">
        <p>
            Texte 1 :
            <asp:TextBox id="TextBox1" runat="server" AutoPostBack="True"
OnTextChanged="TextBox1_TextChanged"></asp:TextBox>
        </p>
        <p>
            Texte 2 :
            <asp:TextBox id="TextBox2" runat="server"
OnTextChanged="TextBox2_TextChanged"></asp:TextBox>
        </p>
        <p>
            <asp:Label id="lblInfo1" runat="server"></asp:Label>
        </p>
        <p>
            <asp:Label id="lblInfo2" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

Nous avons ajouté la procédure [Page_Init] pour mémoriser la requête du client comme dans l'exemple précédent.

1.6.2 Tests

Nous lançons l'application sous WebMatrix par [F5]. Nous obtenons la page suivante :



Le code HTML reçu par le navigateur est le suivant :

```
<html>
<head>
    <title>asp:textbox</title>
```

```
</head>
<body>
    <form name="_ctl0" method="post" action="form4.aspx" id="_ctl0">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" value="dDwtMTY4MDc0MTUxOTs7PoqpeSYSCX7lCiWZvw5p7u+/OrTD" />

<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
   var theform = document._ctl0;
   theform.__EVENTTARGET.value = eventTarget;
   theform.__EVENTARGUMENT.value = eventArgument;
   theform.submit();
 }
// -->
</script>

        <p>
            Texte 1 :
                <input name="TextBox1" type="text" id="TextBox1" onchange="__doPostBack('TextBox1','')"
language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1"></span>

        </p>
        <p>
            <span id="lblInfo2"></span>
        </p>
    </form>
</body>
</html>
```

Il y a beaucoup de choses dans ce code qui ont générées automatiquement par le serveur. Notons les points suivants :

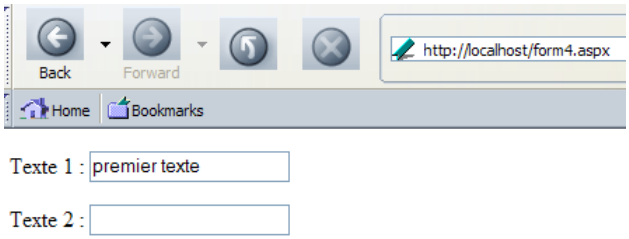
- il y a trois champs cachés : [__VIEWSTATE] que nous avons déjà rencontré, [__EventTarget] et [__EventArgument]. Ces deux derniers champs sont utilisés pour gérer l'événement navigateur "change" sur le champ de saisie [TextBox1]
- les balises serveur <asp:textbox> ont donné naissance aux balises HTML <input type="text" ...> qui correspondent aux champs de saisie
- la balise serveur <asp:textbox id="TextBox1" AutoPostBack="true" ...> a donné naissance à une balise <input type="text" ...> ayant un attribut [onchange="__doPostBack('TextBox1','')"]. Cet attribut dit qu'en cas de changement du contenu de [TextBox1], la fonction Javascript [_doPostBack(...)] doit être exécutée. Celle-ci est la suivante :

```
<script language="javascript">
<!--
 function __doPostBack(eventTarget, eventArgument) {
   var theform = document._ctl0;
   theform.__EVENTTARGET.value = eventTarget;
   theform.__EVENTARGUMENT.value = eventArgument;
   theform.submit();
 }
// -->
</script>
```

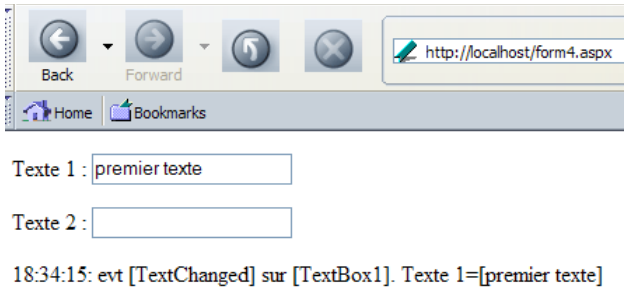
  Que fait la fonction ci-dessus ? Elle affecte une valeur à chacun des deux champs cachés [__EventTarget] et [__EventArgument] puis poste le formulaire. Celui-ci est donc envoyé au serveur. C'est l'effet [AutoPostBack]. L'événement navigateur "change" provoque l'exécution du code "__doPostBack('TextBox1','')". On en déduit que dans le formulaire posté, le champ caché [__EventTarget] aura la valeur 'TextBox1' et le champ caché [__EventArgument] la valeur ''. Cela permettra au serveur de connaître le composant qui a provoqué le POST.
- la balise serveur <asp:textbox id="TextBox2"...> a donné naissance à une balise <input type="text" ...> classique parce que son attribut [AutoPostBack] n'était pas positionné à [true].
- la balise <form> indique que le formulaire sera posté à [form4.aspx] :

```
    <form name="_ctl0" method="post" action="form4.aspx" id="_ctl0">
```

Faisons notre premier test. Tapons un texte dans le premier champ de saisie :



puis mettons le curseur dans le second champ de saisie. Aussitôt, nous obtenons une nouvelle page :



Que s'est-il passé ? Lorsque le curseur a quitté le premier champ de saisie, le navigateur a vérifié si le contenu de celui-ci avait changé. C'était le cas. Il a donc généré, côté navigateur, l'événement [change] sur le champ HTML [TextBox1]. Nous avons vu qu'alors une fonction Javascript s'exécutait et postait le formulaire à [form4.aspx]. Cette page a donc été rechargée par le serveur. Les valeurs postées par le formulaire ont permis au serveur de savoir à son tour que le contenu de la balise serveur [TextBox1] avait changé. La procédure [TextBox1_Changed] a donc été exécutée côté serveur. Elle a placé un message dans le label [lblInfo1]. Une fois cette procédure terminée, [form4.aspx] a été envoyée au navigateur. C'est pourquoi nous avons maintenant un texte dans [lblInfo1]. Ceci dit, on peut s'étonner d'avoir quelque chose dans le champ de saisie [TextBox1]. En effet, aucune procédure exécutée côté serveur n'affecte de valeur à ce champ. On a là un mécanisme général des formulaires web ASP.NET : le serveur renvoie le formulaire dans l'état où il l'a reçu. Pour cela, il réaffecte aux composants la valeur qui a été postée pour eux par le client. Pour certains composants, le client ne poste aucune valeur. C'est le cas par exemples des composants <asp:label> qui sont traduits en balises HTML . On se souvient que le formulaire a un champ caché [__VIEWSTATE] qui représente l'état du formulaire lorsqu'il est envoyé au client. Cet état est la somme des états de tous les composants du formulaire y compris les composants <asp:label> s'il y en a. Comme le champ caché [__VIEWSTATE] est posté par le navigateur client, le serveur est capable de restituer l'état précédent de tous les composants du formulaire. Il ne lui reste plus qu'à modifier ceux qui ont vu leur valeur changée par le POST.

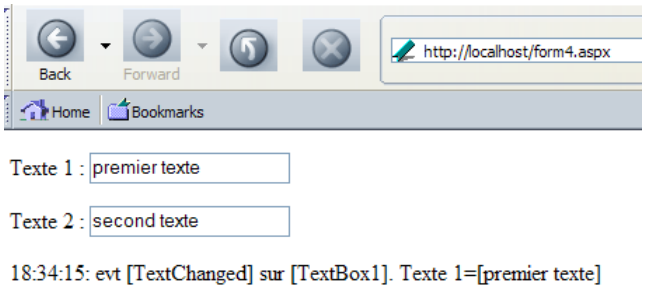
Regardons maintenant dans [request.txt] la requête qui a été faite par le navigateur :

```
POST /form4.aspx HTTP/1.1
```

```
Connection: keep-alive
Keep-Alive: 300
Content-Length: 137
Content-Type: application/x-www-form-urlencoded
Accept: application/x-shockwave-
flash,text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jp
eg,image/gif;q=0.2,*/*;q=0.1
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
Host: localhost
Referer: http://localhost/form4.aspx
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

```
__EVENTTARGET=TextBox1&__EVENTARGUMENT=&__VIEWSTATE=dDwtMTY4MDc0MTUxOTs7PoqpeSYSCX7lCiWZvw5p7u%2B%2FOrTD
&TextBox1=premier+texte&TextBox2=
```

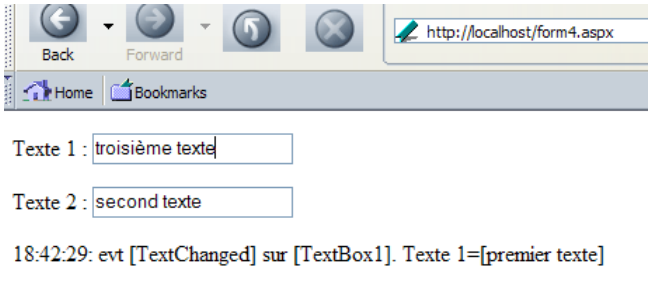
On voit parfaitement le POST ainsi que les paramètres postés. Revenons à notre navigateur et saisissons un texte dans le second champ de saisie :



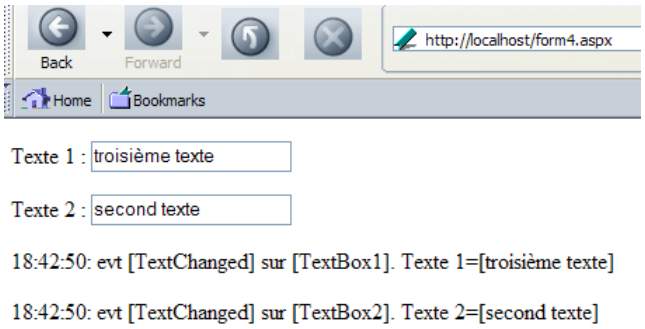
Revenons sur le champ de saisie n° 1 pour saisir un nouveau texte : rien ne se passe cette fois-ci. Pourquoi ? parce que le champ de saisie [TextBox2] n'ayant pas sa propriété [AutoPostBack] à [true], la balise <input type="text"...> générée pour lui ne gère pas l'événement [Change] comme le montre son code HTML :

```
            <input name="TextBox2" type="text" id="TextBox2" />
```

Aucun événement ne se produit donc lorsqu'on quitte le champ de saisie n° 2. Maintenant entrons un nouveau texte dans le champ n° 1 :



Quittons le champ de saisie n° 1. Aussitôt l'événement [Change] sur ce champ est détecté et le formulaire posté au serveur qui renvoie en retour la page suivante :

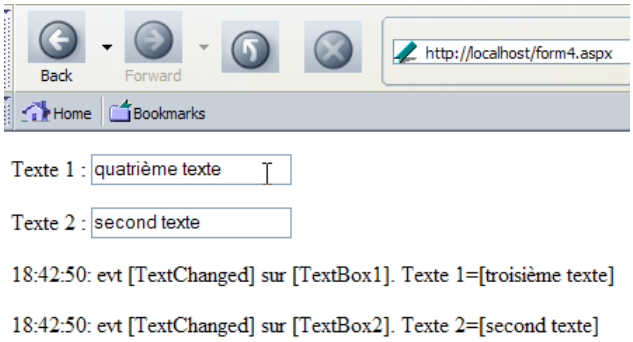


Que s'est-il passé ? Tout d'abord, le navigateur a posté le formulaire. On retrouve ce fait dans la requête du client mémorisée dans [request.txt] :

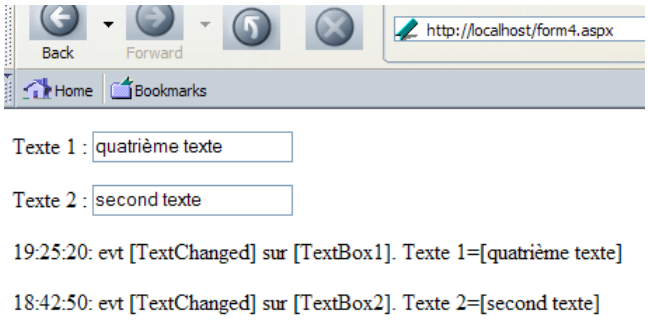
```
POST /form4.aspx HTTP/1.1
....
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316

__EVENTTARGET=TextBox1&__EVENTARGUMENT=&__VIEWSTATE=dDwtMTY4MDc0MTUxOTt0PDtsPGk8MT47PjtsPHQ8O2w8aTwxPjtp
PDU%2BOz47bDx0PHA8cDxsPFRleHQ7PjtsPHByZW1pZXIgdGV4dGU7Pj47Pjs7Pjt0PHA8cDxsPFRleHQ7PjtsPDE4OjQyOjI5OiBldn
QgW1RleHRDaGFuZ2VkXSBzdXIgW1RleHRCb3gxXS4gVGV4dGUgMT1bcHJlbWllciB0ZXh0ZV07Pj47Pjs7Pjs%2BPjs%2BPjs%2BxLOe
rmpUUUz5rTAa%2FFsjda6lVmo%3D&TextBox1=troisi%C3%A8me+texte&TextBox2=second+texte
```

Le serveur commence par restituer aux composants leurs valeurs précédentes grâce au champ caché [__VIEWSTATE] que le client lui a envoyé. Grâce aux champs postés [TextBox1] et [TextBox2] il va affecter aux composants [TextBox1] et [TextBox2] les valeurs qui lui ont été postées. C'est par ce mécanisme que le client va retrouver le formulaire tel qu'il a été posté. Puis, toujours grâce aux champs postés [__VIEWSTATE], [TextBox1] et [TextBox2], le serveur va découvrir que les valeurs des champs de saisie [TextBox1] et [TextBox2] ont changé. Il va donc générer les événements [TextChanged] pour ces deux objets. Le procédures [TextBox1_TextChanged] et [TextBox2_TextChanged] vont être exécutées et les labels [labelInfo1] et [labelInfo2] recevoir une nouvelle valeur. Puis la page [form4.aspx] ainsi modifiée est renvoyée au client.

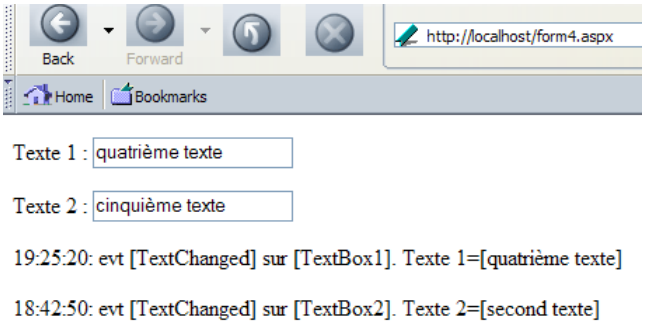
Maintenant nous modifions de nouveau le champ de saisie n° 1 :



Lorsque nous sortons le curseur de saisie du champ 1, l'événement [Change] se produit sur le navigateur. Ensuite la séquence d'événements déjà expliquée (post du navigateur vers le serveur, ..., envoi de la réponse du serveur) se déroule. On obtient la réponse suivante :



A cause de l'heure affichée pour chacun des messages, on voit que seule la procédure [TextBox1_Changed] a été exécutée sur le serveur. La procédure [TextBox2_TextChanged] n'a pas été exécutée parce que justement la valeur de [TextBox2] n'a pas changé. Enfin, mettons un nouveau texte dans le champ n° 2 :



Puis plaçons le curseur sur le champ n° 1 puis remettons-le sur le champ n° 2. La page ne change pas. Pourquoi ? Parce que nous ne changeons pas la valeur du champ n° 1, l'événement navigateur [Change] ne se produit pas lorsque nous quittons ce champ. Du coup, le formulaire n'est pas posté au serveur. Donc rien ne change sur la page. C'est le fait que le contenu de [lblInfo2] ne change pas qui nous montre qu'il n'y a pas de POST. S'il y en avait un, le serveur détecterait que le contenu de [TextBox2] a changé et devrait refléter ce fait dans [lblInfo2].

On retiendra de cet exemple qu'il n'y a pas d'intérêt à mettre la propriété [AutoPostBack] d'un [TextBox] à [true]. Cela provoque un aller-retour client-serveur inutile la plupart du temps.

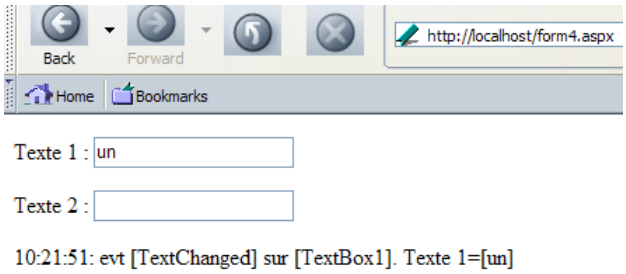
1.6.3 Le rôle du champ __VIEWSTATE

Nous avons vu que le serveur mettait systématiquement un champ caché appelé __VIEWSTATE dans le formulaire qu'il générait. Nous avons dit que ce champ représentait l'état du formulaire et que si on lui retournait ce champ caché, le serveur était capable de reconstituer la valeur précédente du formulaire. L'état d'un formulaire est la somme des états de ses composants. Chacun d'eux a une propriété [EnableViewState] à valeur booléenne indiquant si l'état du composant doit être placé ou non dans le champ caché [__VIEWSATE]. Par défaut cette propriété a la valeur [true] faisant que l'état de tous les composants d'un formulaire est placé dans [__VIEWSTATE]. Quelquefois, ce n'est pas souhaitable.

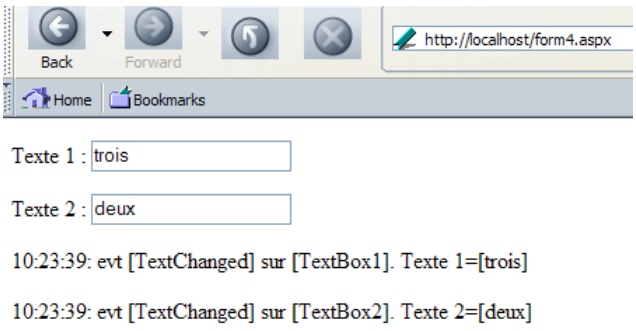
Faisons quelques expériences pour mieux comprendre le rôle de la propriété [EnableViewState]. Mettons cette propriété à [false] pour les deux champs de saisie :

```
...
            <asp:TextBox id="TextBox1" runat="server" OnTextChanged="TextBox1_TextChanged"
AutoPostBack="True" EnableViewState="False"></asp:TextBox>
...
            <asp:TextBox id="TextBox2" runat="server" OnTextChanged="TextBox2_TextChanged"
EnableViewState="False"></asp:TextBox>
...
```

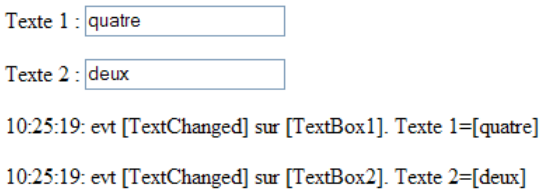
Exécutons maintenant l'application et tapons un premier texte dans le champ n° 1 avant de passer dans le champ n° 2. Un POST est alors fait vers le serveur et on obtient la réponse suivante :



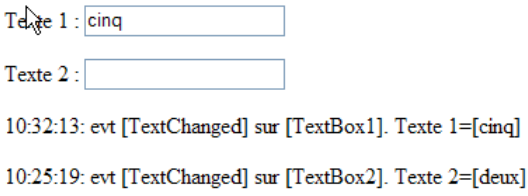
Tapons un texte dans le champ n° 2, modifions celui du champ n° 1 puis revenons dans le champ n° 2 (dans cet ordre). Un nouveau POST est fait vers le serveur et nous obtenons la réponse suivante :



Pour l'instant, tout est comme avant. Maintenant modifions le contenu du champ n° 1 puis passons dans le champ n° 2. Un nouveau POST est fait. La réponse du serveur est la suivante :



Cette fois-ci, il y a un changement. Le serveur a détecté un événement [TextChanged] sur le champ n° 2 puisque l'heure de [lblInfo2] a été modifiée. Or il n'y avait pas de changement. Ceci s'explique par la propriété [EnableViewState=false] de [TextBox2]. Elle fait que le serveur n'a pas mis dans le champ [__VIEWSTATE] du formulaire, l'état précédent de [TextBox2]. Cela revient à lui affecter la chaîne vide comme état précédent. Lorsque le POST dû au changement du contenu de [TextBox1] s'est produit, le serveur a comparé la valeur actuelle de [TextBox2] qui était [deux] à sa valeur précédente (la chaîne vide). Il a en conclu que [TextBox2] avait changé de valeur et a généré l'événement [TextChanged] pour [TextBox2]. On peut valider ce fonctionnement en mettant la chaîne vide dans [TextBox2]. D'après ce qui vient d'être expliqué, le serveur ne devrait pas générer l'événement [TextChanged] pour [TextBox2]. Essayons :



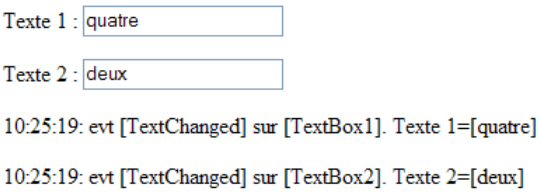
C'est bien ce qui s'est passé. L'heure et le contenu de [lblInfo2] montrent que la procédure [TextBox2_TextChanged] n'a pas été exécutée. Ceci compris, examinons la propriété [EnableViewState] des quatre composants du formulaire :

`TextBox1`	on souhaite maintenir l'état de ce composant afin que le serveur sache s'il a changé ou non
`TextBox2`	idem
`lblInfo1`	on ne souhaite pas maintenir l'état de ce composant. On veut que le texte soit recalculé à chaque nouveau POST. S'il n'est pas recalculé, il doit être vide. Tout cela est correctement obtenu avec [EnableViewState=false]
`lblInfo2`	idem

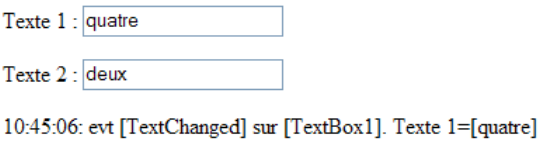
Notre page de présentation devient la suivante :

```
...
            <asp:TextBox id="TextBox1" runat="server" OnTextChanged="TextBox1_TextChanged"
AutoPostBack="True"></asp:TextBox>
...
            <asp:TextBox id="TextBox2" runat="server"
OnTextChanged="TextBox2_TextChanged"></asp:TextBox>
....
            <asp:Label id="lblInfo1" runat="server" enableviewstate="False"></asp:Label>
...
            <asp:Label id="lblInfo2" runat="server" enableviewstate="False"></asp:Label>
...
```

Refaisons la même série de tests que précédemment. Là où nous avons obtenu l'écran



nous obtenons maintenant :



Dans cette étape, on changeait le contenu du champ 1 sans changer celui du champ 2 faisant en sorte que la procédure [TextBox2_TextChanged] n'était pas exécutée côté serveur, ce qui impliquait que le champ [lblInfo2] ne recevait pas de nouvelle valeur. Il est donc affiché avec sa valeur précédente. Dans la version 1 [EnableViewState=true] cette valeur précédente était la valeur saisie lors de la requête précédente. Dans la version 2 [EnableViewState=false], cette valeur précédente est la chaîne vide.

Il est parfois inutile de conserver l'état précédent des composants. Plutôt que de mettre [EnableViewState=false] pour chacun d'eux on peut déclarer que la page ne doit pas maintenir son état. Cela se fait dans la directive [Page] du code de présentation :

```
<%@ Page Language="VB" EnableViewState="False" %>
```

Dans ce cas, quelque soit la valeur de sa propriété [EnableViewState], l'état d'un composant n'est pas mis dans le champ caché [__VIEWSTATE]. Tout se passe alors comme si son état précédent était la chaîne vide.

Utilisons maintenant le client [curl] pour mettre en lumière d'autres mécanismes. Nous demandons tout d'abord l'url [http://localhost/form4.aspx] :

```
dos>curl --include --url http://localhost/form4.aspx
```

```
HTTP/1.1 200 OK
Server: Microsoft ASP.NET Web Matrix Server/0.6.0.0
Date: Sun, 04 Apr 2004 17:51:14 GMT
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 1077
Connection: Close


<html>
<head>
    <title>asp:textbox</title>
</head>
<body>
    <form name="_ctl0" method="post" action="form4.aspx" id="_ctl0">
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<input type="hidden" name="__VIEWSTATE" value="dDwtMTY4MDc0MTUxOTs7PoqpeSYSCX7lCiWZvw5p7u+/OrTD" />

<script language="javascript">
<!--
  function __doPostBack(eventTarget, eventArgument) {
    var theform = document._ctl0;
    theform.__EVENTTARGET.value = eventTarget;
    theform.__EVENTARGUMENT.value = eventArgument;
    theform.submit();
  }
// -->
</script>

        <p>
            Texte 1 :
                  <input name="TextBox1" type="text" id="TextBox1" onchange="__doPostBack('TextBox1','')"
language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1"></span>
        </p>
        <p>
            <span id="lblInfo2"></span>
        </p>
    </form>
</body>
</html>
```

Nous recevons du serveur le code HTML de [form4.aspx]. Il n'est pas différent de celui qu'avait reçu le navigateur. Rappelons ici, la requête faite par le navigateur lorsqu'il postait le formulaire :

```
POST /form4.aspx HTTP/1.1
```

```
Connection: keep-alive
...
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7b) Gecko/20040316
```

```
__EVENTTARGET=TextBox1&__EVENTARGUMENT=&__VIEWSTATE=dDwtMTY4MDc0MTUxOTs7PoqpeSYSCX7lCiWZvw5p7u%2B%2FOrTD
&TextBox1=premier+texte&TextBox2=
```

Faisons ce même POST mais sans envoyer le champ [__VIEWSTATE] :

```
dos>curl --include --url http://localhost/form4.aspx --data __EVENTTARGET=TextBox1 --data __EVENTARGUMENT=
--data TextBox1=premier+texte --data TextBox2=

.................
        <p>
            Texte 1 :
            <input name="TextBox1" type="text" value="premier texte" id="TextBox1"
onchange="__doPostBack('TextBox1','')" language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1">19:57:48: evt [TextChanged] sur [TextBox1]. Texte 1=[premier texte]</span>
        </p>
```

```
        <p>
            <span id="lblInfo2"></span>
        </p>
..............
```

On notera les points suivants :

- le serveur a détecté un événement [TextChanged] sur [TextBox1] puisqu'il a généré le texte [lblInfo1]. L'absence de [__VIEWSTATE] ne l'a pas gêné. En son absence, il suppose que la valeur précédente d'un champ de saisie est la chaîne vide.
- il a été capable de remettre le texte posté pour [TextBox1] dans l'attribut [value] de la balise [TextBox1] afin que le champ [TextBox1] réapparaisse avec la valeur saisie. Pour cela, il n'a pas besoin de [__VIEWSTATE] mais seulement de la valeur postée pour [TextBox1]

Maintenant, refaisons la même requête sans rien changer. Nous obtenons la nouvelle réponse suivante :

```
dos>curl --include --url http://localhost/form4.aspx --data __EVENTTARGET=TextBox1 --data __EVENTARGUMENT=
--data TextBox1=premier+texte --data TextBox2=


        <p>
            Texte 1 :
            <input name="TextBox1" type="text" value="premier texte" id="TextBox1"
onchange="__doPostBack('TextBox1','')" language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1">20:05:47: evt [TextChanged] sur [TextBox1]. Texte 1=[premier texte]</span>
        </p>
        <p>
            <span id="lblInfo2"></span>
        </p>
```

En l'absence de [__VIEWSTATE], le serveur n'a pas pu voir que le champ [TextBox1] n'avait pas changé de valeur. Il fait alors comme si la valeur précédente était la chaîne vide. Il a donc généré ici l'événement [TextChanged] sur [TextBox1]. Rejouons toujours la même requête avec cette fois-ci le champ [TextBox1] vide et [TextBox2] non vide :

```
dos>curl --include --url http://localhost/form4.aspx --data __EVENTTARGET=TextBox1 --data __EVENTARGUMENT=
--data TextBox2=second+texte --data TextBox1=
......
        <p>
            Texte 1 :
            <input name="TextBox1" type="text" id="TextBox1" onchange="__doPostBack('TextBox1','')"
language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" value="second texte" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1"></span>
        </p>
        <p>
            <span id="lblInfo2">20:11:54: evt [TextChanged] sur [TextBox2]. Texte 2=[second texte]</span>
        </p>
......
```

En l'absence de [__VIEWSTATE] la valeur précédente de [TextBox1] a été considérée comme la chaîne vide. La valeur postée de [TextBox1] étant également la chaîne vide, l'événement [TextChanged] sur [TextBox1] n'a pas été généré. La procédure [TextBox1_TextChanged] n'a pas été exécutée et donc le champ [lblInfo1] n'a pas reçu de nouvelle valeur. On sait que dans ce cas, le composant garde son ancienne valeur (si EnableViewState=true). Or ici, ce n'est pas le cas, [lblInfo1] a perdu sa valeur précédente. Ceci, parce que cette valeur est cherchée dans [__VIEWSTATE]. Comme ce champ est absent, la chaîne vide a été affectée à [lblInfo1]. Pour [TextBox2], le serveur a comparé sa valeur postée [second texte] à sa valeur précédente. En l'absence de [__VIEWSTATE], cette valeur précédente est égale à la chaine vide. La valeur postée de [TextBox2] étant différente de la chaîne vide, l'événement [TextChanged] sur [TextBox2] a été généré. La procédure [TextBox2_TextChanged] a été exécutée et le champ [lblInfo2] a reçu une nouvelle valeur.

On peut se demander si les paramètres [__EVENTTARGET] et [__EVENTARGUMENT] sont bien utiles. En n'envoyant pas ces paramètres, le serveur ne saura pas quel événement a provoqué le [submit]. Essayons :

```
dos>curl --include --url http://localhost/form4.aspx --data TextBox2=second+texte --data
TextBox1=premier+texte
```

```
.............................
        <p>
            Texte 1 :
            <input name="TextBox1" type="text" id="TextBox1" onchange="__doPostBack('TextBox1','')"
language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1"></span>
        </p>
        <p>
            <span id="lblInfo2"></span>
        </p>
    </form>
.....................
```

On voit qu'aucun événement [TextChanged] n'a été traité. Par ailleurs, les champs postés [TextBox1] et [TextBox2] ne retrouvent pas leurs valeurs postées. Tout se passe en fait comme si on avait fait un GET. Tout redevient normal si on a le champ [__EVENTTARGET] dans les champs postés, même s'il n'a pas de valeur :

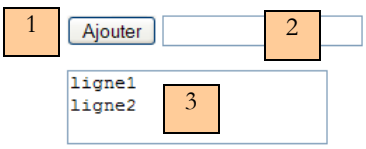
```
dos>curl --include --url http://localhost/form4.aspx --data __EVENTTARGET= --data TextBox2=second+texte --
data TextBox1=premier+texte
.......
        <p>
            Texte 1 :
            <input name="TextBox1" type="text" value="premier texte" id="TextBox1"
onchange="__doPostBack('TextBox1','')" language="javascript" />
        </p>
        <p>
            Texte 2 :
            <input name="TextBox2" type="text" value="second texte" id="TextBox2" />
        </p>
        <p>
            <span id="lblInfo1">20:34:14: evt [TextChanged] sur [TextBox1]. Texte 1=[premier texte]</span>
        </p>
        <p>
            <span id="lblInfo2">20:34:14: evt [TextChanged] sur [TextBox2]. Texte 2=[second texte]</span>
        </p>
........
```

1.6.4 Autres propriétés du composant TextBox

Le composant serveur [TextBox] permet également de générer les balises HTML <input type="password"..> et <texarea>..</textarea>, c.a.d. les balises correspondant respectivement à un champ de saisie protégée et un champ de saisie multilignes. Cette génération est contrôlée par la propriété [TextMode] du composant [TextBox]. Elle a trois valeurs possibles :

valeur	balise HTML générée
`SingleLine`	<input type="text" ...>
`MultiLine`	<textarea>...</textarea>
`Password`	<input type="password ...>

Nous examinons l'usage de ces propriétés avec l'exemple [form5.aspx] suivant :



n°	nom	type	propriétés	rôle
1	`btnAjouter`	Button		bouton [submit] - sert à ajouter le contenu de [TextBox1] à celui de [TextBox2]
2	`TextBox1`	TextBox	TextMode=Password Text=	champ de saisie protégée
3	`TextBox2`	TextBox	TextMode=Multilignes Text=	cumule les saisies faites dans [TextBox1]

La propriété [EnableViewState] de la page est mise à [false]. Côté serveur, nous gérons l'événement clic sur le boutton [btnAjouter] :

```
Sub btnAjouter_Click(sender As Object, e As EventArgs)
  ' on ajoute le contenu de [textBox1] à celui de [TextBox2]
  textbox2.text=textbox2.text + textbox1.text+controlchars.crlf
End Sub
```

Pour comprendre ce code, il faut se rappeler le mode de traitement du POST d'un formulaire. Les procédures [Page_Init] et [Page_Load] sont exécutées en premier. Puis viennent toutes les procédures d'événements mis en cache. Enfin la procédure gérant l'événement qui a provoqué le [POST] est exécutée, ici la procédure [btnAjouter_Click]. Lorsque les gestionnaires d'événements s'exécutent, tous les composants de la page ayant une valeur dans le POST ont pris cette valeur. Les autres ont retrouvé leur valeur précédente si leur propriété [EnableViewState] était à [true] ou leur valeur de conception si leur propriété [EnableViewState] était à [false]. Ici, les valeurs des champs [TextBox1] et [TextBox2] vont faire partie du POST fait par le client. Aussi dans le code précédent, [textbox1.text] aura pour valeur la valeur postée par le client, de même pour [textbox2.text]. La procédure [btnAjouter_Click] met dans le champ [TextBox2] la valeur postée pour [TextBox2] ajoutée à celle postée pour [TextBox1] ajoutée à la marque de fin de ligne [ControlChars.CrLf] définie dans l'espace de noms [Microsoft.VisualBasic]. Il n'est pas nécessaire d'importer cet espace de noms, le serveur web l'important par défaut.

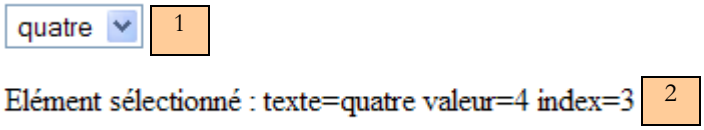
Le code final de [form5.aspx] est le suivant :

```
<%@ Page Language="VB" EnableViewState="False" %>
<script runat="server">

    Sub btnAjouter_Click(sender As Object, e As EventArgs)
      ' on ajoute le contenu de [textBox1] à celui de [TextBox2]
      textbox2.text+=textbox1.text+controlchars.crlf
    End Sub

</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            <asp:Button id="btnAjouter" onclick="btnAjouter_Click" runat="server" Text="Ajouter"
EnableViewState="False"></asp:Button>
            <asp:TextBox id="TextBox1" runat="server" TextMode="Password" Width="353px"
EnableViewState="False"></asp:TextBox>
        </p>
        <p>
            <asp:TextBox id="TextBox2" runat="server" TextMode="MultiLine" Width="419px" Height="121px"
EnableViewState="False"></asp:TextBox>
        </p>
    </form>
</body>
</html>
```

Un peu plus haut, on a donné la copie d'écran d'une exécution.

1.7 Le composant DropDownList

La balise <asp:DropDownList> permet d'insérer une liste déroulante dans le code de présentation d'une page. Nous créons une page [form6.aspx] pour obtenir la présentation suivante :



n°	nom	type	propriétés	rôle
1	DropDownList1	DropDownList	AutoPostback=true EnableViewState=true	liste déroulante
2	lblInfo	Label	EnableViewState=false	message d'information

Le code de présentation généré est le suivant :

```
Page Language="VB" %>
<script runat="server">

</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            <asp:DropDownList id="DropDownList1" runat="server"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged" AutoPostBack="True"></asp:DropDownList>
        </p>
        <p>
            <asp:Label id="lblInfo" runat="server" enableviewstate="False"></asp:Label>
        </p>
    </form>
</body>
</html>
```

Pour l'instant, la liste déroulante ne contient aucun élément. Nous allons la remplir dans la procédure [Page_Load]. Pour cela nous devons connaître certaines des propriétés et méthodes de la classe [DropDownList] :

`Items`	collection de type [ListItemCollection] des éléments de la liste déroulante. Les membres de cette collection dont de type [ListItem].
`Items.Count`	nombre d'éléments de la collection [Items]
`Items(i)`	élement n° i de la liste - de type [ListItem]
`Items.Add`	pour ajouter un nouvel élément [ListItem] à la collection [Items]
`Items.Clear`	pour supprimer tous les éléments de la collection [Items]
`Items.RemoveAt(i)`	pour supprimer l'élément n° i de la collection [Items]
`SelectedItem`	1er élément [ListItem] de la collection [Items] ayant sa propriété [Selected] à vrai
`SelectedIndex`	n° de l'élément [SelectedItem] dans la collection [Items]

Les éléments de la collection [Items] de la classe [DropDownList] sont de type [ListItem]. Chaque élément [ListItem] donne naissance à une balise HTML <option> :

```
<option value="V" [selected="selected"]>T</option>
```

Nous décrivons certaines propriétés et méthodes de la classe [ListItem] :

`ListItem(String texte, String value)`	constructeur - crée un élément [ListItem] avec les propriétés [texte] et [value]. Un élément ListItem(T,V) donnera naissance à la balise HTML <option value="V">T</option>. La classe [ListITem] permet donc de décrire les éléments d'une liste HTML
`Selected`	booléen. Si vrai, l'option correspondante de la liste HTML aura l'attribut [selected="selected"]. Cet attribut indique au navigateur que l'élément correspondant doit appararaître sélectionné dans la liste HTML
`Text`	le texte T de l'option HTML <option value="V" [selected="selected"]>T</option>
`Value`	la valeur V de l'attribut [Value] de l'option HTML <option value="V" [selected="selected"]>T</option>

Nous avons assez d'informations pour écrire dans la procédure [Page_Load] de la page, le code de remplissage de la liste déroulante [DropDownList1] :

```
    Sub Page_Load(sender As Object, e As EventArgs)
    ' on remplit le combo si c'est la 1ère fois qu'on est appelé
    if not IsPostBack then
      dim valeurs() as String = {"1","2","3","4"}
      dim textes() as  String = {"un","deux","trois","quatre"}

      dim i as integer
      for i=0 to valeurs.length-1
        DropDownList1.Items.Add(new ListItem(textes(i),valeurs(i)))
      next
    end if
    end sub
```

Le composant [DropDownList1] ainsi initialisé, sa traduction HTML sera la suivante :

```
<select name="DropDownList1" id="DropDownList1" onchange="__doPostBack('DropDownList1','')"
language="javascript">

 <option value="1">un</option>
 <option value="2">deux</option>
 <option value="3">trois</option>
 <option value="4">quatre</option>

</select>
```

Nous savons que la procédure [Page_Load] est exécutée à chaque exécution de la page [form6.aspx]. Celle-ci est appelée la 1ère fois par un GET, puis par un POST à chaque fois que l'utilisateur sélectionne un nouvel élément dans la liste déroulante. Faut-il exécuter à chaque fois le code de remplissage de cette liste dans [Page_Load] ? La réponse dépend de l'attribut [EnableViewState] du composant [DropDownList1]. Si cet attribut est à vrai, on sait que l'état du composant [DropDownList1] va être conservé au fil des requêtes dans le champ caché [__VIEWSTATE]. Cet état comprend deux choses :

- la liste de toutes les valeurs de la liste déroulante
- la valeur de l'élément sélectionné dans cette liste

Il paraît alors tentant de mettre la propriété [EnableViewState] du composant [DropDownList1] à [true] afin de ne pas avoir à recalculer les valeurs à mettre dans la liste. Le problème alors, est que la procédure [Page_Load] étant exécutée à chaque fois que la page [form6.aspx] est demandée, ces valeurs vont être quand même calculées. L'objet [Page] dont [form6.aspx] est une instance possède un attribut [IsPostBack] à valeur booléenne. Si cet attribut est à vrai, cela signifie que la page est appelée par un POST. A faux, il signifie que la page a été appelée par un GET. Dans notre système d'aller-retour client-serveur, le client demande toujours la même page [form6.aspx] au serveur. La première fois, il la demande avec un GET, les fois suivantes avec un POST. On en conclut que la propriété [IsPostBack] peut nous servir à détecter le premier appel GET du client. On ne génère les valeurs de la liste déroulante que lors de ce premier appel. Aux requêtes suivantes, ces valeurs seront générées par le mécanisme du [VIEWSTATE]. Dans d'autres situations, le contenu d'une liste peut varier d'une requête à l'autre et doit alors être recalculé à chacune d'elles. Dans ce cas, on mettra l'attribut [EnableViewState] de celle-ci à [false] pour éviter un double calcul inutile du contenu de la liste sauf si on a besoin de connaître les éléments sélectionnés précédemment dans la liste car cette information est conservée dans le [VIEWSTATE].

L'attribut [AutoPostBack] de la liste [DropDownList1] a été mis à vrai. Cela signifie que le navigateur postera le formulaire dès qu'il détectera l'événement "changement de l'élément sélectionné" dans la liste déroulante. Le serveur à son tour détectera grâce au [VIEWSTATE] et aux valeurs postées, que l'élément sélectionné dans le composant [DropDownList1] a changé. Il déclenchera alors l'événement [SelectedIndexChanged] sur ce composant. Nous le traiterons avec la procédure suivante :

```
    Sub DropDownList1_SelectedIndexChanged(sender As Object, e As EventArgs)
      ' changement de sélection
      lblInfo.text="Elément sélectionné : texte="+dropdownlist1.selecteditem.text+ _
        " valeur=" + dropdownlist1.selecteditem.value + _
        " index="+ dropdownlist1.selectedindex.tostring
    End Sub
```

Lorsque cette procédure s'exécute, l'objet [DropDownList1] a retrouvé ses éléments de type [ListItem] grâce au [VIEWSTATE]. Par ailleurs, l'un d'eux, de type [ListItem] a son attribut [Selected] à vrai, celui dont la valeur a été postée par le navigateur. On a accès à cet élément de plusieurs façons :

`DropDownList1.SelectedItem`	est le 1er élément [ListItem] de la liste ayant son attribut [Selected] à vrai
`DropDownList1.SelectedItem.Text`	correspond à la partie [texte] de la balise HTML de l'élément <option value="...">texte</option> sélectionné par l'utilisateur
`DropDownList1.SelectedItem.Value`	correspond à la partie [value] de la balise HTML de l'élément <option value="...">texte</option> sélectionné par l'utilisateur
`DropdDownList1.SelectedIndex`	n° dans la collection [DropDownList1.Items] du 1er élément [ListItem] ayant son attribut [Selected] à vrai

Le code final de [form6.aspx] est le suivant :

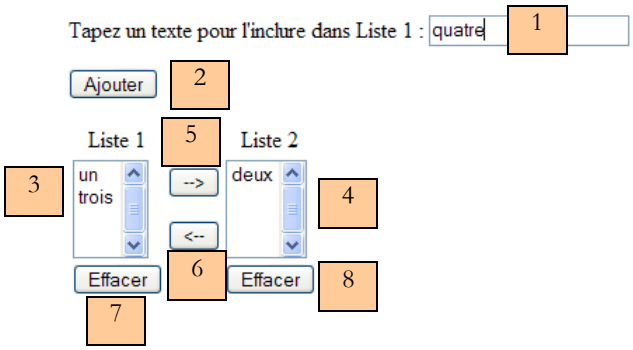
```
<%@ Page Language="VB" %>
<script runat="server">

    Sub Page_Load(sender As Object, e As EventArgs)
    ' on remplit le combo si c'est la 1ère fois qu'on est appelé
    if not IsPostBack then
      dim valeurs() as String = {"1","2","3","4"}
      dim textes() as  String = {"un","deux","trois","quatre"}
```

```
      dim i as integer
      for i=0 to valeurs.length-1
        DropDownList1.Items.Add(new ListItem(textes(i),valeurs(i)))
      next
    end if
    end sub

    Sub DropDownList1_SelectedIndexChanged(sender As Object, e As EventArgs)
      ' changement de sélection
      lblInfo.text="Elément sélectionné : texte="+dropdownlist1.selecteditem.text+ _
        " valeur=" + dropdownlist1.selecteditem.value + _
        " index="+ dropdownlist1.selectedindex.tostring
    End Sub

</script>
```

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            <asp:DropDownList id="DropDownList1" runat="server"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged" AutoPostBack="True"></asp:DropDownList>
        </p>
        <p>
            <asp:Label id="lblInfo" runat="server" enableviewstate="False"></asp:Label>
        </p>
    </form>
</body>
</html>
```

1.8 Le composant ListBox

La balise <asp:ListBox> permet d'insérer une liste dans le code de présentation d'une page. Nous créons [form7.aspx] pour obtenir la présentation suivante :



n°	nom	type	propriétés	rôle
1	txtSaisie	TextBox	EnableViewState=false	champ de saisie
2	btnAjouter	Button		bouton [submit] qui transfère dans Liste 1, le contenu de txtSaisie s'il est non vide.
3	ListBox1	ListBox	EnableViewState=true SelectionMode=Single	liste de valeurs à sélection simple
4	ListBox2	ListBox	EnableViewState=true SelectionMode=Multiple	liste de valeurs à sélection multiple
5	btn1vers2	Button		bouton [submit] qui transfère dans [liste 2] l'élément sélectionné de [liste 1]
6	btn2vers1	Button		bouton [submit] qui transfère dans [liste 1] les éléments sélectionnés de [liste 2]
7	btnRaz1	Button		bouton [submit] qui efface les éléments de [liste 1]

n°	nom	type	propriétés	rôle
8	btnRaz2	Button		bouton [submit] qui efface les éléments de [liste 2]

Le code de présentation généré est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">
</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            Tapez un texte pour l'inclure dans Liste 1 :
            <asp:TextBox id="txtSaisie" runat="server" EnableViewState="False"></asp:TextBox>
        </p>
        <p>
            <asp:Button id="btnAjouter" onclick="btnAjouter_Click" runat="server"
Text="Ajouter"></asp:Button>
        </p>
        <p>
            <table>
                <tbody>
                    <tr>
                        <td>
                            <p align="center">
                                Liste 1
                            </p>
                        </td>
                        <td>
                        </td>
                        <td>
                            <p align="center">
                                Liste 2
                            </p>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <asp:ListBox id="ListBox1" runat="server"></asp:ListBox>
                        </td>
                        <td>
                            <p>
                                <asp:Button id="btn1vers2" onclick="btn1vers2_Click" runat="server"
Text="-->"></asp:Button>
                            </p>
                            <p>
                                <asp:Button id="btn2vers1" onclick="btn2vers1_Click" runat="server"
Text="<--"></asp:Button>
                            </p>
                        </td>
                        <td>
                            <p>
                                <asp:ListBox id="ListBox2" runat="server"
SelectionMode="Multiple"></asp:ListBox>
                            </p>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <p align="center">
                                <asp:Button id="btnRaz1" onclick="btnRaz1_Click" runat="server"
Text="Effacer"></asp:Button>
                            </p>
                        </td>
                        <td>
                        </td>
                        <td>
                            <p align="center">
                                <asp:Button id="btnRaz2" onclick="btnRaz2_Click" runat="server"
Text="Effacer"></asp:Button>
                            </p>
                        </td>
                    </tr>
                </tbody>
            </table>
        </p>
```

```
    </form>
</body>
</html>
```

La classe [ListBox] est dérivée de la même classe [ListControl] que la classe [DropDownList] étudiée précédemment. On y retrouve toutes les propriétés et méthodes vues pour [DropDownList] car elles appartenaient en fait à [ListControl]. Une propriété nouvelle apparaît :

`SelectionMode`	fixe le mode de sélection de la liste HTML <select> qui sera générée à partir du composant. Si [SelectionMode=Single], alors un seul élément pourra être sélectionné. Si SelectionMode=Multiple, plusieurs éléments pourront être sélectionnés. Pour cela, l'attribut [multiple="multiple"] sera généré dans la balise <select> de la liste HTML.

Traitons les événements. Le clic sur le bouton [Ajouter] sera traité par la procédure [btnAjouter_Click] suivante :

```
    Sub btnAjouter_Click(sender As Object, e As EventArgs)
      ' ajout dans la liste 1
      dim texte as string=txtSaisie.text.trim
      if texte<> "" then ListBox1.Items.Add(New ListItem(texte))
      ' raz txtSaisie
      txtSaisie.text=""
    End Sub
```

Si le texte saisi dans [txtSaisie] n'est pas la chaîne vide ou blanche, un nouvel élément est ajouté à la liste [ListBox1]. Nous savons que nous devons ajouter un élément de type [ListItem]. Précédemment, nous avions utilisé le constructeur [ListItem(T as String, V as String)] pour faire un travail analogue. Un tel élément donne naissance à la balise HTML [<option value="V">T</option>]. Ici, nous utilisons le constructeur [ListItem(T as String)] qui donne naissance à la balise HTML [<option value="T">T</option>], c.a.d. que le texte [T] de l'option est pris pour former la valeur de l'option. Une fois le contenu de [txtSaisie] ajouté à la liste [ListBox1], le champ [txTSaisie] est vidé.

Les clics sur les boutons [Effacer] seront traités par les procédures suivantes :

```
    Sub btnRaz1_Click(sender As Object, e As EventArgs)
      ' raz liste 1
      ListBox1.Items.Clear
    End Sub

    Sub btnRaz2_Click(sender As Object, e As EventArgs)
      ' raz liste 2
      ListBox2.Items.Clear
    End Sub
```

Les clics sur les boutons de transfert entre listes sont eux traités par les procédures suivantes :

```
    Sub btn1vers2_Click(sender As Object, e As EventArgs)
      ' transfert de l'élment sélectionné dans liste 1 vers liste 2
      transfert(ListBox1,ListBox2)
    End Sub

    Sub btn2vers1_Click(sender As Object, e As EventArgs)
      ' transfert de l'élment sélectionné dans liste 2 vers liste 1
      transfert(ListBox2,ListBox1)
    End Sub

    sub transfert(l1 as listbox, l2 as listbox)
      ' transferts des éléments sélectionnés dans l1 vers l2
      ' qq chose à faire ?
      if l1.selectedindex=-1 then return
      dim i as integer
      ' on commence par la fin
      for i=l1.items.count-1 to 0 step -1
          ' sélectionné ?
          if l1.items(i).selected then
          ' plus sélectionné
          l1.items(i).selected=false
          ' transfert dans l2
          l2.items.add(l1.items(i))
          ' suppression dans l1
          l1.items.removeAt(i)
        end if
      next
    end sub
```

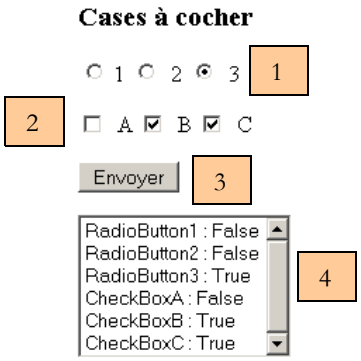
Parce que les deux boutons font le même travail, transférer des éléments d'une liste à une autre, on peut se ramener à une unique procédure de transfert ayant deux paramètres :

- l1 de type [ListBox] qui est la liste source
- l2 de type [ListBox] qui est la liste destination

Tout d'abord, on vérifie qu'il y a bien au moins un élément sélectionné dans la liste l1, sinon il n'y a rien à faire. Pour cela, on examine la propriété [l1.selectedindex] qui représente le n° du premier élément sélectionné dans la liste. S'il n'y en a aucun, sa valeur est -1. S'il y a au moins un élément sélectionné dans l1, on opère le transfert vers l2. Pour cela, on parcourt toute la liste des éléments de l1 et on regarde pour chacun d'eux si son attribut [selected] est à vrai. Si oui, son attribut [selected] est mis à [false], puis il est recopié dans la liste l2 puis supprimé de la liste l1. Cette suppression provoque une renumérotation des éléments de la liste l1. C'est pourquoi la liste des éléments de l1 est-elle parcourue à l'envers. Si on la parcourt à l'endroit et qu'on supprime l'élément n° 10, l'élément n° 11 devient le n° 10 et le n° 12 l'élément n° 11. Après avoir traité l'élément n° 10, notre boucle à l'endroit va traiter l'élément n° 11 qui d'après ce qu'on vient d'expliquer est l'ancien n° 12. Celui qui avait le n° 11 et qui porte maintenant le n° 10 nous échappe. En parcourant les éléments de la liste l1 dans l'autre sens, on évite ce problème.

1.9 Les composants CheckBox, RadioButton

Les balises <asp:RadioButton> et <asp:CheckBox> permettent d'insérer respectivement un bouton radio et une case à cocher dans le code de présentation d'une page. Nous créons une page [form8.aspx] pour obtenir la présentation suivante :



n°	nom	type	propriétés	rôle
1	RadioButton1 RadioButton2 RadioButton3	RadioButton	RadioButton1.Checked=true RadioButton1.Text=1 RadioButton2.Checked=false RadioButton2.Text=2 RadioButton3.Checked=false RadioButton3.Text=3 pour les 3 boutons : GroupName=radio	boutons radio
2	CheckBoxA CheckBoxB CheckBoxC	CheckBox	Checked=false pour tous CheckBoxA.Text=A CheckBoxB.Text=B CheckBoxC.Text=C	cases à cocher
3	btnEnvoyer	Button		bouton [submit]
4	lstInfos	ListBox		liste d'informations

Pour que le navigateur traite les trois boutons radio comme des boutons exclusifs les uns des autres, il faut les rassembler dans un groupe de boutons radio. Cela se fait au moyen de l'attribut [GroupName] de la classe [RadioButton]. L'état de la page n'a pas à être maintenu dans cette application. Aussi mettons-nous l'attribut [EnableViewState="false"] à la page. Le code de présentation est le suivant :

```
<%@ Page Language="VB" EnableViewState= "false "%>
```

```
<html>
<head>
</head>
<body>
    <form id="frmControls" runat="server">
        <h3>Cases à cocher
        </h3>
        <p>
            <asp:RadioButton id="RadioButton1" runat="server" Checked="True" GroupName="radio"
Text="1"></asp:RadioButton>
            <asp:RadioButton id="RadioButton2" runat="server" GroupName="radio"
Text="2"></asp:RadioButton>
```

```
             <asp:RadioButton id="RadioButton3" runat="server" GroupName="radio"
Text="3"></asp:RadioButton>
        </p>
        <p>
            <asp:CheckBox id="CheckBoxA" runat="server" Text="A"></asp:CheckBox>
            <asp:CheckBox id="CheckBoxB" runat="server" Text="B"></asp:CheckBox>
            <asp:CheckBox id="CheckBoxC" runat="server" Text="C"></asp:CheckBox>
        </p>
        <p>
            <asp:Button id="btnEnvoyer" onclick="btnEnvoyer_Click" runat="server"
Text="Envoyer"></asp:Button>
            <asp:Button id="btnTree" onclick="btnTree_Click" runat="server"
Text="Contrôles"></asp:Button>
        </p>
        <p>
            <asp:ListBox id="lstInfos" runat="server" Rows="6" Height="131px"></asp:ListBox>
        </p>
    </form>
</body>
</html>
```

Il nous faut écrire la procédure [btnEnvoyer_Click] pour traiter l'événement [Click] sur ce bouton. L'état d'un bouton radio ou d'une case à cocher est donné par son attribut [Checked], booléen à vrai si la case est cochée, à faux sinon. Il nous suffit donc d'écrire dans la liste [lstInfos] la valeur de l'attribut [Checked] des six boutons radio et cases à cocher. Comme il n'y a pas de difficulté particulière à cela, nous innovons un peu :

```
<script runat="server">

    Sub btnEnvoyer_Click(sender As Object, e As EventArgs)
      ' on place des informations dans le listbox
      for each c as control in FindControl("frmControls").controls
          ' le contrôle est-il dérivé de CheckBox ?
          if TypeOf(c) is CheckBox then
              lstInfos.Items.Add(c.ID + " : " + Ctype(c,CheckBox).Checked.ToString)
          end if
      next
    End Sub

</script>
```

La page peut être vue comme une structure arborescente de contrôles. Dans notre exemple, notre page comporte des textes et des contrôles serveur. Les textes sont vus comme un contrôle particulier appelé [LiteralControl]. Le moindre texte donne naissance à ce contrôle même une suite d'espaces entre deux contrôles. Chaque contrôle a un attribut ID qui l'identifie. C'est l'attribut ID qui appararaît dans les balises :

```
            <asp:CheckBox id="CheckBoxA" runat="server" ></asp:CheckBox>
```

Si nous ignorons les contrôles [LiteralControl], la page étudiée a les contrôles suivants :

- [HtmlForm] qui est le formulaire [ID=frmControls]. Celui-ci est à son tour un conteneur de contrôles. Il contient les contrôles suivants :
-- [ID=RadioButton1] de type [RadioButton]
-- [ID=RadioButton2] de type [RadioButton]
-- [ID=RadioButton3] de type [RadioButton]
-- [ID=CheckBoxA] de type [CheckBox]
-- [ID= CheckBoxA] de type [CheckBox]
-- [ID= CheckBoxA] de type [CheckBox]
-- [ID=btnEnvoyer] de type [Button]

Un contrôle possède les propriété suivantes :

`[Control].Controls`	rend la collection des contrôles enfants de [Control] s'il en a
`[Control].FindControl (ID)`	rend le contrôle identifié par ID se trouvant à la racine de l'arborescence des contrôles enfants de [Control]. Dans l'exemple ci-dessus : Page.FindControl("frmControls") désigne le conteneur [HtmlForm]. Pour atteindre le bouton radio [RadioButton1], il faudra écrire Page.FindControl("frmControls").FindControl("RadioButton1")
`[Control].ID`	identifiant de [Control]

Revenons au code de la procédure [btnEnvoyer_Click] :

```
    Sub btnEnvoyer_Click(sender As Object, e As EventArgs)
      ' on place des informations dans le listbox
      for each c as control in FindControl("frmControls").controls
          ' le contrôle est-il dérivé de CheckBox ?
          if TypeOf(c) is CheckBox then
              lstInfos.Items.Add(c.ID + " : " + Ctype(c,CheckBox).Checked.ToString)
          end if
      next
    End Sub
```

On veut afficher l'état des boutons radio et des cases à cocher qui se trouvent dans le formulaire. Nous parcourons tous les contrôles de celui-ci. Si le contrôle courant est d'un type dérivé de [CheckBox] nous affichons sa propriété [Checked]. La classe [RadioButton] étant dérivée de la classe [CheckBox], le test est valable pour les deux types de contrôles. La copie d'écran présentée plus haut montre un exemple d'exécution.

1.10 Les composants CheckBoxList, RadioButtonList

Parfois, on souhaite amener l'utilisateur à faire des choix entre des valeurs qu'on ne connaît pas au moment de la conception de la page. Ces choix proviennent d'un fichier de configuration, d'une base de données, ... et ne sont connus qu'au moment de l'exécution. Il existe des solutions à ce problème et nous les avons rencontrées. La liste à sélection simple convient bien lorsque l'utilisateur ne peut faire qu'un choix et la liste à sélection multiple lorsqu'il peut en faire plusieurs. D'un point de vue esthétique et si le nombre de choix n'est pas important, on peut vouloir utiliser des boutons radio à la place de la liste à sélection simple ou des cases à cocher à la place de la liste à sélection multiple. C'est possible avec les composants [CheckBoxList] et [RadioButtonList].

Les classes [CheckBoxList] et [RadioButtonList] sont dérivées de la même classe [ListControl] que les classes [DropDownList] et [ListBox] étudiées précédemment. Aussi va-t-on retrouver certaines des propriétés et méthodes vues pour ces classes, celles qui appartenaient en fait à [ListControl] :

`Items`	collection de type [ListItemCollection] des éléments de la liste déroulante. Les membres de cette collection dont de type [ListItem].
`Items.Count`	nombre d'éléments de la collection [Items]
`Items(i)`	élement n° i de la liste - de type [ListItem]
`Items.Add`	pour ajouter un nouvel élément [ListItem] à la collection [Items]
`Items.Clear`	pour supprimer tous les éléments de la collection [Items]
`Items.RemoveAt(i)`	pour supprimer l'élément n° i de la collection [Items]
`SelectedItem`	1er élément [ListItem] de la collection [Items] ayant sa propriété [Selected] à vrai
`SelectedIndex`	n° de l'élément précédent dans la collection [Items]

Certaines propriétés sont spécifiques aux classes [CheckBoxList] et [RadioButtonList] :

`RepeatDirection`	[horizontal] ou [vertical] pour des listes horizontales ou verticales.

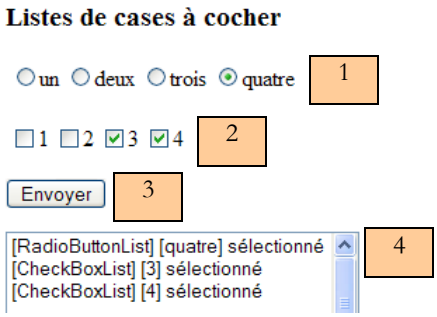
Les éléments de la collection [Items] sont de type [ListItem]. Chaque élément [ListItem] donnera naissance à une balise différente selon qu'on a affaire à un objet [CheckBoxList] ou [RadioButtonList] :

```
<input type="checkbox" [selected="selected"] value="V">Texte
```
ou
```
<input type="radio" [selected="selected"] value="V">Texte
```

Nous redonnons certaines propriétés et méthodes de la classe [ListItem] :

`ListItem(String texte, String value)`	constructeur - crée un élément [ListItem] avec les propriétés texte et value. Un élément ListItem(T,V) donnera naissance à la balise HTML <input type="checkbox" value="V">T ou <input type="radio" value="V">T selon les cas.
`Selected`	booléen. Si vrai, l'option correspondante de la liste HTML aura l'attribut [selected="selected"]. Cet attribut indique au navigateur que l'élément correspondant doit appararaître sélectionné dans la liste HTML
`Text`	le texte T de l'option HTML <input type=".." value="V" [selected="selected"]>T
`Value`	la valeur de l'attribut Value de l'option HTML <input type=".." value="V" [selected="selected"]>T

Nous nous proposons de construire la page [form8b.aspx] suivante :



n°	nom	type	propriétés	rôle
1	RadioButtonList1	RadioButtonList	EnableViewState=true RepeatDirection=horizontal	liste de boutons radio
2	CheckBoxList1	CheckBoxList	EnableViewState=true RepeatDirection=horizontal	liste de cases à cocher
3	btnEnvoyer	Button		bouton [submit] qui affiche dans [4] la liste des éléments sélectionnés dans les deux listes
4	lstInfos	ListBox	EnableViewState=false	liste de valeurs

Le code de présentation de la page est le suivant :

```
<%@ Page Language="VB" autoeventwireup="false" %>
<script runat="server">
...
</script>
<html>
<head>
</head>
<body>
    <form id="frmControls" runat="server">
        <h3>Listes de cases à cocher
        </h3>
        <p>
            <asp:RadioButtonList id="RadioButtonList1" runat="server"
RepeatDirection="Horizontal"></asp:RadioButtonList>
        </p>
        <p>
            <asp:CheckBoxList id="CheckBoxList1" runat="server"
RepeatDirection="Horizontal"></asp:CheckBoxList>
        </p>
        <p>
            <asp:Button id="btnEnvoyer" onclick="btnEnvoyer_Click" runat="server"
Text="Envoyer"></asp:Button>
        </p>
        <p>
            <asp:ListBox id="lstInfos" runat="server" EnableViewState="False" Rows="6"></asp:ListBox>
        </p>
    </form>
</body>
</html>
```

Le code de contrôle est le suivant :

```
<script runat="server">

    Sub Page_Load(sender As Object, e As EventArgs) handles MyBase.Load
        ' on remplit les listes si c'est la 1ère fois qu'on est appelé
        if not IsPostBack then
          ' textes pour la liste RadioButton
          dim textesRadio() as String = {"1","2","3","4"}
          ' textes pour la liste CheckBox
          dim textesCheckBox() as  String = {"un","deux","trois","quatre"}
        ' remplissage liste radio
           dim i as integer
           for i=0 to textesRadio.length-1
```

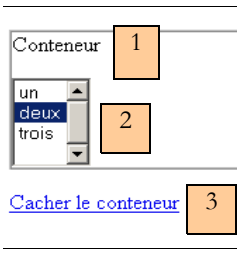
```
                RadioButtonList1.Items.Add(new ListItem(textesRadio(i)))
            next
            ' sélection élément n° 1
            RadioButtonList1.SelectedIndex=1
            ' remplissage liste checkbox
            for i=0 to textesCheckBox.length-1
                CheckBoxList1.Items.Add(new ListItem(textesCheckBox(i)))
            next
        end if
    end sub


    Sub btnEnvoyer_Click(sender As Object, e As EventArgs)
      ' on place des informations dans le listbox lstinfos
      affiche(RadioButtonList1)
      affiche(CheckBoxList1)
    End Sub

    sub affiche(l1 as ListControl)
      ' affiche les valeurs des éléments sélectionnés de l1
      ' qq chose à faire ?
      if l1.selectedindex=-1 then return
      dim i as integer
      ' on commence par la fin
      for i= 0 to l1.items.count-1
        ' sélectionné ?
        if l1.items(i).selected then
          lstInfos.Items.Add("["+TypeName(l1)+"] ["+l1.items(i).text+"] sélectionné")
        end if
      next
    end sub

</script>
```

Dans la procédure [Page_Load] qui est exécutée à chaque exécution de la page, les deux listes sont initialisées. Pour éviter qu'elles le soient à chaque fois, on utilise la propriété [IsPostBack] pour ne le faire que la première fois. Les fois suivantes, les listes seront régénérées automatiquement par le mécanisme du [VIEWSTATE]. Une fois la page affichée, l'utilisateur coche certaines cases et utilise le bouton [Envoyer]. Les valeurs du formulaire sont alors postées au formulaire lui-même. Après exécution de [Page_Load], c'est la procédure [btnEnvoyer_Click] qui est exécutée. Celle-ci fait appel à la procédure [affiche] pour remplir la liste [lstInfos]. Celle-ci reçoit comme paramètre un objet de type [ListControl], ce qui permet de lui envoyer indifféremment un objet [RadioButtonList] ou un objet [CheckBoxList], classes dérivées de [ListControl]. La liste [lstInfos] peut avoir son attribut [EnableViewState] à [false] puisque son état n'a pas à être maintenu entre les différentes requêtes.

1.11 Les composants Panel, LinkButton

La balise <asp:panel> permet d'insérer un conteneur de contrôles dans une page. L'intérêt du conteneur est que certaines de ses propriétés s'appliquent à tous les contrôles qu'il contient. Il en est ainsi de sa propriété [Visible]. Cette propriété existe pour tout contrôle serveur. Si un conteneur a la propriété [Visible=true], chacun de ses contrôles sera contrôlé par sa propre propriété [Visible]. S'il a la propriété [Visible=false], alors le conteneur et tout ce qu'il contient n'est pas affiché. Cela peut être plus simple que de gérer la propriété [Visible] de chacun des contrôles du conteneur.

La balise <asp:LinkButton> permet d'insérer un lien dans le code de présentation d'une page. Elle a un rôle analogue au bouton [Button]. Elle provoque en effet un POST côté client grâce à une fonction Javascript qui lui est associée. Nous créons une page [form9.aspx] pour obtenir la présentation suivante :



n°	nom	type	propriétés	rôle
1	Panel1	Panel	EnableViewState=true	conteneur de contrôles
2	ListBox1	ListBox	EnableViewState=true	une liste de trois valeurs
3	lnkCacher	LinkButton	EnableViewState=false	lien pour cacher le conteneur

Lorsque le conteneur est caché, un nouveau lien apparaît :

n°	nom	type	propriétés	rôle
4	lnkVoir	LinkButton	EnableViewState=false	lien pour afficher le conteneur

Le code de présentation de la page est le suivant :

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            <asp:Panel id="Panel1" runat="server" BorderStyle="Ridge" BorderWidth="1px">
                <p>
                    Conteneur
                </p>
                <p>
                    <asp:ListBox id="ListBox1" runat="server">
                        <asp:ListItem Value="1">un</asp:ListItem>
                        <asp:ListItem Value="2">deux</asp:ListItem>
                        <asp:ListItem Value="3" Selected="True">trois</asp:ListItem>
                    </asp:ListBox>
                </p>
            </asp:Panel>
        </p>
        <p>
            <asp:LinkButton id="lnkVoir" onclick="lnkVoir_Click" runat="server">Voir le
conteneur</asp:LinkButton>
        </p>
        <p>
            <asp:LinkButton id="lnkCacher" onclick="lnkCacher_Click" runat="server">Cacher le
conteneur</asp:LinkButton>
        </p>
    </form>
</body>
</html>
```

Remarquons que ce code initialise la liste [ListBox1] avec trois valeurs. Les gestionnaires des événements [Clic] sur les deux liens sont les suivants :

```
<%@ Page Language="VB" %>
<script runat="server">

    Sub Page_Load(sender As Object, e As EventArgs)
...
    end sub

    Sub lnkVoir_Click(sender As Object, e As EventArgs)
        ' affiche le conteneur 1
        panel1.Visible=true
        ' changt des liens
        lnkVoir.visible=false
        lnkCacher.visible=true
    End Sub

    Sub lnkCacher_Click(sender As Object, e As EventArgs)
        ' cache le conteneur 1
        panel1.Visible=false
        ' changt des liens
        lnkVoir.visible=true
        lnkCacher.visible=false
    End Sub
</script>
```

Nous utiliserons la procédure [Page_Load] pour initialiser le formulaire. Nous le ferons lors de la première requête (IsPostBack=false) :

```
<%@ Page Language="VB" %>
<script runat="server">

    Sub Page_Load(sender As Object, e As EventArgs)
        ' la 1ère fois
```

```
        if not IsPostBack then
            ' on montre le conteneur
            lnkVoir_Click(nothing,nothing)
        end if
    end sub
.....
</script>
```

1.12 Pour poursuivre...

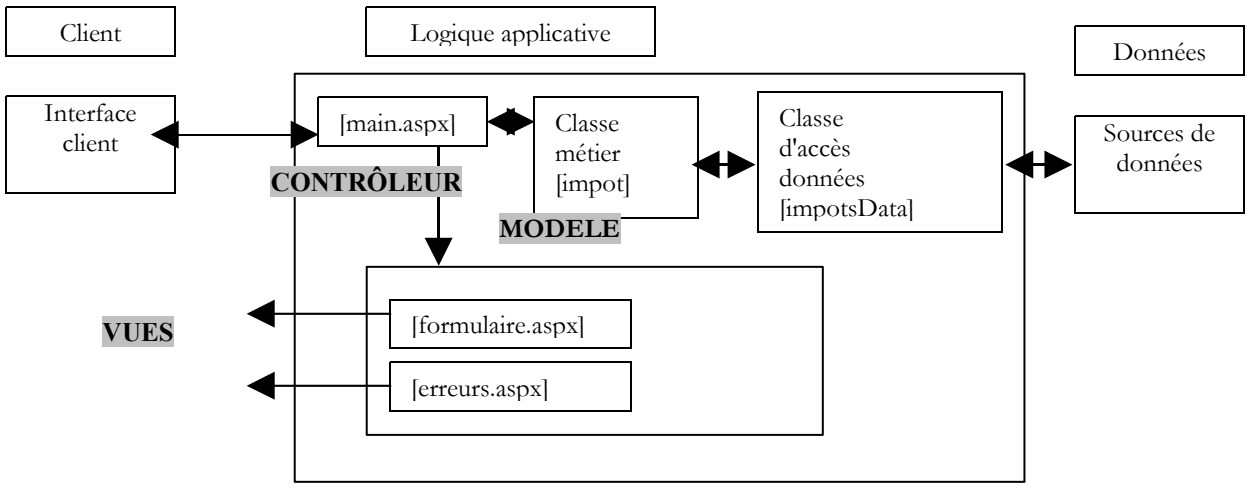
Les paragraphes précédents ont présenté un certain nombre de composants serveur. Seules quelques propriétés de ceux-ci ont été présentées à chaque fois. Pour approfondir l'étude de ces composants, le lecteur pourra procéder de diverses façons :

- découvrir les propriétés d'un composant avec un IDE tel que WebMatrix. Celui-ci expose en effet les principales propriétés des composants utilisés dans un formulaire.
- consulter la documentation de .NET pour découvrir l'intégralité des classes correspondant à chacun des composants serveur. C'est la méthode à préférer pour une maîtrise totale du composant. On y découvrira l'arborescence des classes dont dérivent les composants avec les propriétés, méthodes, constructeurs, événements de chacune d'elles. De plus la documentation fournit parfois des exemples.

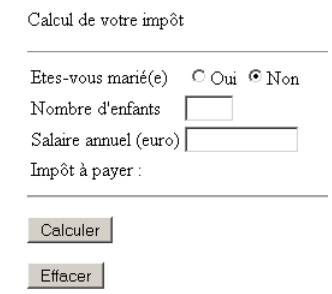
Dans ce chapitre, nous avons utilisé la technique [WebMatrix] du tout en un, c.a.d. qu'on a mis le code de présentation et le code de contrôle d'une page dans le même fichier. De façon générale, nous ne préconisons pas cette méthode mais de préférence celle dite du [codebehind] utilisée dans les parties 1 et 2 de ce cours, qui met ces deux codes dans deux fichiers séparés. On rappelle que l'intérêt de cette séparation réside dans le fait que le code de contrôle peut être compilé sans avoir à exécuter l'application web. Par ailleurs, nos exemples, et nous l'avons expliqué dès le début du chapitre, avaient un profil bien particulier : ils étaient constitués d'une unique page qui était un formulaire que s'échangeaient le client et le serveur dans des cycles successifs de demande-réponse, la première requête client étant un GET, et les suivantes des POST.

1.13 Composants serveur et contrôleur d'application

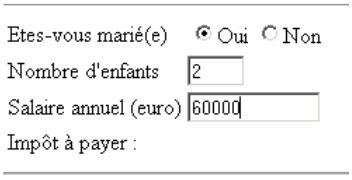
Dans les parties 1 et 2 de ce cours, nous avons construit plusieurs applications web. Elles ont toutes été construites selon l'architecture MVC (Modèle-Vue-Contrôleur) qui découpe l'aplication en blocs bien distincts et en facilite la maintenance. Nous construisions alors nos interfaces utilisateur avec des balises HTML standard. Avec ce qui vient d'être vu, il est naturel de vouloir maintenant utiliser des composants serveur. Reprenons un problème déjà étudié longuement, qui était celui du calcul d'un impôt. Son architecture MVC était la suivante :



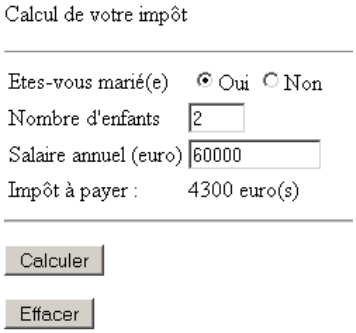
L'application a deux vues : [formulaire.aspx] et [erreurs.aspx]. La vue [formulaire.aspx] est présentée lorsque l'url [main.aspx] est demandée la première fois :



L'utilisateur renseigne le formulaire :



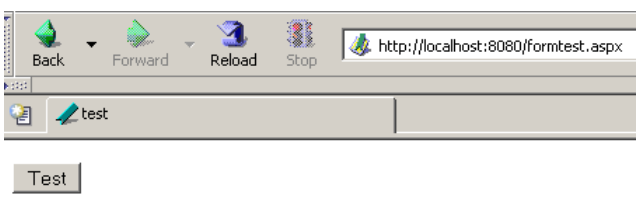
et utilise le bouton [Calculer] pour obtenir la réponse suivante :



Dans une application MVC, toute requête doit passer par le contrôleur, ici [main.aspx]. Cela signifie que lorsque le formulaire [formulaire.aspx] a été rempli par l'utilisateur, il doit être posté à [main.aspx] et non pas à [formulaire.aspx]. Ceci n'est tout simplement pas possible si nous construisons l'interface utilisateur [formulaire.aspx] avec des composants serveurs ASP. Pour s'en rendre compte, construisons un formulaire [formtest.aspx] avec un composant <asp:button> :

```
<%@ Page Language="VB" EnableViewState="false"%>
<html>
<head>
 <title>test</title>
</head>
<body>
    <form action="main.aspx" runat="server">
        <p>
            <asp:Button id="btnTest" runat="server" EnableViewState="false" Text="Test"></asp:Button>
        </p>
    </form>
</body>
</html>
```

On notera l'attribut [action="main.aspx"] de la balise <form..>. Exécutons cette application. La page de présentation ne présente qu'un bouton :



Regardons le code HTML envoyé par le serveur :

```
<html>
<head>
 <title>test</title>
</head>
<body>
    <form name="_ctl0" method="post" action="formtest.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwtNTMwNzcxMzI0Ozs+" />

        <p>
            <input type="submit" name="btnTest" value="Test" id="btnTest" />
        </p>

    </form>
</body>
</html>
```

On voit que le POST du formulaire a pour cible le formulaire lui-même [action="formtest.aspx"] alors que nous avions écrit dans [formtest.aspx] la balise HTML serveur :

```
    <form action="main.aspx" runat="server">
```

L'attribut [runat="server"] de la balise <form> nous est imposé par l'utilisation des composants serveur. Une erreur de compilation se produit si nous ne mettons pas cet attribut. Lorsqu'on le met, alors l'attribut [action] de la balise <form> est **ignoré**. Le serveur génère toujours un attribut [action] qui pointe sur le formulaire lui-même. On en déduit que dans une application MVC, on ne peut utiliser de formulaires construits avec la balise <form ... runat="server">. Or cette balise est indispensable à tous les composants ASP serveur récupérant des saisies de l'utilisateur. Autant dire qu'on ne peut utiliser de formulaires ASP serveur dans une application MVC. C'est une grande découverte. En effet, l'un des points forts du marketing d'ASP.NET est qu'on peut construire une application web comme une application windows. C'est vrai si notre application ne respecte pas l'architecture MVC mais faux sinon. Or l'architecture MVC paraît une notion fondamentale du développement web actuel qu'il semble difficile d'ignorer.

Il est cependant possible d'utiliser l'architecture MVC conjointement avec des formulaires à composants ASP pour des applications ayant peu de vues différentes, grâce à l'artifice suivant :

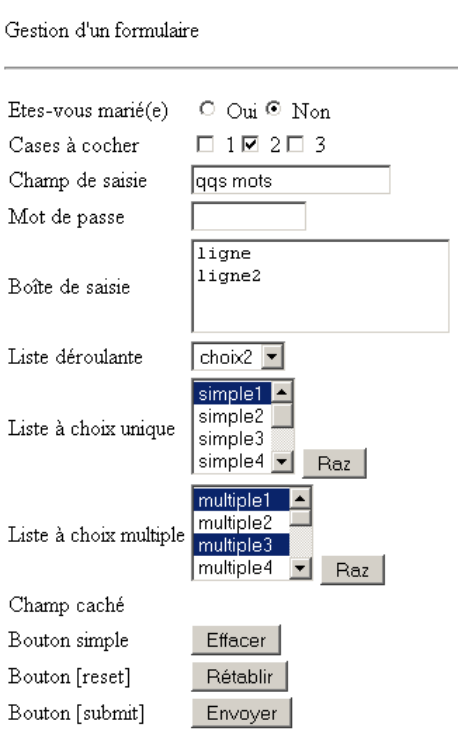
- l'application consiste en une page unique qui fait office de contrôleur
- les vues se traduisent dans cette page par des conteneurs différents, un conteneur par vue. Pour afficher une vue, on rend visible son conteneur et on cache les autres

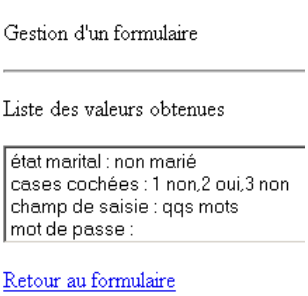
C'est une solution élégante que nous mettons en oeuvre maintenant dans quelques exemples

1.14 Exemples d'applications MVC avec composants serveur ASP

1.14.1 Exemple 1

Nous mettons en oeuvre dans ce premier exemple les composants serveur que nous avons présentés. La page [form10.aspx] sera la suivante :





La copie d'écran gauche ci-dessus est le formulaire tel qu'il est présenté au client. Celui-ci le remplit et le valide par [Envoyer]. Le serveur lui renvoie une vue lui présentant une liste des valeurs saisies (copie d'écran droite). Un lien permet à l'utilisateur de retourner au formulaire. Il retrouve celui-ci tel qu'il l'a validé. Le code de présentation de [form10.aspx] est le suivant :

```
<html>
<head>
    <title>Exemple</title> <script language="javascript">
   function effacer(){
     alert("Vous avez cliqué sur [Effacer]")
   }
 </script>
</head>
<body>
    <p>
        Gestion d'un formulaire
    </p>
    <p>
        <hr />
    </p>
    <form runat="server">
        <p>
```

```
            <asp:Panel id="panelinfo" runat="server" EnableViewState="False">
                <p>
                    Liste des valeurs obtenues
                </p>
                <p>
                    <asp:ListBox id="lstInfos" runat="server" EnableViewState="False"></asp:ListBox>
                </p>
                <p>
                    <asp:LinkButton id="LinkButton1" onclick="LinkButton1_Click" runat="server">Retour
au formulaire</asp:LinkButton>
                </p>
                <p>
                    <hr />
                </p>
            </asp:Panel>
```

```
        </p>
        <p>
```

```
           <asp:Panel id="panelform" runat="server" >
               <table>
                   <tbody>
                       <tr>
                           <td>
                               Etes-vous marié(e)</td>
                           <td>
```

```
                                <asp:RadioButton id="rdOui" runat="server"
GroupName="rdmarie"></asp:RadioButton>
                                Oui<asp:RadioButton id="rdNon" runat="server"  GroupName="rdmarie"
Checked="True"></asp:RadioButton>
                                Non</td>
                        </tr>
                        <tr>
                            <td>
                                Cases à cocher</td>
                            <td>
                                <asp:CheckBox id="chk1" runat="server"></asp:CheckBox>
                                1<asp:CheckBox id="chk2" runat="server"></asp:CheckBox>
                                2<asp:CheckBox id="chk3" runat="server"></asp:CheckBox>
                                3</td>
                        </tr>
                        <tr>
                            <td>
                                Champ de saisie</td>
                            <td>
                                <asp:TextBox id="txtSaisie" runat="server"  MaxLength="20"
Columns="20"></asp:TextBox>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Mot de passe</td>
                            <td>
                                <asp:TextBox id="txtmdp" runat="server"  MaxLength="10" Columns="10"
TextMode="Password"></asp:TextBox>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Boîte de saisie</td>
                            <td>
                                <asp:TextBox id="txtArea" runat="server"  Columns="20"
TextMode="MultiLine" Rows="3"></asp:TextBox>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Liste déroulante</td>
                            <td>
                                <asp:DropDownList id="cmbValeurs" runat="server"></asp:DropDownList>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Liste à choix unique</td>
                            <td>
                                <asp:ListBox id="lstSimple" runat="server"></asp:ListBox>
                                <asp:Button id="btnRazSimple" onclick="btnRazSimple_Click" runat="server"
EnableViewState="False" Text="Raz"></asp:Button>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Liste à choix multiple</td>
                            <td>
                                <asp:ListBox id="lstMultiple" runat="server"
SelectionMode="Multiple"></asp:ListBox>
                                <asp:Button id="razMultiple" onclick="razMultiple_Click" runat="server"
EnableViewState="False" Text="Raz"></asp:Button>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Champ caché</td>
                            <td>
                                <asp:Label id="lblSecret" runat="server"
visible="False"></asp:Label></td>
                        </tr>
                        <tr>
                            <td>
                                Bouton simple</td>
                            <td>
                                <input id="btnEffacer" onclick="effacer()" type="button" value="Effacer"
/></td>
                        </tr>
                        <tr>
                            <td>
```

```
                                Bouton [reset]</td>
                            <td>
                                <input id="btnReset" type="reset" value="Rétablir" /></td>
                        </tr>
                        <tr>
                            <td>
                                Bouton [submit]</td>
                            <td>
                                <asp:Button id="btnEnvoyer" onclick="btnEnvoyer_Click" runat="server"
EnableViewState="False" Text="Envoyer"></asp:Button>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </asp:Panel>
```

```
        </p>
    </form>
</body>
</html>
```

La page a deux conteneurs, un pour chaque vue : [panelform] pour la vue formulaire, [panelinfo] pour la vue informations. La liste des composants du conteneur [panelForm] est la suivante :

nom	type	propriétés	rôle
panelform	Panel	EnableViewState=true	vue formulaire
rdOui rdNon	RadioButton	EnableViewState=true GroupName=rdmarie	boutons radio
chk1 chk2 chk3	CheckBox	EnableViewState=true	cases à cocher
txtSaisie	TextBox	EnableViewState=true	champ de saisie
txtMdp	TextBox	EnableViewState=true	champ de saisie protégée
txtArea	TextBox	EnableViewState=true	boîte de saisie multilignes
cmbValeurs	DropDownList	EnableViewState=true	liste déroulante
lstSimple	ListBox	EnableViewState=true SelectionMode=Single	liste à sélection simple
btnRazSimple	Button	EnableViewState=false	désélectionne tous les éléments de lstSimple
lstMultiple	ListBox	EnableViewState=true SelectionMode=Multiple	liste à sélection multiple
btnRazMultiple	Button	EnableViewState=false	désélectionne tous les éléments de lstMultiple
lblSecret	Label	EnableViewState=true Visible=false	champ caché
btnEffacer	HTML standard		affiche une alerte
btnEnvoyer	Button	EnableViewState=false	bouton [submit] du formulaire
btnReset	HTML standard		bouton [reset] du formulaire

Le rôle du [VIEWSTATE] pour les composants est ici important. Tous les composants sauf les boutons doivent avoir la propriété [EnableViewState=true]. Pour en comprendre la raison, il faut se rappeler le fonctionnement de l'application. Supposons que le champ [txtSaisie] ait la propriété [EnableViewState=false] :

1. le client demande une première fois la page [form10.aspx]. Il obtient la vue [formulaire]
2. il le remplit et le poste avec le bouton [Envoyer]. Les champs de saisie sont alors postés et le serveur affecte aux composants serveur la valeur postée ou leur état [VIEWSTATE] s'ils en avaient un. Ainsi le champ [txtSaisie] se voit affectée la valeur saisie par l'utilisateur. Aussi pour cette étape, son état [VIEWSTATE] ne sert à rien. Comme résultat de l'opération, la vue [informations] est envoyée, en fait toujours la page [form10.aspx] mais avec un conteneur affiché différent.
3. l'utilisateur consulte cette nouvelle vue et utilise le lien [Retour au formulaire] pour revenir à celui-ci. Un POST est alors fait vers [form10.aspx]. Il n'y a alors au plus qu'une valeur postée : la valeur sélectionnée dans la liste d'informations par l'utilisateur, information non exploitée par la suite. Toujours est-il qu'il n'y a a pas de champ [txtSaisie] posté.

4. le serveur reçoit le POST et affecte aux composants serveur la valeur postée ou leur état [VIEWSTATE] s'ils en avaient un. Ici, [txtSaisie] n'a pas de valeur postée. Si son attribut [EnableViewState] est à [false], la chaîne vide lui sera affectée. Comme on souhaite qu'il ait la valeur saisie par l'utilisateur, il faut qu'il ait la propriété [EnableViewState=true].

Le conteneur [panelinfo] a les contrôles suivants :

nom	type	propriétés	rôle
panelinfo	Panel	EnableViewState=false	vue informations
lstInfos	ListBox	EnableViewState=false	liste d'informations récapitulant les valeurs saisies par l'utilisateur
LinkButton1	LinkButton	EnableViewState=false	lien de retour vers le formulaire

Au moment des tests, si on regarde le code HTML généré par le code de présentation ci-dessus, on pourra être étonné du code généré pour le champ caché [lblSecret] :

```
                    <tr>
                        <td>
                            Champ caché</td>
                        <td></td>
                    </tr>
```

Le composant [lblSecret] n'est pas traduit en code HTML car il a la propriété [Visible=false]. Cependant parce qu'il a la propriété [EnableViewState=true], sa valeur sera néanmoins conservée dans le champ caché [__VIEWSATE]. Aussi sera-t-on capable de la récupérer comme le montreront les tests.

Il nous reste à écrire les gestionnaires d'événements. Dans [Page_Load] nous initialiserons le formulaire :

```
Sub page_Load(sender As Object, e As EventArgs)
        ' la 1ère fois, on initialise les éléments
        ' les fois suivantes, ceux-ci retrouvent leurs valeurs par le VIEWSTATE
        if IsPostBack then return
        ' init formulaire
        ' panelinfo pas affiché
        panelinfo.visible=false
        ' paneform affiché
        panelform.visible=true
        ' boutons radio
        rdNon.Checked=true
        ' cases à cocher
        chk2.Checked=true
        ' champ de saisie
        txtSaisie.Text="qqs mots"
        ' champ mot de passe
        txtMdp.Text="ceciestsecret"
        ' boîte de saisie
        txtArea.Text="ligne"+ControlChars.CrLf+"ligne2"+ControlChars.CrLf
        ' combo
        dim i as integer
        for i=1 to 4
            cmbValeurs.Items.Add(new ListItem("choix"+i.ToString,i.ToString))
        next
        cmbValeurs.SelectedIndex=1
        ' liste à sélection simple
        for i=1 to 7
            lstSimple.Items.Add(new ListItem("simple"+i.ToString,i.ToString))
        next
        lstSimple.SelectedIndex=0
        ' liste à sélection multiple
        for i=1 to 10
            lstMultiple.Items.Add(new ListItem("multiple"+i.ToString,i.ToString))
        next
        lstMultiple.Items(0).Selected=true
        lstMultiple.Items(2).Selected=true
        ' champ caché
        lblSecret.Text="secret"
End Sub
```

Le clic sur les boutons [lstRazSimple] et [lstMultiple] :

```
Sub btnRazSimple_Click(sender As Object, e As EventArgs)
    ' raz liste simple
    lstSimple.SelectedIndex=-1
End Sub
```

```
Sub razMultiple_Click(sender As Object, e As EventArgs)
    ' raz liste multiple
    lstMultiple.SelectedIndex=-1
End Sub
```

Le clic sur le bouton [Envoyer] :

```
Sub btnEnvoyer_Click(sender As Object, e As EventArgs)
    ' le panel info est rendu visible et le panel formulaire est caché
  panelinfo.Visible=true
  panelform.visible=false
  ' on récupère les valeurs postées et on les met dans lstInfos
  ' boutons radio
  dim info as string="état marital : "+iif(rdoui.checked,"marié"," non marié")
  affiche(info)
  ' cases à cocher
  info=" cases cochées : "+iif(chk1.checked,"1 oui","1 non")+","+ _
      iif(chk2.checked,"2 oui","2 non")+","+iif(chk3.checked,"3 oui","3 non")
  affiche(info)
  ' champ de saisie
  affiche("champ de saisie : " + txtSaisie.Text.Trim)
  ' mot de passe
  affiche("mot de passe : " + txtMdp.Text.Trim)
  ' boîte de saisie
  dim lignes() as String
  lignes=new Regex("\r\n").Split(txtArea.Text.Trim)
  dim i as integer
  for i=0 to lignes.length-1
      lignes(i)="["+lignes(i).Trim+"]"
  next
  affiche("Boîte de saisie : " + String.Join(",",lignes))
  ' combo
  affiche("éléments sélectionnés dans combo : "+selection(cmbValeurs))
  ' liste simple
  affiche("éléments sélectionnés dans liste simple : "+selection(lstSimple))
  ' liste multiple
  affiche("éléments sélectionnés dans liste multiple : "+selection(lstMultiple))
  ' champ caché
  affiche ("Champ caché : " + lblSecret.Text)
End Sub

sub affiche(msg as String)
    ' affiche msg dans lstInfos
    lstInfos.Items.Add(msg)
end sub

       function selection(liste as ListControl) as string
           ' on parcourt les éléments de liste
           ' pour trouver ceux qui sont sélectionnés
           dim i as integer
           dim info as string=""
           for i=0 to liste.Items.Count-1
               if liste.Items(i).Selected then info+="[" + liste.Items(i).Text + "]"
           next
           return info
       end function
```

Enfin le clic sur le lien [Retour vers le formulaire] :

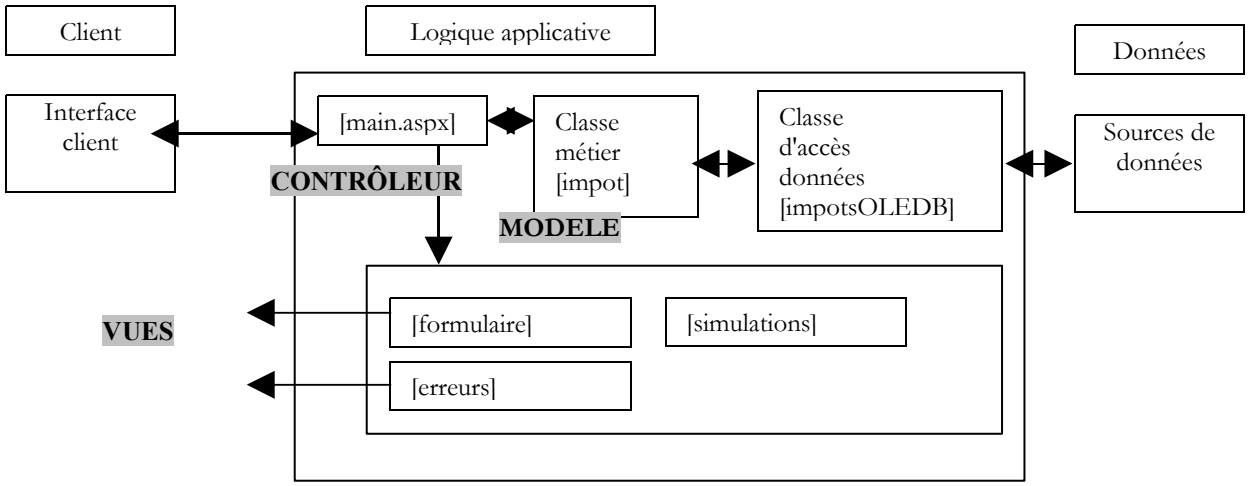
```
Sub LinkButton1_Click(sender As Object, e As EventArgs)
    ' on affiche le formulaire et on cache le panel info
    panelform.visible=true
    panelinfo.visible=false
End Sub
```

1.14.2 Exemple 2

Nous reprenons ici, une application déjà traitée dans les parties 1 et 2 de ce cours avec des formulaires HTML standard. L'application permet de faire des simulations de calculs d'impôt. Elle s'appuie sur une classe [impot] qu'on ne rappellera pas ici. Cette classe a besoin de données qu'elle trouve dans une source de données OLEDB. Pour l'exemple, ce sera une source ACCESS.

1.14.2.1 La structure MVC de l'application

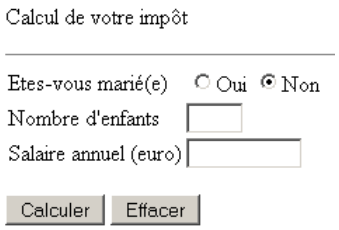
La structure MVC de l'application est la suivante :



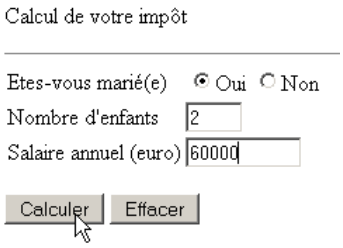
Les trois vues seront incorporées dans le code de présentation du contrôleur [main.aspx] sous la forme de conteneurs. Aussi cette application a-t-elle une unique page [main.aspx].

1.14.2.2 Les vues de l'application web

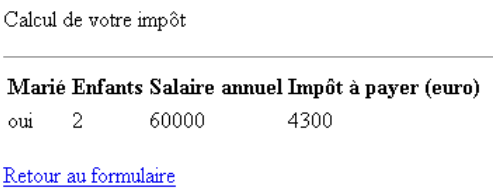
La vue [formulaire] est le formulaire de saisie des informations permettant le calcul de l'impôt d'un utilisateur :



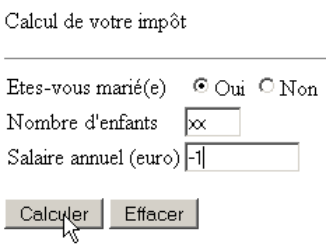
L'utilisateur remplit le formulaire :



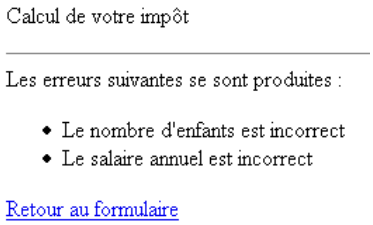
Il utilise le bouton [Envoyer] pour demander le calcul de son impôt. Il obtient la vue [simulations] suivante :



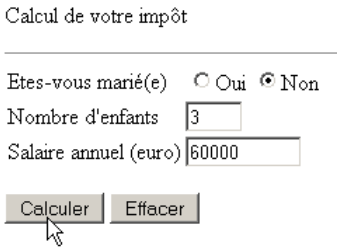
Il retourne au formulaire par le lien ci-dessus. Il le retrouve dans l'état où il l'a saisi. Il peut faire des erreurs de saisie :



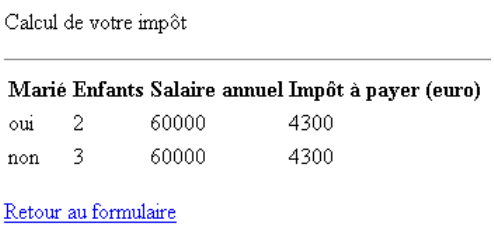
Celles-ci lui sont signalées par la vue [erreurs] :



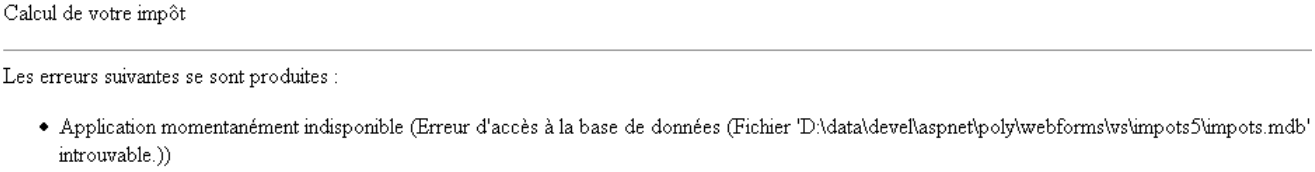
Il retourne au formulaire par le lien ci-dessus. Il le retrouve dans l'état où il l'a saisi. Il peut faire de nouvelles simulations :



Il obtient alors la vue [simulations] avec une simulation de plus :



Enfin, si la source de données n'est pas disponible, cela est signalé à l'utilisateur dans la vue [erreurs] :



1.14.2.3 Le code de présentation de l'application

Rappelons que la page [main.aspx] rassemble toutes les vues sous la forme de trois conteneurs :

- [panelform] pour la vue [formulaire]
- [panelerreurs] pour la vue [erreurs]
- [panelsimulations] pour la vue [simulations]

Nous renouons avec la séparation du code de de présentation et du code de contrôle en les mettant dans deux fichiers séparés (technique du code-behind de Visual Studio). Le premier sera dans [main.aspx] et le second dans [main.aspx.vb]. Le code de [main.aspx] est le suivant :

```
<%@ page codebehind="main.aspx.vb" inherits="vs.main" AutoEventWireUp="false" %>
<HTML>
 <HEAD>
   <title>Calcul d'impôt </title>
 </HEAD>
 <body>
   <P>Calcul de votre impôt</P>
   <HR width="100%" SIZE="1">
   <FORM id="Form1" runat="server">
     <asp:panel id="panelform" Runat="server">
       <TABLE id="Table1" cellSpacing="1" cellPadding="1" border="0">
         <TR>
           <TD height="19">Etes-vous marié(e)</TD>
           <TD height="19">
             <asp:RadioButton id="rdOui" runat="server" GroupName="rdMarie"></asp:RadioButton>Oui
             <asp:RadioButton id="rdNon" runat="server" GroupName="rdMarie"
Checked="True"></asp:RadioButton>Non</TD>
         </TR>
         <TR>
           <TD>Nombre d'enfants</TD>
           <TD>
             <asp:TextBox id="txtEnfants" runat="server" MaxLength="3" Columns="3"></asp:TextBox></TD>
         </TR>
         <TR>
           <TD>Salaire annuel (euro)</TD>
           <TD>
             <asp:TextBox id="txtSalaire" runat="server" MaxLength="10" Columns="10"></asp:TextBox></TD>
         </TR>
       </TABLE>
       <P>
         <asp:Button id="btnCalculer" runat="server" Text="Calculer"></asp:Button>
         <asp:Button id="btnEffacer" runat="server" Text="Effacer"></asp:Button></P>
     </asp:panel>
     <asp:panel id="panelerreurs" runat="server">
       <P>Les erreurs suivantes se sont produites :</P>
       <P>
         <asp:Literal id="erreursHTML" runat="server"></asp:Literal></P>
       <P></P>
       <asp:LinkButton id="lnkForm1" runat="server">Retour au formulaire</asp:LinkButton>
     </asp:panel>
     <asp:panel id="panelsimulations" runat="server">
       <P>
         <TABLE>
           <TR>
             <TH>
               Marié</TH>
             <TH>
               Enfants</TH>
             <TH>
               Salaire annuel</TH>
             <TH>
               Impôt à payer (euro)</TH></TR>
           <asp:Literal id="simulationsHTML" runat="server"></asp:Literal></TABLE>
         <asp:LinkButton id="lnkForm2" runat="server">Retour au formulaire</asp:LinkButton></P>
     </asp:panel>
   </FORM>
 </body>
</HTML>
```

Nous avons délimité les trois conteneurs. On notera qu'ils sont tous dans la balise <form runat="server">. C'est obligatoire, car si on veut pouvoir profiter des avantages des composants serveur, ceux-ci doivent être placés dans une telle balise. Le point important à saisir est qu'on a ici un unique formulaire que vont s'échanger le client et le serveur web. On est donc bien dans la configuration utilisée dans tout ce chapitre sur les composants serveur. Détaillons les composants de chaque conteneur :

Conteneur [panelform] :

nom	type	propriétés	rôle
panelform	Panel	EnableViewState=true	vue formulaire
rdOui rdNon	RadioButton	EnableViewState=true GroupName=rdmarie	boutons radio
txtEnfants	TextBox	EnableViewState=true	nombre d'enfants
txtSalaire	TextBox	EnableViewState=true	salaire annuel

btnCalculer	Button		bouton [submit] du formulaire - lance le calcul de l'impôt
btnEffacer	Button		bouton [submit] du formulaire - vide le formulaire

Conteneur [panelerreurs] :

nom	type	propriétés	rôle
panelerreurs	Panel	EnableViewState=true	vue erreurs
lnkForm1	LinkButton	EnableViewState=true	lien vers le formulaire
erreursHTML	Literal		code HTML de la liste d'erreurs

Conteneur [panelsimulations] :

nom	type	propriétés	rôle
panelsimulations	Panel	EnableViewState=true	vue simulations
lnkForm2	LinkButton	EnableViewState=true	lien vers le formulaire
simulationsHTML	Literal		code HTML de la liste des simulations dans une table HTML

1.14.2.4 Le code de contrôle de l'application

Le code de contrôle de l'application se trouve réparti dans les fichiers [global.asax.vb] et [main.aspx.vb]. Le fichier [global.asax] est défini comme suit :

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

Le fichier [global.asax.vb] est le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration
Imports System.Collections

Public Class Global
    Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on crée un objet impot
   Dim objImpot As impot
   Try
     objImpot = New impot(New impotsOLEDB(ConfigurationSettings.AppSettings("chaineConnexion")))
     ' on met l'objet dans l'application
     Application("objImpot") = objImpot
     ' pas d'erreur
     Application("erreur") = False
   Catch ex As Exception
     'il y a eu erreur, on le note dans l'application
     Application("erreur") = True
     Application("message") = ex.Message
   End Try
 End Sub

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' début de session - on crée une liste de simulations vides
   Session.Item("simulations") = New ArrayList
 End Sub
End Class
```

Lorsque l'application démarre (1ère requête faite à l'application), la procédure [Application_Start] est exécutée. Elle cherche à créer un objet de type [impot] prenant ses données dans une source OLEDB. Le lecteur est invité à lire le chapitre de la partie 2 du cours où cette classe a été définie, s'il l'a oubliée. La construction de l'objet [impot] peut échouer si la source de données n'est pas disponible. Dans ce cas, l'erreur est mémorisée dans l'application afin que toutes les requêtes ultérieures sachent que celle-ci n'a pu s'initialiser correctement. Si la construction se passe bien, l'objet [impot] créé est lui aussi mémorisé dans l'application. Il sera utilisé par toutes les requêtes de calcul d'impôt. Lorsqu'un client fait sa première requête, une session est créée pour lui par la procédure

[**Application_Start**]. Cette session est destinée à mémoriser les différentes simulations de calcul d'impôt qu'il va faire. Celles-ci seront mémorisées dans un objet [ArrayList] associé à la clé de session "simulations". Lorsque la session démarre, cette clé est associée à un objet [ArrayList] vide.

Les informations nécessaires à l'application sont placées dans son fichier de configuration [web.config] :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>
   <add key="chaineConnexion" value="Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data
Source=D:\data\serge\devel\aspnet\poly\webforms\vs\impots5\impots.mdb" />
 </appSettings>
</configuration>
```

La clé [chaineConnexion] désigne la chaîne de connexion à la source OLEDB.

L'autre partie du code de contrôle de l'application se trouve dans [main.aspx.vb] :

```
Imports System.Collections
Imports Microsoft.VisualBasic
Imports st.istia.univangers.fr
Imports System

Public Class main
 Inherits System.Web.UI.Page

 Protected WithEvents rdOui As System.Web.UI.WebControls.RadioButton
 Protected WithEvents rdNon As System.Web.UI.WebControls.RadioButton
 Protected WithEvents txtEnfants As System.Web.UI.WebControls.TextBox
 Protected WithEvents txtSalaire As System.Web.UI.WebControls.TextBox
 Protected WithEvents btnCalculer As System.Web.UI.WebControls.Button
 Protected WithEvents btnEffacer As System.Web.UI.WebControls.Button
 Protected WithEvents panelform As System.Web.UI.WebControls.Panel
 Protected WithEvents lnkForm1 As System.Web.UI.WebControls.LinkButton
 Protected WithEvents lnkForm2 As System.Web.UI.WebControls.LinkButton
 Protected WithEvents panelerreurs As System.Web.UI.WebControls.Panel
 Protected WithEvents panelsimulations As System.Web.UI.WebControls.Panel
 Protected WithEvents simulationsHTML As System.Web.UI.WebControls.Literal
 Protected WithEvents erreursHTML As System.Web.UI.WebControls.Literal

 ' variables locales

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

 Private Sub afficheFormulaire()
...
 End Sub

 Private Sub afficheSimulations(ByVal simulations As ArrayList, ByVal lien As String)
...
 End Sub

Private Sub afficheErreurs(ByVal erreurs As ArrayList, ByVal lien As String)
...
End Sub

 Private Sub btnCalculer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCalculer.Click
...
 End Sub

 Private Function checkData() As ArrayList
...
 End Function

 Private Sub lnkForm1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm1.Click
....
 End Sub

 Private Sub lnkForm2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm2.Click
...
 End Sub

 Private Sub btnEffacer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnEffacer.Click
```

```
...
 End Sub

 Private Sub razForm()
...
 End Sub
End Class
```

Le premier événement traité par le code est [Page_Load] :

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' tout d'abord, on regarde dans quel état est l'application
   If CType(Application("erreur"), Boolean) Then
     ' l'application n'a pas pu s'initialiser
     ' on affiche la vue erreurs sans lien de retour
     Dim erreurs As New ArrayList
     erreurs.Add("Application momentanément indisponible (" + CType(Application("message"), String) +
")")
     afficheErreurs(erreurs, "")
     Exit Sub
   End If
   ' pas d'erreurs - à la 1ère requête, on présente le formulaire
   If Not IsPostBack Then afficheFormulaire()
 End Sub
```

Rappelons que lorsque la procédure [Page_Load] s'exécute sur un POST client, tous les composants du formulaire ont une valeur : soit la valeur postée par le client s'il y en a une, soit la valeur précédente du composant grâce au [VIEWSTATE]. Dans ce formulaire, tous les composants ont la propriété [EnableViewState=true]. Avant de commencer à traiter la requête, nous nous assurons que l'application a pu s'initialiser correctement. Si ce n'est pas le cas, on affiche la vue [erreurs] avec la procédure [afficheErreurs]. Si c'est la première requête (IsPostBack=false), nous faisons afficher la vue [formulaire] avec [afficheFormulaire].

La procédure affichant la vue [erreurs] est la suivante :

```
 Private Sub afficheErreurs(ByVal erreurs As ArrayList, ByVal lien As String)
   ' affiche le conteneur erreurs
   panelerreurs.Visible = True
   Dim i As Integer
   erreursHTML.Text = ""
   For i = 0 To erreurs.Count - 1
     erreursHTML.Text += "<li>" + erreurs(i).ToString + "</li>" + ControlChars.CrLf
   Next
   lnkForm1.Text = lien
   ' les autres conteneurs sont cachés
   panelform.Visible = False
   panelsimulations.Visible = False
 End Sub
```

La procédure a deux paramètres :

- une liste de messages d'erreurs dans [erreurs]
- un texte de lien dans [lien]

Le code HTML à générer pour la liste d'erreurs est placé dans le litéral [erreursHTML]. Le texte du lien est lui placé dans la propriété [Text] de l'objet [LinkButton] de la vue.

La procédure affichant la vue [formulaire] est la suivante :

```
 Private Sub afficheFormulaire()
   ' affiche le formulaire
   panelform.Visible = True
   ' les autres conteneurs sont cachés
   panelerreurs.Visible = False
   panelsimulations.Visible = False
 End Sub
```

Cette procédure se contente de rendre visible le conteneur [panelform]. Les composants sont affichés avec leur valeur postée ou précédente (VIEWSTATE).

Lorsque l'utilisateur clique sur le bouton [Calculer] de la vue [formulaire], un POST vers [main.aspx] est fait. La procédure [Page_Load] est exécutée puis la procédure [btnCalculer_Click] :

```
 Private Sub btnCalculer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCalculer.Click
   ' on vérifie la validité des données saisies
   Dim erreurs As ArrayList = checkData()
```

```
    ' s'il y a des erreurs, on le signale
    If erreurs.Count <> 0 Then
      ' on affiche la page d'erreurs avec un lien de retour
      afficheErreurs(erreurs, "Retour au formulaire")
      Exit Sub
    End If
    ' pas d'erreurs - on calcule l'impôt
    Dim impot As Long = CType(Application("objImpot"), impot).calculer( _
    rdOui.Checked, CType(txtEnfants.Text, Integer), CType(txtSalaire.Text, Long))
    ' on rajoute le résultat aux simulations existantes
    Dim simulation() As String = New String() {CType(IIf(rdOui.Checked, "oui", "non"), String), _
     txtEnfants.Text.Trim, txtSalaire.Text.Trim, impot.ToString}
    ' on rajoute le résultat aux simulations existantes
    Dim simulations As ArrayList = CType(Session.Item("simulations"), ArrayList)
    simulations.Add(simulation)
    ' on affiche la page de résultat
    afficheSimulations(simulations, "Retour au formulaire")
  End Sub
```

La procédure commence par vérifier la validité des champs du formulaire avec la procédure [checkData] qui rend une liste [ArrayList] de messages d'erreurs. Si la liste n'est pas vide, alors la vue [erreurs] est affichée et la procédure est terminée. Si les données saisies sont valides, alors le montant de l'impôt est calculé grâce à l'objet de type [impot] qui avait été stocké dans l'application au démarrage de celle-ci. Cette nouvelle simulation est ajoutée à la liste des simulations déjà faites et stockée dans la session.

La fonction [CheckData] vérifie la validité des données. Elle rend une liste [ArrayList] de messages d'erreurs, vide si les données sont valides :

```
  Private Function checkData() As ArrayList
    ' au départ pas d'erreurs
    Dim erreurs As New ArrayList
    ' nbre d'enfants
    Try
      Dim nbEnfants As Integer = CType(txtEnfants.Text, Integer)
      If nbEnfants < 0 Then Throw New Exception
    Catch
      erreurs.Add("Le nombre d'enfants est incorrect")
    End Try
    ' salaire
    Try
      Dim salaire As Long = CType(txtSalaire.Text, Long)
      If salaire < 0 Then Throw New Exception
    Catch
      erreurs.Add("Le salaire annuel est incorrect")
    End Try
    ' on rend la liste des erreurs
    Return erreurs
  End Function
```

Enfin la vue [simulations] est affichée par la procédure [afficheSimulations] suivante :

```
  Private Sub afficheSimulations(ByVal simulations As ArrayList, ByVal lien As String)
    ' affiche la vue simulations
    panelsimulations.Visible = True
    ' les autres conteneurs sont cachés
    panelerreurs.Visible = False
    panelform.Visible = False
    ' contenu de la vue simulations
    ' chaque simulation est un tableau de 4 éléments string
    Dim simulation() As String
    Dim i, j As Integer
    simulationsHTML.Text = ""
    For i = 0 To simulations.Count - 1
      simulation = CType(simulations(i), String())
      simulationsHTML.Text += "<tr>"
      For j = 0 To simulation.Length - 1
        simulationsHTML.Text += "<td>" + simulation(j) + "</td>"
      Next
      simulationsHTML.Text += "</tr>" + ControlChars.CrLf
    Next
    ' lien
    lnkForm2.Text = lien
  End Sub
```

La procédure a deux paramètres :

- une liste de simulations dans [simulations]
- un texte de lien dans [lien]

Le code HTML à générer pour la liste de simulations est placé dans le litéral [simulationsHTML]. Le texte du lien est lui placé dans la propriété [Text] de l'objet [LinkButton] de la vue.

Lorsque l'utilisateur clique sur le bouton [Effacer] de la vue [formulaire], c'est la procédure [btnEffacer_click] qui s'exécute (toujours après [Page_Load]) :

```
 Private Sub btnEffacer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnEffacer.Click
   ' affiche le formulaire vide
   razForm()
   afficheFormulaire()
 End Sub

 Private Sub razForm()
   ' vide le formulaire
   rdOui.Checked = False
   rdNon.Checked = True
   txtEnfants.Text = ""
   txtSalaire.Text = ""
 End Sub
```

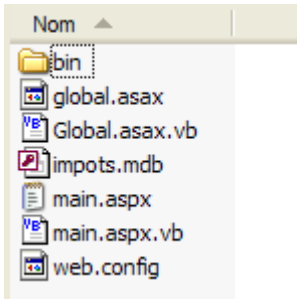
Le code ci-dessus est suffisamment simple pour ne pas avoir à être commenté. Il nous reste à gérer le clic sur les liens des vues [erreurs] et [simulations] :

```
 Private Sub lnkForm1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm1.Click
   ' affiche le formulaire
   afficheFormulaire()
 End Sub

 Private Sub lnkForm2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm2.Click
   ' affiche le formulaire
   afficheFormulaire()
 End Sub
```

Les deux procédures se contentent de faire afficher la vue [formulaire]. Nous savons que les champs de celle-ci vont obtenir une valeur qui est soit la valeur postée pour eux soit leur valeur précédente. Comme ici, le POST du client n'envoie aucune valeur pour les champs du formulaire, ceux-ci vont retrouver leur valeur précédente. Le formulaire est donc affiché avec les valeurs saisies par l'utilisateur. Nous nous souvenons que, dans la version sans composants serveur, nous avions fait ce travail de restauration nous-mêmes.

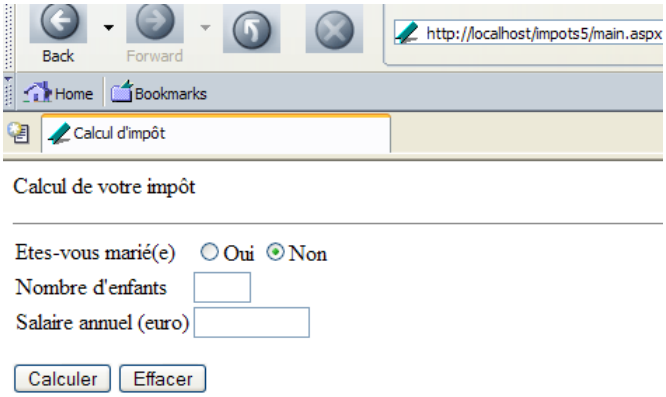
1.14.2.5 Tests

Tous les fichiers nécessaires à l'application sont placés dans un dossier <application-path> :

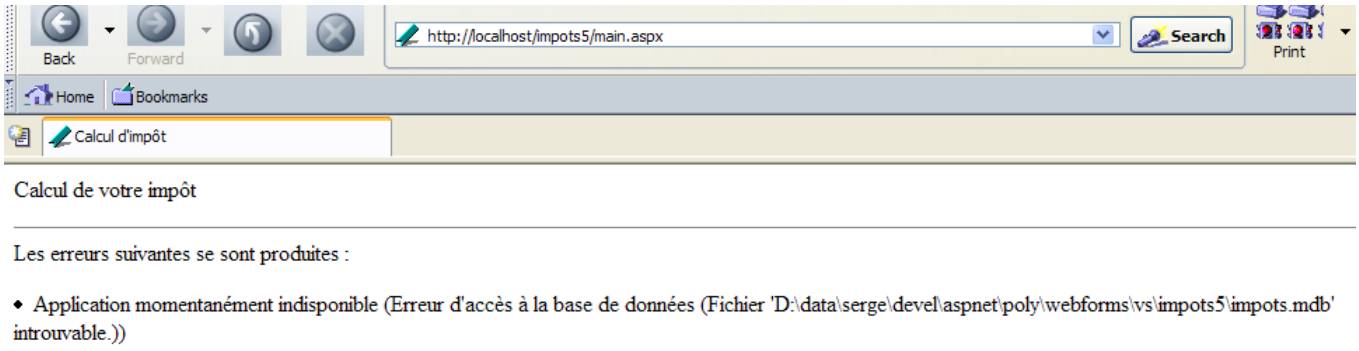


Le dossier [bin] contient la dll contenant les classes [impot], [impotsData], [impotsOLEDB] nécessaires à l'application :

Le lecteur pourra, s'il le souhaite, relire le chapitre 5 où est expliquée la façon de créer le fichier [impot.dll] ci-dessus. Ceci fait, le serveur Cassini est lancé avec les paramètres (<application-path>,/impots5). On demande l'url [http://impots5/main.aspx] avec un navigateur :

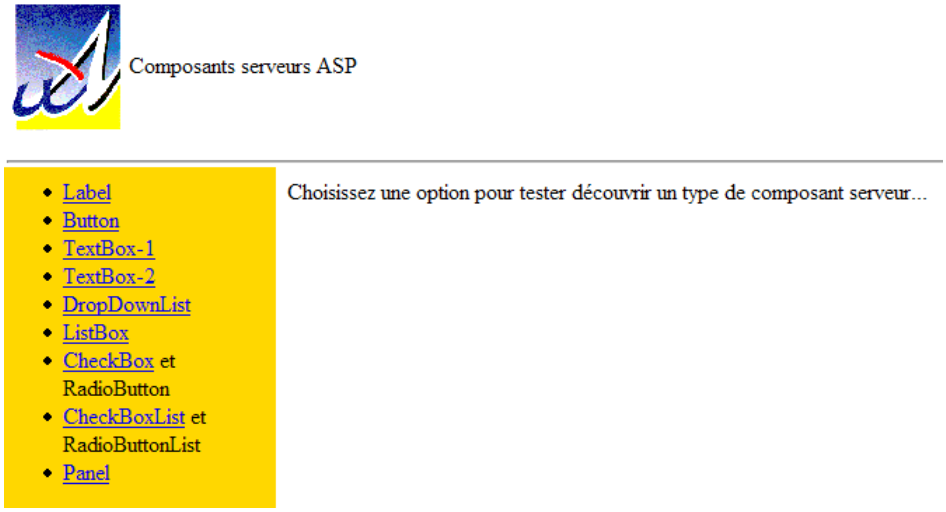


Si on renomme le fichier ACCESS [impots.mdb] en [impots1.mdb], on aura la page suivante :

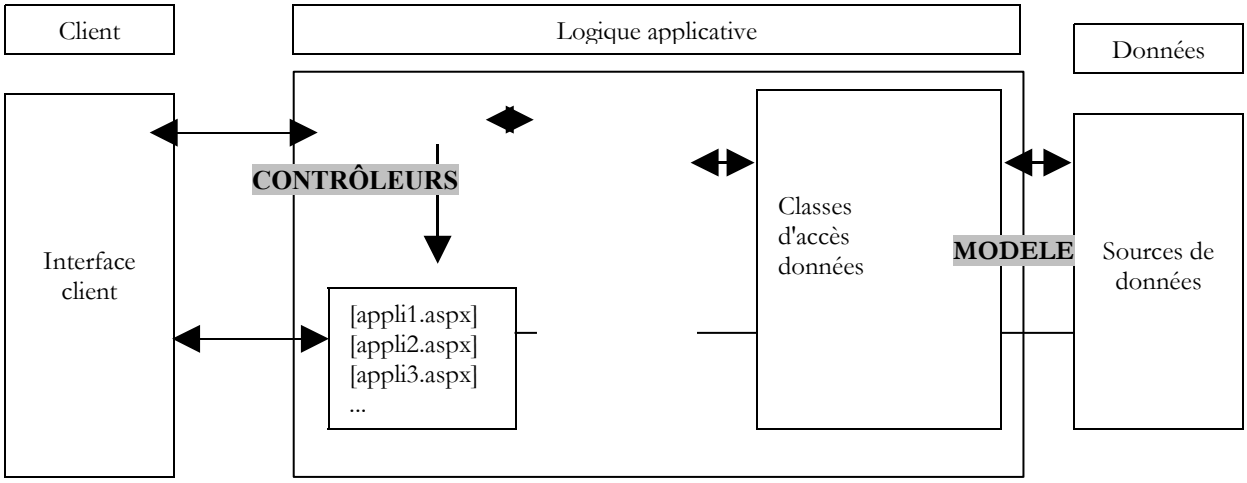


1.14.3 Exemple 3

Nous avons montré sur ces deux exemples qu'il était possible de construire des applications web respectant l'architecture MVC avec des composants serveur. L'exemple précédent montre que la solution avec composants serveur est plus simple que la solution utilisant des balises HTML standard. Nos deux exemples n'avaient qu'une page avec plusieurs vues au sein de la même page. On peut avoir une architecture MVC avec plusieurs formulaires ASP serveur tant que le fait que ceux-ci se postent à eux-mêmes les valeurs du formulaire ne pose pas de problème. C'est très souvent le cas des applications avec menu. Prenons l'exemple suivant :



Nous avons réuni dans une même page des liens vers les applications que nous avons écrites jusqu'à maintenant. Ce type d'application se prête bien à une architecture MVC. Simplement il n'y a plus un, mais plusieurs contrôleurs.



Le contrôleur [main.aspx] joue le rôle de contrôleur principal. C'est lui qui est appelé par les liens de la page d'accueil de l'application. Il va pouvoir faire des opérations communes à toutes les actions possibles, puis il fera exécuter l'action particulière associée au lien utilisé. Il passera alors la main à l'un des contrôleurs secondaires, celui chargé de faire exécuter l'action. A partir de ce moment, les échanges se font entre le client et ce contrôleur particulier. On ne passe plus par le contrôleur principal [main.aspx]. On n'est donc plus dans le cadre MVC avec contrôleur unique qui filtre toutes les requêtes. Chacun des contrôleurs ci-dessus peut présenter plusieurs vues avec le mécanisme des conteneurs au sein d'une unique page que nous avons présenté.

Le fait qu'on n'ait plus un contrôleur unique qui choisit les vues à envoyer au client présente des inconvénients. Prenons l'exemple de la gestion des erreurs. Chacune des actions exposées par l'application peut être amenée à afficher une vue d'erreurs. Chaque contrôleur [applix.aspx] va avoir sa propre vue [erreurs] parce que celle-ci est simplement un conteneur particulier de la page du contrôleur. Il n'y a pas moyen d'avoir une vue [erreurs] unique qui serait utilisée par toutes les applications individuelles. En effet, une telle vue présente généralement un lien de retour vers le formulaire erroné et celui-ci doit être restauré dans l'état où il a été validé, afin de permettre à l'utilisateur de corriger ses erreurs. Cette restauration se fait par le mécanisme du [VIEWSTATE] qui ne fonctionne pas entre contrôleurs différents. Si les applications sont développées par des personnes différentes, on risque d'avoir des pages d'erreurs à l'aspect différent selon l'action choisie par l'utilisateur, entachant ainsi l'homogénéité de l'application globale. Nous verrons un peu plus tard, qu'ASP.NET offre une solution à ce problème particulier de vue à partager. Celle-ci peut faire l'objet d'un nouveau composant serveur que nous construisons nous-mêmes. Il suffit d'utiliser ce composant dans les différentes applications pour assurer l'homgénéité de l'application globale. Plus difficile à gérer, le problème de l'ordre des actions. Lorsque toutes les requêtes passent par un contrôleur unique, celui-ci peut vérifier que l'action demandée est compatible avec la précédente. Ce code de contrôle est à un unique endroit. Ici, il va falloir le répartir sur les différents contrôleurs compliquant la maintenance de l'application globale.

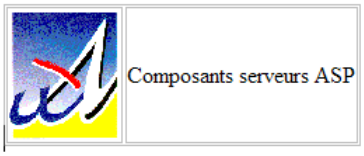
Revenons à notre application ci-dessus. La page d'entrée de celle-ci est une page HTML classique [accueil.htm] :

```
<html>
 <head>
   <TITLE>Composants ASP Serveur</TITLE>
   <meta name="pragma" content="no-cache">
 </head>
 <frameset rows="130,*" frameborder="0">
   <frame name="banner" src="bandeau.htm" scrolling="no">
   <frameset cols="200,*">
     <frame name="contents" src="options.htm">
     <frame name="main" src="main.htm">
   </frameset>
   <noframes>
     <p id="p1">
       Ce jeu de frames HTML affiche plusieurs pages Web. Pour afficher ce jeu de
       frames, utilisez un navigateur Web qui prend en charge HTML 4.0 et version
       ultérieure.
     </p>
   </noframes>
 </frameset>
</html>
```

Cette page d'accueil est un ensemble de trois cadres appelés **banner**, **contents** et **main** :

banner : bandeau.htm	
contents : options.htm	**main** : main.htm

La page [bandeau.htm] placée dans le cadre [banner] est la suivante :



Son code HTML est le suivant :

```
<html>
 <head>
   <META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE" />
   <title>bandeau</title>
 </head>
 <body>
   <P>
     <TABLE>
       <TR>
         <TD><IMG alt="logo université d'angers" src="univ01.gif"></TD>
         <TD>Composants serveurs ASP</TD>
       </TR>
     </TABLE>
   </P>
   <HR>
 </body>
</html>
```

La page [options.htm] est placée dans le cadre [contents]. C'est un ensemble de liens :

- Label
- Button
- TextBox-1
- TextBox-2
- DropDownList
- ListBox
- CheckBox et RadioButton
- CheckBoxList et RadioButtonList
- Panel

```
<html>
 <head>
   <meta http-equiv="pragma" content="no-cache" />
   <title>options</title>
 </head>
 <body bgcolor="Gold">
   <ul>
     <li>
       <a href="main.aspx?action=label" target="main">Label</a>
     <li>
       <a href="main.aspx?action=button" target="main">Button</a>
     <li>
       <a href="main.aspx?action=textbox1" target="main">TextBox-1</a></li>
     <li>
       <a href="main.aspx?action=textbox2" target="main">TextBox-2</a></li>
     <li>
       <a href="main.aspx?action=dropdownlist" target="main">DropDownList</a></li>
     <li>
       <a href="main.aspx?action=listbox" target="main">ListBox</a></li>
     <li>
       <a href="main.aspx?action=casesacocher" target="main">CheckBox</a> et<br>
       RadioButton</li>
     <li>
       <a href="main.aspx?action=listecasesacocher" target="main">CheckBoxList</a>
et<br>
       RadioButtonList</a></li>
     <li>
       <a href="main.aspx?action=panel" target="main">Panel</a></li>
   </ul>
 </body>
</html>
```

Les différents liens pointent tous sur le contrôleur principal [main.aspx] avec un paramètre [action] indiquant l'action à faire. On demande à ce que la cible des liens soit affichée dans le cadre [main] (target="main").

La première page affichée dans le cadre [main] est [main.htm] :

```
<html>
 <head>
   <title>main</title>
 </head>
 <body>
   <P>Choisissez une option pour tester découvrir un type de composant serveur...</P>
 </body>
</html>
```

Le contrôleur principal [main.aspx, main.aspx.vb] est le suivant :

[main.aspx]

```
<%@ page src="main.aspx.vb" inherits="main" autoeventwireup="false" %>
```

[main.aspx.vb]

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

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' on récupère l'action à faire
   Dim action As String
   If Request.QueryString("action") Is Nothing Then
     action = "label"
   Else
     action = Request.QueryString("action").ToString.ToLower
   End If
   ' on exécute l'action
   Select Case action
     Case "label"
       Server.Transfer("form2.aspx")
     Case "button"
       Server.Transfer("form3.aspx")
     Case "textbox1"
       Server.Transfer("form4.aspx")
     Case "textbox2"
       Server.Transfer("form5.aspx")
     Case "dropdownlist"
       Server.Transfer("form6.aspx")
     Case "listbox"
       Server.Transfer("form7.aspx")
     Case "casesacocher"
       Server.Transfer("form8.aspx")
     Case "listecasesacocher"
       Server.Transfer("form8b.aspx")
     Case "panel"
       Server.Transfer("form9.aspx")
     Case Else
       Server.Transfer("form2.aspx")
   End Select
 End Sub
End Class
```

Notre contrôleur est simple. Selon la valeur du paramètre [action], il transfère le traitement de la requête à la page adéquate. Il n'apporte aucune plus-value vis à vis d'une page HTML avec des liens. Néanmoins, il suffirait d'ajouter une page d'authentification pour voir son intérêt. Si l'utilisateur devait s'authentifier (login, mot de passe) pour avoir accès aux applications, le contrôleur [main.aspx] serait un bon endroit pour vérifier que cette authentification a été faite.

2 Composants serveur ASP - 2

2.1 Introduction

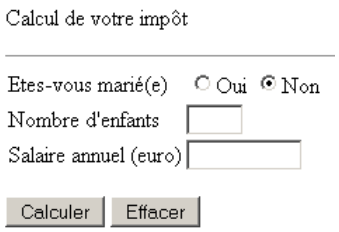
Nous continuons notre travail sur l'interface utilisateur en découvrant :

- des composants de validation de données
- la façon de lier des données à des composants serveur
- les composants serveur HTML

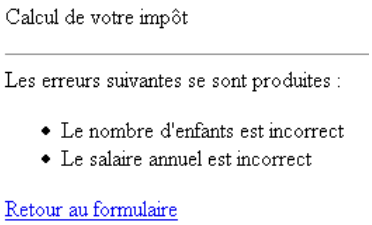
2.2 Les composants de validation de données

2.2.1 Introduction

Dans des applications à formulaires, il est primordial de vérifier la validité des données saisies dans ceux-ci. Dans l'exemple sur le calcul d'impôt, nous avions le formulaire suivant :



A réception des valeurs de ce formulaire, il nous faut vérifier la validité des données saisies : le nombre d'enfants et le salaire doivent être des entiers positifs ou nuls. Si ce n'est pas le cas, le formulaire est renvoyé au client tel qu'il a été saisi avec des messages indiquant les erreurs. Nous avions traité ce cas avec deux vues, l'une pour le formulaire ci-dessus et l'autre pour les erreurs :



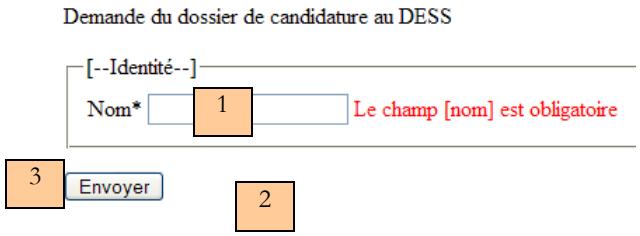
ASP.NET offre des composants appelés composants de validation qui permettent de vérifier les cas suivants :

Composant	Rôle
`RequiredFieldValidator`	vérifie qu'un champ est non vide
`CompareValidator`	vérifie deux valeurs entre-elles
`RangeValidator`	vérifie q'une valeur est entre deux bornes
`RegularExpressionValidator`	vérifie qu'un champ vérifie une expression régulière
`CustomValidator`	permet au développeur de donner ses propres règles de validation - ce composant pourrait remplacer tous les autres
`ValidationSummary`	permet de rassembler les messages d'erreurs émis par les contrôles précédents en un unique endroit de la page

Nous allons maintenant présenter chacun de ces composants.

2.2.2 RequiredFieldValidator

Nous construisons la page [requiredfieldvalidator1.aspx] suivante :



n°	nom	type	propriétés	rôle
1	txtNom	TextBox	EnableViewState=true	champ de saisie
2	RequiredFieldValidator1	RequiredFieldValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Le champ [nom] est obligatoire	composant de validation
3	btnEnvoyer	Button	EnableViewState=false CausesValidation=true	bouton [submit]

Le champ [RequiredFieldValidator1] sert à afficher un message d'erreur si le champ [txtNom] est vide ou contient une suite d'espaces. Ses propriétés sont les suivantes :

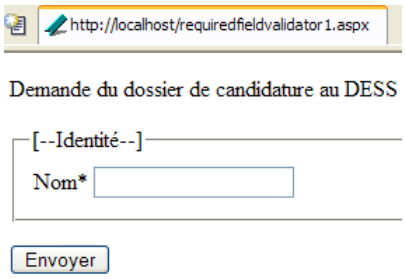
ControlToValidate	champ dont la valeur doit être contrôlée par le composant. Que signifie la valeur d'un composant ? C'est la valeur de l'attribut [value] de la balise HTML correspondante. Pour un [TextBox] ce sera le contenu du champ de saisie, pour un [DropDownList] la valeur de l'élément sélectionné.
EnableClientScript	booléen - à vrai indique que le contenu du champ précédent doit être contrôlé également côté client. Dans ce cas, le formulaire ne sera posté par le navigateur que si le formulaire ne comporte pas d'erreurs. Néanmoins, les tests de validation sont faits également sur le serveur pour le cas où le client ne serait pas un navigateur par exemple.
ErrorMessage	le message d'erreur que le composant doit afficher en cas d'erreur détectée

La vérification des données de la page par les contrôles de validation n'est faite que si le bouton ou le lien ayant provoqué le [POST] de la page a la propriété [CausesValidation=true]. [true] est la valeur par défaut de cette propriété.

Le code ASPX / HTML de la page est le suivant :

```
1.  <%@ Page Language="VB" %>
2.  <html>
3.  <head>
4.  </head>
5.  <body>
6.      <form runat="server">
7.          <p>
8.              Demande du dossier de candidature au DESS
9.          </p>
10.         <fieldset>
11.             <legend>[--Identité--]</legend>
12.             <table>
13.                 <tbody>
14.                     <tr>
15.                         <td>
16.                             Nom*</td>
17.                         <td>
18.                             <asp:TextBox id="txtNom" runat="server"></asp:TextBox>
19.                             <asp:RequiredFieldValidator id="RequiredFieldValidator1"
    runat="server" ErrorMessage="Le champ [nom] est obligatoire" ControlToValidate="txtNom"
    EnableViewState="False"></asp:RequiredFieldValidator>
20.                         </td>
21.                     </tr>
22.                 </tbody>
23.             </table>
24.         </fieldset>
25.         <p>
26.             <asp:Button id="btnEnvoyer" runat="server" Text="Envoyer"></asp:Button>
27.         </p>
28.     </form>
29. </body>
30. </html>
```

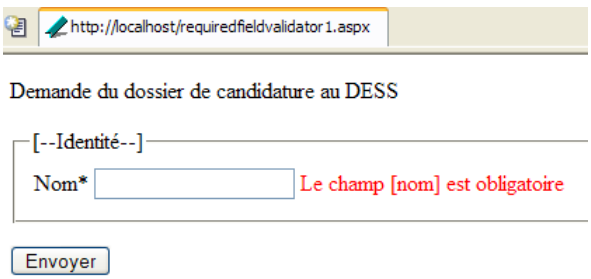
Exécutons cette application. Nous utilisons tout d'abord un navigateur [Mozilla] :



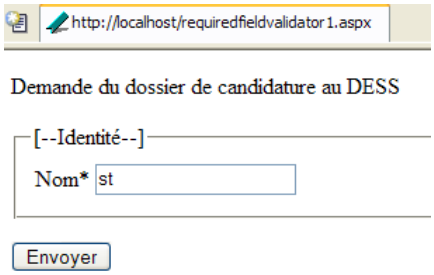
Le code HTML reçu par [Mozilla] est le suivant :

```
<html>
<head>
</head>
<body>
    <form name="_ctl0" method="post" action="requiredfieldvalidator1.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwxNDI1MDc1NTU1Ozs+SGtdZvVxefDCDxnsqbDnqCaROsk=" />

        <p>
            Demande du dossier de candidature au DESS
        </p>
        <fieldset>
            <legend>[--Identité--]</legend>

            <table>
                <tbody>
                    <tr>
                        <td>
                            Nom*</td>
                        <td>
                            <input name="txtNom" type="text" id="txtNom" />
                             
                        </td>
                    </tr>
                </tbody>
            </table>
        </fieldset>
        <p>
            <input type="submit" name="btnEnvoyer" value="Envoyer" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); " language="javascript"
id="btnEnvoyer" />
        </p>
    </form>

</body>
</html>
```

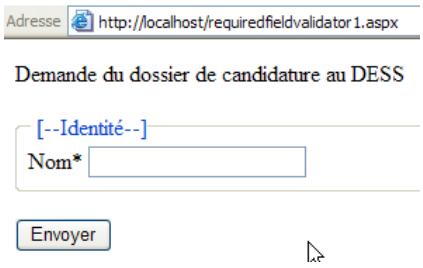
On voit que le bouton [btnEnvoyer] est lié à une fonction Javascript via son attribut [onclick]. On peut remarquer également que la page ne contient aucun code [Javascript]. Le test [typeof(Page_ClientValidate) == 'function'] va échouer et la fonction javascript ne sera pas appelée. Le formulaire sera alors posté au serveur et c'est lui qui fera les contrôles de validité. Utilisons le bouton [Envoyer] sans mettre de nom. La réponse du serveur est la suivante :



Que s'est-il passé ? Le formulaire a été posté au serveur. Celui-ci a fait exécuter le code de tous les composants de validation se trouvant sur la page. Ici il n'y en a qu'un. Si au moins un composant de validation détecte une erreur, le serveur renvoie le formulaire tel qu'il a été saisi (en fait pour les composants ayant [EnableViewState=true], avec de plus un message d'erreur pour chaque composant de validation ayant détecté une erreur. Ce message est l'attribut [ErrorMessage] du composant de validation. C'est ce mécanisme que nous voyons à l'oeuvre ci-dessus. Si nous mettons quelque chose dans le champ [nom], le message d'erreur n'apparaît plus lorsqu'on valide le formulaire :



Maintenant, utilisons Internet Explorer et demandons l'url [http://localhost/requiredfieldvalidator1.aspx]. Nous obtenons la page suivante :



Le code HTML reçu par IE est le suivant :

```
<html>
<head>
</head>
<body>
    <form name="_ctl0" method="post" action="requiredfieldvalidator1.aspx" language="javascript"
onsubmit="ValidatorOnSubmit();" id="_ctl0">
<input type="hidden" name="__VIEWSTATE" value="dDwxNDI1MDc1NTU1Ozs+SGtdZvVxefDCDxnsqbDnqCaROsk=" />
```

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

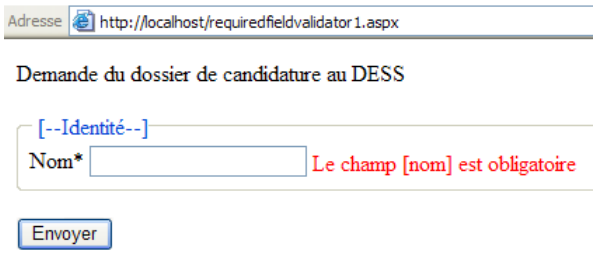
```


        <p>
            Demande du dossier de candidature au DESS
        </p>
        <fieldset>
            <legend>[--Identité--]</legend>
            <table>
                <tbody>
                    <tr>
                        <td>
                            Nom*</td>
                        <td>
                            <input name="txtNom" type="text" id="txtNom" />
                            <span id="RequiredFieldValidator1" controltovalidate="txtNom"
errormessage="Le champ [nom] est obligatoire" evaluationfunction="RequiredFieldValidatorEvaluateIsValid"
initialvalue="" style="color:Red;visibility:hidden;">Le champ [nom] est obligatoire</span>
                        </td>
                    </tr>
                </tbody>
            </table>
        </fieldset>
        <p>
            <input type="submit" name="btnEnvoyer" value="Envoyer" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); " language="javascript"
id="btnEnvoyer" />
        </p>

<script language="javascript">
<!--
 var Page_Validators =  new Array(document.all["RequiredFieldValidator1"]);
   // -->
</script>


<script language="javascript">
<!--
var Page_ValidationActive = false;
if (typeof(clientInformation) != "undefined" && clientInformation.appName.indexOf("Explorer") != -1) {
    if (typeof(Page_ValidationVer) == "undefined")
```

```
        alert("Impossible de trouver la bibliothèque de scripts
/aspnet_client/system_web/1_1_4322/WebUIValidation.js. Essayez de placer ce fichier manuellement ou
effectuez une réinstallation en exécutant 'aspnet_regiis -c'.");
    else if (Page_ValidationVer != "125")
        alert("Cette page utilise une version incorrecte de WebUIValidation.js. La page requiert la
version 125. La bibliothèque de scripts est " + Page_ValidationVer + ".");
    else
        ValidatorOnLoad();
}

function ValidatorOnSubmit() {
    if (Page_ValidationActive) {
        ValidatorCommonOnSubmit();
    }
}
// -->
</script>


   </form>
</body>
</html>
```

On voit que ce code est beaucoup plus important que celui reçu par [Mozilla]. Nous n'entrerons pas dans les détails. La différence vient du fait que le serveur a inclus des fonctions javascript dans le document HTML envoyé. Pourquoi deux codes HTML différents alors que l'url demandée par les deux navigateurs est la même ? Nous avons eu l'occasion de dire que la technologie ASP.NET faisait que le serveur adaptait le document HTML envoyé au client à la nature de cleui-ci. Il existe sur le marché différents navigateurs n'ayant pas les mêmes capacités. Les navigateurs Microsoft et Netscape par exemple, n'utilisent pas la même modélisation objet pour le document qu'ils reçoivent. Aussi le code Javascript pour manipuler ce document côté navigateur présente-t-il des différences entre les deux navigateurs. De même, les éditeurs ont créé des versions successives de leur navigateur, améliorant sans cesse leurs capacités. Aussi un document HTML compris par IE5 peut ne pas l'être par IE3. Nous avons ci-dessus un exemple de cette adaptation du serveur au niveau de son client. Pour une raison qui n'a pas été approfondie ici, le serveur web a estimé que le client [Mozilla] n'avait pas la capacité de gérer le code javascript de validation. Aussi ce code n'a-t-il pas été placé dans le document HTML qui lui a été envoyé et la validation s'est faite côté serveur. Pour [IE6], ce code javascript a été inclus dans le document HTML envoyé, comme nous pouvons le constater. Pour le voir à l'oeuvre, faisons l'expérience suivante :

- arrêtons le serveur web
- utilisons le bouton [Envoyer] sans remplir le champ [nom]

Nous obtenons la nouvelle page suivante :



C'est bien le navigateur qui a placé le message d'erreur. En effet, le serveur est arrêté. On peut s'en assurer en mettant quelque chose dans le champ [nom] et en validant. Cette fois la réponse est la suivante :

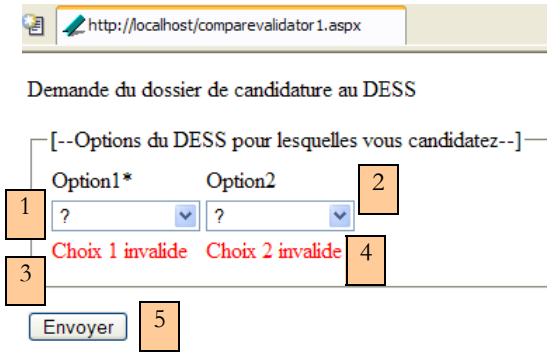
Que s'est-il passé ? Le navigateur a exécuté le code Javascript de validation. Ce code a dit que la page était valide. Le navigateur a donc posté le formulaire au serveur web qui était arrêté. Le navigateur s'en est aperçu et a affiché la page ci-dessus. Si on relance le serveur, tout redevient normal.

Que retenir de cet exemple ?

- l'intérêt de la notion de composant de validation qui permet de renvoyer au client tout formulaire incorrect
- l'intérêt du [VIEWSTATE] qui permet de renvoyer ce formulaire tel qu'il a été saisi
- la faculté d'adaptation du serveur à son client. Celui-ci est identifié par l'entête HTTP [User-Agent:] qu'il envoie au serveur. Ce n'est donc pas le serveur qui "devine" à qui il a affaire. Cette faculté d'adaptation présente un grand intérêt pour le développeur qui n'a pas à se préoccuper du type de client de son application.

2.2.3 CompareValidator

Nous construisons la page [comparevalidator1.aspx] suivante :



n°	nom	type	propriétés	rôle
1	cmbChoix1	DropDownList	EnableViewState=true	liste déroulante
2	cmbChoix2	DropDownList	EnableViewState=true	liste déroulante
3	CompareValidator1	CompareValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Choix 1 invalide ValueToCompare=? Operator=NotEqual Type=string ControlToValidate=cmbChoix1	composant de validation
4	CompareValidator2	CompareValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Choix 2 invalide ControlToCompare=cmbChoix1 Operator=NotEqual Type=string ControlToValidate=cmbChoix2	composant de validation
5	btnEnvoyer	Button	EnableViewState=false CausesValidation=true	bouton [submit]

Le composant [CompareValidator] sert à comparer deux valeurs entre-elles. Les opérateurs utilisables sont [Equal, NotEqual, LessThan, LessThanEqual, GreaterThan, GreaterThanEqual]. La première valeur est fixée par la propriété [ControlToValidate], la seconde par [ValueToCompare] si la valeur précédente doit être comparée à une constante, ou par [ControlToCompare] si elle doit être comparée à la valeur d'un autre composant. Les propriétés importantes du composant [CompareValidator] sont les suivantes :

ControlToValidate	champ dont le contenu doit être contrôlé par le composant
EnableClientScript	booléen - à vrai indique que le contenu du champ précédent doit être contrôlé également côté client
ErrorMessage	le message d'erreur que le composant doit afficher en cas d'erreur détectée
ValeurToCompare	valeur à laquelle doit être comparée la valeur du champ [ControlToValidate]
ControlToCompare	composant à la valeur duquel doit être comparée la valeur du champ [ControlToValidate]
Operator	opérateur de comparaison entre les deux valeurs
Type	type des valeurs à comparer

Le code de présentation de cette page est le suivant :

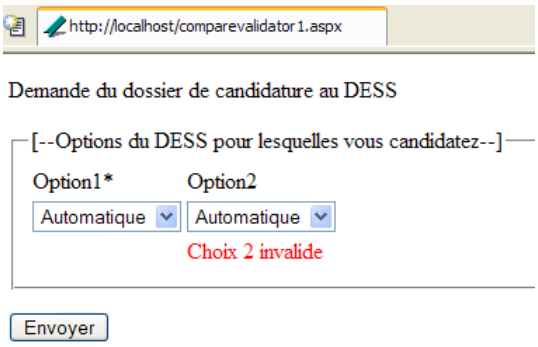
```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            Demande du dossier de candidature au DESS
```

```
        </p>
        <fieldset>
            <legend>[--Options du DESS pour lesquelles vous candidatez--]</legend>
            <table>
                <tbody>
                    <tr>
                        <td>
                            Option1*</td>
                        <td>
                            Option2</td>
                    </tr>
                    <tr>
                        <td>
                            <asp:DropDownList id="cmbChoix1" runat="server">
                                <asp:ListItem Value="?" Selected="True">?</asp:ListItem>
                                <asp:ListItem Value="Automatique">Automatique</asp:ListItem>
                                <asp:ListItem Value="Informatique">Informatique</asp:ListItem>
                            </asp:DropDownList>
                        </td>
                        <td>
                            <asp:DropDownList id="cmbChoix2" runat="server">
                                <asp:ListItem Value="?">?</asp:ListItem>
                                <asp:ListItem Value="Automatique">Automatique</asp:ListItem>
                                <asp:ListItem Value="Informatique">Informatique</asp:ListItem>
                            </asp:DropDownList>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <asp:CompareValidator id="CompareValidator1" runat="server"
ErrorMessage="Choix 1 invalide" ControlToValidate="cmbChoix1" ValueToCompare="?"
Operator="NotEqual"></asp:CompareValidator>
                        </td>
                        <td>
                            <asp:CompareValidator id="CompareValidator2" runat="server"
ErrorMessage="Choix 2 invalide" ControlToValidate="cmbChoix2" Operator="NotEqual"
ControlToCompare="cmbChoix1"></asp:CompareValidator>
                        </td>
                    </tr>
                </tbody>
            </table>
        </fieldset>
        <p>
            <asp:Button id="btnEnvoyer" runat="server" Text="Envoyer"></asp:Button>
        </p>
    </form>
</body>
</html>
```

Les contraintes de validation de la page sont les suivantes :

- la valeur sélectionnée dans [cmbChoix1] doit être différente de la chaîne "?". Ceci implique les propriétés suivantes pour le composant [CompareValidator1] : Operator=NotEqual, ValueToCompare=?, Type=string, ControlToValidate=cmbChoix1
- la valeur sélectionnée dans [cmbChoix2] doit être différente de celle sélectionnée dans [cmbChoix1]. Ceci implique les propriétés suivantes pour le composant [CompareValidator2] : Operator=NotEqual, ControlToCompare=cmbChoix1, Type=string, ControlToValidate=cmbChoix2

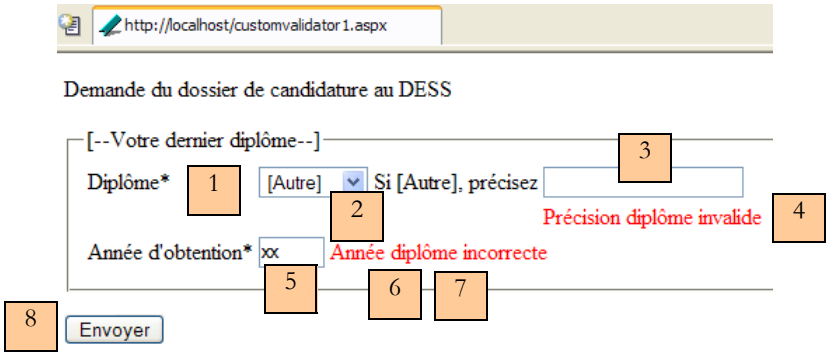
Nous exécutons cette application. Voici un exemple de page reçue au cours des échanges client-serveur :



Si la même page est demandée par Internet Explorer 6, du code Javascript est inclus dans celle-ci, ce qui amène à un fonctionnement légèrement différent. Les éventuelles erreurs sont signalées dès que l'utilisateur change de valeur dans l'une des listes déroulantes. C'est une amélioration du confort de l'utilisateur.

2.2.4 CustomValidator, RangeValidator

Nous construisons la page [customvalidator1.aspx] suivante :



n°	nom	type	propriétés	rôle
1	cmbDiplomes	DropDownList	EnableViewState=true	liste déroulante
2	CompareValidator2	CompareValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Diplôme invalide Operator=NotEqual ValueToCompare=? Type=String ControlToValidate=cmbDiplomes	composant de validation du contrôle [1]
3	txtAutreDiplome	TextBox	EnableViewState=false	champ de saisie
4	CustomValidator1	CustomValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Précision diplôme invalide ClientValidationFunction=chkAutreDiplome	composant de validation du champ [3]
5	txtAnDiplome	TextBox	EnableViewState=true	champ de saisie
6	RangeValidator1	RangeValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Année diplôme doit être dans l'intervalle [1990,2004] MinValue=1990 MaxValue=2004 Type=Integer ControlToValidate=txtAnDiplome	composant de validation du champ [5]
7	RequiredFieldValidator2	RequiredFieldValidator	EnableViewState=false EnableClientScript=true ErrorMessage=Année diplôme requise ControlToValidate=txtAnDiplome	composant de validation du champ [5]
8	btnEnvoyer	Button	EnableViewState=false CausesValidation=true	bouton [submit]

Le composant [RangeValidator] sert à vérifier que la valeur d'un contrôle se trouve entre deux bornes [MinValue] et [MaxValue]. Ses propriétés sont les suivantes :

ControlToValidate	champ dont la valeur doit être contrôlée par le composant
EnableClientScript	booléen - à vrai indique que le contenu du champ précédent doit être contrôlé également côté client
ErrorMessage	le message d'erreur que le composant doit afficher en cas d'erreur détectée
MinValue	valeur min de la valeur du champ à contrôler
MaxValue	valeur max de la valeur du champ à contrôler
Type	type de la valeur du champ à contrôler

Le composant [CustomValidator] permet de faire des validations que ne savent pas faire les composants de validation proposés par ASP.NET. Cette validation est faite par une fonction écrite par le développeur. Celle-ci est exécutée côté serveur. Comme le contrôle peut être fait également côté client, le développeur peut être amené à développer une fonction javascript qu'il inclura dans le document HTML. Les propriétés du composant [CustomValidator] sont les suivantes :

`ControlToValidate`	champ dont la valeur doit être contrôlée par le composant
`EnableClientScript`	booléen - à vrai indique que le contenu du champ précédent doit être contrôlé également côté client
`ErrorMessage`	le message d'erreur que le composant doit afficher en cas d'erreur détectée
`ClientValidationFunction`	la fonction à exécuter côté client

Le code de présentation de la page est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">
...
</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <p align="left">
            Demande du dossier de candidature au DESS
        </p>
        <fieldset>
            <legend>[--Votre dernier diplôme--]</legend>
            <table>
                <tbody>
                    <tr>
                        <td>
                            Diplôme*</td>
                        <td>
                            <asp:DropDownList id="cmbDiplomes" runat="server">
                                <asp:ListItem Value="?">?</asp:ListItem>
                                <asp:ListItem Value="[Autre]">[Autre]</asp:ListItem>
                                <asp:ListItem Value="Ma&#238;trise">Ma&#238;trise</asp:ListItem>
                                <asp:ListItem Value="DESS">DESS</asp:ListItem>
                                <asp:ListItem Value="DEA">DEA</asp:ListItem>
                            </asp:DropDownList>
                        </td>
                        <td>
                            Si [Autre], précisez</td>
                        <td>
                            <asp:TextBox id="txtAutreDiplome" runat="server"
EnableViewState="False"></asp:TextBox>
                        </td>
                    </tr>
                    <tr>
                        <td>
                        </td>
                        <td>
                            <p>
                                <asp:CompareValidator id="CompareValidator2" runat="server"
ErrorMessage="Diplôme invalide" ControlToValidate="cmbDiplomes" ValueToCompare="?" Operator="NotEqual"
></asp:CompareValidator>
                            </p>
                        </td>
                        <td>
                        </td>
                        <td>
                            <asp:CustomValidator id="CustomValidator1" runat="server"
ErrorMessage="Précision diplôme invalide" OnServerValidate="CustomValidator1_ServerValidate_1"
EnableViewState="False" ClientValidationFunction="chkAutreDiplome"></asp:CustomValidator>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            Année d'obtention*</td>
                        <td colspan="3">
                            <asp:TextBox id="txtAnDiplome" runat="server" Columns="4"></asp:TextBox>
                            <asp:RangeValidator id="RangeValidator1" runat="server" ErrorMessage="Année
diplôme doit être dans l'intervalle [1990,2004]" ControlToValidate="txtAnDiplome" MinimumValue="1990"
MaximumValue="2004" Type="Integer"></asp:RangeValidator>
```

```
                            <asp:RequiredFieldValidator id="RequiredFieldValidator2" runat="server"
ErrorMessage="Année diplôme requise" ControlToValidate="txtAnDiplome"></asp:RequiredFieldValidator>
                        </td>
                    </tr>
                </tbody>
            </table>
        </fieldset>
        <p>
            <asp:Button id="btnEnvoyer" runat="server" Text="Envoyer"></asp:Button>
        </p>
    </form>
</body>
</html>
```

L'attribut [OnServerValidate] du composant [CustomValidator] permet de préciser la fonction chargée de la validation côté serveur. Ci-dessus le composant [CustomValidator1] fait appel à la procédure [CustomValidator1_ServerValidate_1] suivante :

```
<%@ Page Language="VB" %>
<script runat="server">

    Sub CustomValidator1_ServerValidate_1(sender As Object, e As ServerValidateEventArgs)
        ' le champ [txtAutreDiplome] doit être non vide si [cmbDiplomes]=[autre]
        e.isvalid=not (cmbDiplomes.selecteditem.text="[Autre]" and txtAutreDiplome.text.trim="")
            and not (cmbDiplomes.selecteditem.text<>"[Autre]" and cmbDiplomes.selecteditem.text<>"?" and
txtAutreDiplome.text.trim<>"")
    End Sub

</script>
....
```

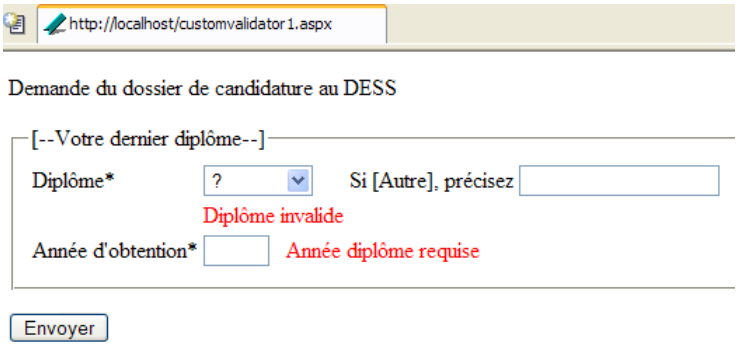
Une fonction de validation associée à un composant [CustomValidator] reçoit deux paramètres :

- **sender** : objet ayant provoqué l'événement
- **e** : l'événement. La procédure doit positionner l'attribut [e.IsValid] à vrai si les données vérifiées sont correctes, à faux sinon.

Ici, les vérifications faites sont les suivantes :

- la liste déroulante [cmbDiplomes] ne peut avoir [?] pour valeur. C'est vérifié par le composant [CompareValidator2]
- l'utilisateur sélectionne un diplôme dans la liste déroulante [cmbDiplomes]. Si son diplôme n'existe pas dans la liste, il a la possibilité de sélectionner l'option [Autre] de la liste. Il doit alors indiquer son diplôme dans le champ de saisie [txtAutreDiplome]. Si [Autre] est sélectionnée dans [cmbDiplomes] alors le champ [txtAutreDiplome] doit être non vide. Si ni [Autre], ni [?] n'est sélectionnée dans [cmbDiplomes] alors le champ [txtAutreDiplome] doit être vide. Ceci est vérifié par la fonction associée au composant [CustomValidator1].
- l'année d'obtention du diplôme doit être dans l'intervalle [1900-2004]. Ceci est vérifié par [RangeValidator1]. Cependant si l'utilisateur ne met rien dans le champ, la fonction de validation de [RangeValidator1] n'est pas utilisée. Aussi ajoute-t-on le composant [RequiredFieldValidator2] pour vérifier la présence d'un contenu. Ceci est une règle générale. Le contenu d'un champ [TextBox] n'est pas vérifié si celui-ci est vide. Le seul cas où il l'est, est celui où il est associé à un composant [RequiredFieldValidator].

Voici un exemple d'exécution dans le navigateur [Mozilla] :



Si nous utilisons le navigateur [IE6], nous pouvons ajouter une fonction de validation côté client pour le composant [CustomValidator1]. Pour cela, ce composant a les propriétés suivantes :

- EnableClientScript=true
- ClientValidationFunction=chkAutreDiplome

La fonction [chkAutreDiplome] est la suivante :

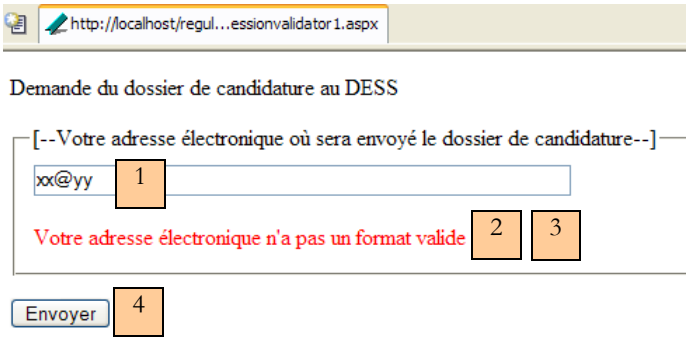
```
<head>
    <meta http-equiv="pragma" content="no-cache" />
    <script language="javascript">
   function chkAutreDiplome(source,args){
     // vérifie la validité du champ txtAutreDiplome
     with(document.frmCandidature){
       diplome=cmbDiplomes.options[cmbDiplomes.selectedIndex].text;
         args.IsValid= !(diplome=="[Autre]" && txtAutreDiplome.value=="")
           && ! (diplome!="[Autre]" && diplome!="?" && txtAutreDiplome.value!="");
     }
   }
   </script>
</head>
```

La fonction [chkAutreDiplome] reçoit les mêmes deux paramètres que la fonction serveur [CustomValidator1_ServerValidate_1] :

- **source** : l'objet qui a provoqué l'événement
- **args** : l'événement. Celui-ci a un attribut [IsValid] qui doit être positionné à vrai par la fonction si les données vérifiées sont correctes. Une valeur à faux affichera à l'endroit du composant de validation le message d'erreur qui lui est associé et le formulaire ne sera alors pas posté au serveur.

2.2.5 RegularExpressionValidator

Nous construisons la page [regularexpressionvalidator1.aspx] suivante :



n°	nom	type	propriétés	rôle
1	txtMel	TextBox	EnableViewState=true	champ de saisie
2	RequiredFieldValidator3	RequiredFieldValidator	ControlToValidate=txtMel Display=Dynamic	composant de validation du contrôle [1]
3	RegularExpressionValidator1	RegularExpressionValidator	ControlToValidate=txtMel Display=Dynamic	composant de validation du contrôle [1]
4	btnEnvoyer	Button	EnableViewState=false CausesValidation=true	bouton [submit]

Ici, nous voulons vérifier le format d'une adresse électronique. Nous le faisons avec un composant [RegularExpressionValidator] qui permet de vérifier la validité d'un champ à l'aide d'une expression régulière. Rappelons-ici qu'une expression régulière est un modèle de chaîne de caractères. On vérifie donc le contenu d'un champ vis à vis d'un modèle. Comme le contenu d'un champ de saisie n'est pas vérifié s'il est vide, il nous faut également un composant [RequiredFieldValidator] pour vérifier que l'adresse électronique n'est pas vide. La classe [RegularExpressionValidator] a des propriétés analogues aux classes des composants déjà étudiés :

ControlToValidate	champ dont la valeur doit être contrôlée par le composant
EnableClientScript	booléen - à vrai indique que le contenu du champ précédent doit être contrôlé également côté client
ErrorMessage	le message d'erreur que le composant doit afficher en cas d'erreur détectée
RegularExpression	l'expression régulière à laquelle le contenu de [ControlToValidate] va être comparé
Display	mode d'affichage : **Static** : le champ est toujours présent même s'il n'affiche pas de message d'erreur, **Dynamic** : le champ n'est présent que s'il y a un message d'erreur, **None** : le msg d'erreur n'est pas affiché. Ce champ existe également pour les autres composants mais il n'avait pas été utilisé jusqu'à maintenant.

L'expression régulière modélisant une adresse électronique pourrait être la suivante : \s*[\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]+)+\s*

Une adresse électronique est de la forme [champ1.champ2....@champA.champB...]. Il y a obligatoirement au moins un champ avant le signe @ et au moins deux champs derrière. Ces champs sont constitués de caractères alphanumériques et du signe -. Le caractère alphanumérique est représenté par le symbole \w. La séquence [\w-] signifie le caractère \w ou le caractère -. Le signe + derrière une séquence S signifie que celle-ci peut se répéter 1 ou plusieurs fois, le signe * signifie qu'elle peut se répéter 0 ou plusieurs fois, le signe ? qu'elle peut se répéter 0 ou 1 fois.

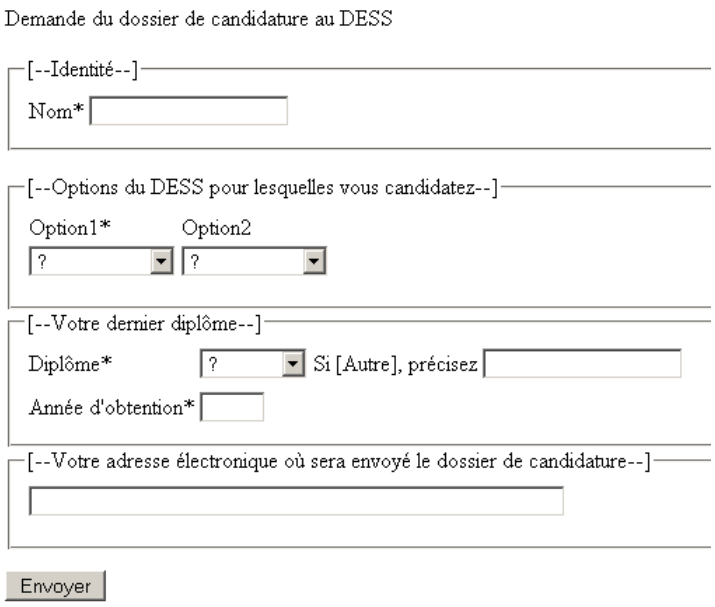
Un champ de l'adresse électronique correspond au modèle [\w-]+. Le fait qu'il y ait obligatoirement au moins un champ avant le signe @ et au moins deux après, séparés par le signe ., correspond au modèle : [\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]+)+. On peut laisser l'utilisateur mettre des espaces devant et après l'adresse. On les enlèvera lors du traitement. Une suite d'espaces éventuellement vide est représentée par le modèle \s*. D'où l'expression régulière de l'adresse électonique : \s*[\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]+)+\s*.

Le code de présentation de la page est alors le suivant :

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p align="left">
            Demande du dossier de candidature au DESS
        </p>
        <fieldset>
            <legend>[--Votre adresse électronique où sera envoyé le dossier de candidature--]</legend>
            <table>
                <tbody>
                    <tr>
                        <td>
                            <asp:TextBox id="txtMel" runat="server" Columns="60"></asp:TextBox>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <p>
                                <asp:RequiredFieldValidator id="RequiredFieldValidator3" runat="server"
Display="Dynamic" ControlToValidate="txtMel" ErrorMessage="Votre adresse électronique est
requise"></asp:RequiredFieldValidator>
                            </p>
                            <p>
                                <asp:RegularExpressionValidator id="RegularExpressionValidator1"
runat="server" Display="Dynamic" ControlToValidate="txtMel" ErrorMessage="Votre adresse électronique n'a
pas un format valide" ValidationExpression="\s*[\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]
+)+\s*"></asp:RegularExpressionValidator>
                            </p>
                        </td>
                    </tr>
                </tbody>
            </table>
        </fieldset>
        <p>
            <asp:Button id="btnEnvoyer" runat="server" Text="Envoyer"></asp:Button>
        </p>
    </form>
</body>
</html>
```

2.2.6 ValidationSummary

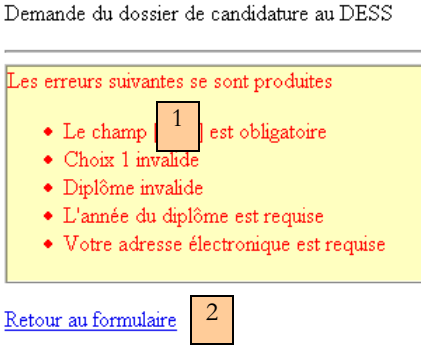
Pour des raisons esthétiques, on peut vouloir rassembler les messages d'erreurs en un unique endroit comme dans l'exemple suivant [summaryvalidator1.aspx] où nous avons réuni dans une unique page tous les exemples précédents :



Tous les contrôles de validation ont les propriétés suivantes :

- [Display=Dynamic] qui fait que les contrôles n'occupent pas d'espace dans la page si leur message d'erreur est vide
- [EnableClientScript=false] pour interdire toute validation côté client
- [Text=*]. Ce sera le message affiché en cas d'erreur, le contenu de l'attribut [ErrorMessage] étant lui affiché par le contrôle [ValidationSummary] que nous présentons ci-dessous.

Cet ensemble de contrôles est placé dans un composant [Panel] appelée [vueFormulaire]. Dans un autre composant [Panel] appelé [vueErreurs], nous plaçons un contrôle [ValidationSummary] :



n°	nom	type	propriétés	rôle
1	`ValidationSummary1`	ValidationSummary	HeaderText=Les erreurs suivantes se sont produites, EnableClientScript=false, ShowSummary=true	affiche les erreurs de tous les contrôles de validation de la page
2	`lnkErreursToFormulaire`	LinkButton	CausesValidation=false	lien de retour au formulaire

Pour le lien [lnkErreursToFormulaire], il n'y a pas lieu d'activer la validation puisque ce lien ne poste aucune valeur à vérifier.

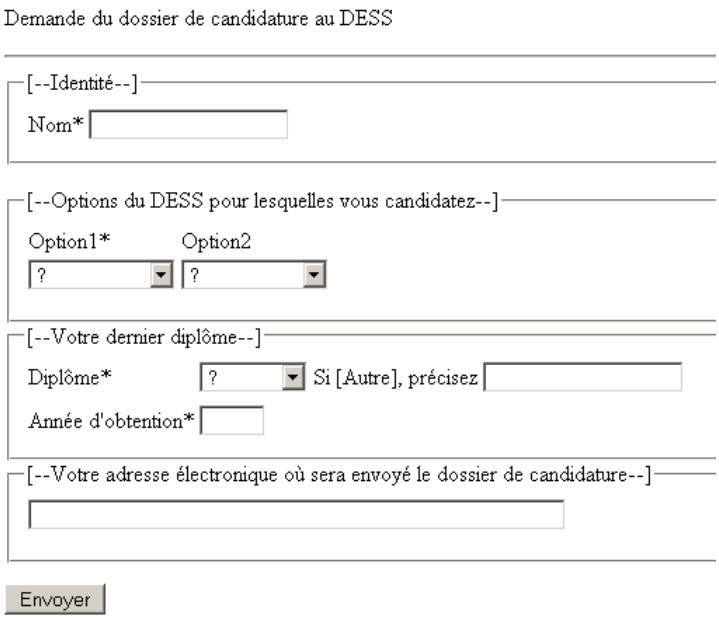
Lorsque toutes les données sont valides, on présente à l'utilisateur la vue [infos] suivante :

n°	nom	type	propriétés	rôle
1	lblInfo	Label		message d'information à destination de l'utilisateur

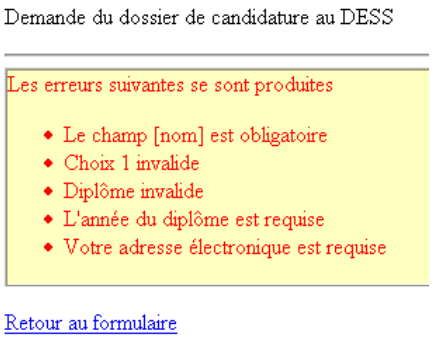
Le code de présentation de cette page est le suivant :

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p align="left">
            Demande du dossier de candidature au DESS
        </p>
        <p>
            <hr />
            <asp:panel id="vueErreurs" runat="server">
                <p align="left">
                    <asp:ValidationSummary id="ValidationSummary1" runat="server" ShowMessageBox="True"
BorderColor="#C04000" BorderWidth="1px" BackColor="#FFFFC0" HeaderText="Les erreurs suivantes se sont
produites"></asp:ValidationSummary>
                </p>
                <p>
                    <asp:LinkButton id="lnkErreursToFormulaire" onclick="lnkErreursToFormulaire_Click"
runat="server" CausesValidation="False">Retour au formulaire</asp:LinkButton>
                </p>
            </asp:panel>
            <asp:panel id="vueFormulaire" runat="server">
....
                    <asp:Button id="btnEnvoyer" onclick="btnEnvoyer_Click" runat="server"
Text="Envoyer"></asp:Button>
                </p>
            </asp:panel>
        <asp:panel id="vueInfos" runat="server">
            <asp:Label id="lblInfo" runat="server"></asp:Label>
        </asp:panel>
    </form>
</body>
</html>
```

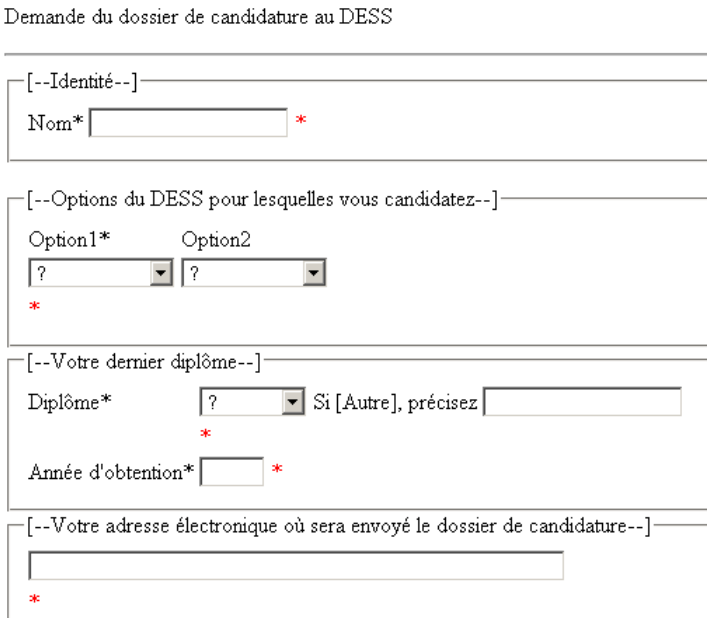
Voici quelques exemples de résultats obtenus. Nous validons la vue [formulaire] sans saisir de valeurs :



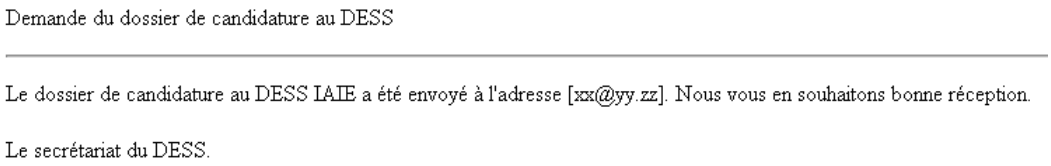
Le bouton [Envoyer] nous donne la réponse suivante :



C'est la vue [erreurs] qui a été affichée. Si nous utilisons le lien de retour au formulaire, nous retrouvons celui-ci dans l'état suivant :



C'est la vue [formulaire]. On remarquera le caractère [*] près des données erronées. C'est le champ [Text] des contrôles de validation qui a été affiché. Si nous remplissons correctement les champs, nous obtiendrons la vue [infos] :



Le code de contrôle de la page est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">

    ' procédure exécutée au chargement de la page
    Sub page_Load(sender As Object, e As EventArgs)
        ' à la 1ère requête, on présente la vue [formulaire]
        if not ispostback then
            afficheVues(true,false,false)
        end if
    end sub

    sub afficheVues(byval formulaireVisible as boolean, _
        erreursVisible as boolean, infosVisible as boolean)
        ' jeu de vues
        vueFormulaire.visible=formulaireVisible
        vueErreurs.visible=erreursVisible
        vueInfos.visible=infosVisible
    end sub
```

```
    Sub CustomValidator1_ServerValidate(sender As Object, e As ServerValidateEventArgs)
...
    End Sub

    Sub CustomValidator2_ServerValidate(sender As Object, e As ServerValidateEventArgs)
...
    End Sub

    Sub CustomValidator1_ServerValidate_1(sender As Object, e As ServerValidateEventArgs)
...
    End Sub

    Sub lnkErreursToFormulaire_Click(sender As Object, e As EventArgs)
        ' affiche la vue formulaire
        afficheVues(true,false,false)
        ' refait les contrôles de validité
        Page.validate
    End Sub

    Sub btnEnvoyer_Click(sender As Object, e As EventArgs)
        ' la page est-elle valide ?
        if not Page.IsValid then
            ' on affiche la vue [erreurs]
            afficheVues(false,true,false)
        else
            ' sinon la vue [infos]
            lblInfo.Text="Le dossier de candidature au DESS IAIE a été envoyé à l'adresse ["+ _
                        txtMel.Text + "]. Nous vous en souhaitons bonne réception.<br><br>Le secrétariat
du DESS."
            afficheVues(false,false,true)
        end if
    end sub

</script>
```

```
<html>
...
</html>
```

- Dans la procédure [Page_Load] exécutée à chaque requête du client, nous affichons la vue [formulaire], les autres étant cachées. Ceci est fait seulement lors de la première requête. La procédure fait appel à une procédure utilitaire [afficheVues] à qui on passe trois booléens pour faire afficher ou non les trois vues.
- Lorsque la procédure [btnEnvoyer_Click] est appelée, les vérifications de données ont été faites. Le bouton [btnEnvoyer] a la propriété [CausesValidation=true] qui force cette vérification des données. Tous les contrôles de validation ont été exécutés et leur propriété [IsValid] positionnée. Celle-ci indique si la donnée vérifiée par le contrôle était valide ou non. Par ailleurs, la propriété [IsValid] de la page elle-même a été également positionnée. Elle est à [vrai] uniquement si la propriété [IsValid] de tous les contrôles de validation de la page ont la valeur [vrai]. Ainsi la procédure [btnEnvoyer_Click] commence-t-elle par vérifier si la page est valide ou non. Si elle est invalide, on fait afficher la vue [erreurs]. Celle-ci contient le contrôle [ValidationSummary] qui reprend les attributs [ErrorMessage] de tous les contrôles (voir copie d'écran plus haut). Si la page est valide, c'est la vue [infos] qui est affichée avec un message d'information.
- La procédure [lnkErreursToFormulaire_Click] est chargée de faire afficher la vue [formulaire] dans l'état où elle a été validée. Tous les champs de saisie de la vue [formulaire] ayant la propriété [EnableViewState=true], leur état est automatiquement régénéré. Curieusement, l'état des composants de validation n'est lui pas restitué. On pouvait s'attendre en effet, au retour de la vue [erreurs] à voir les contrôles erronés afficher leur champ [Text]. Ce n'est pas le cas. On a donc forcé la validation des données en utilisant la méthode [Page.Validate] de la page. Ceci doit être fait une fois que le panel [vueFormulaire] a été rendu visible. Il y a donc au total deux validations. Ceci serait à éviter dans la pratique. Ici, l'exemple nous permettait d'introduire de nouvelles notions sur la validation de page.

2.3 Composants ListControl et liaison de données

Un certain nombre des composants serveur étudiés permettent d'afficher une liste de valeurs (DropDownList, ListBox). D'autres que nous n'avons pas encore présentés permettent d'afficher **plusieurs** listes de valeurs dans des tables HTML. Pour tous, il est possible, par programme, d'associer une par une les valeurs des listes qui sont des objets de type String au composant associé. Il est possible également d'associer des objets plus complexes à ces composants tels des objets de type [Array], [ArrayList], [DataSet], [HashTable], ... ce qui simplifie le code qui associe les données au composant. On appelle cette association, une **liaison de données**.

Tous les composants dérivés de la classe [ListControl] peuvent être associés à une liste de données. Il s'agit des composants [DropDownList], [ListBox], [CheckButtonList], [RadioButtonList]. La source de données associée à ces composants peut être diverse : [Array], [ArrayList], [DataTable], [DataSet], [HashTable],...., de façon générale un objet implémentant l'une des interfaces IEnumerable, ICollection, IListSource. La source de données affecte les propriétés [Text] et [Value] de chacun des membres [Item]

de l'objet [ListControl]. Si T est la valeur de [Text] et V la valeur de [Value], la balise HTML générée pour chaque élément de [ListControl] est la suivante :

`DropDownList, ListBox`	<option value="**V**">**T**</option>
`CheckButtonList`	<input type="checkbox" value="**V**">**T**
`RadioButtonList`	<input type="radio" value="**V**">**T**

L'association d'un composant [ListControl] à une source de données se fait au travers des propriétés suivantes :

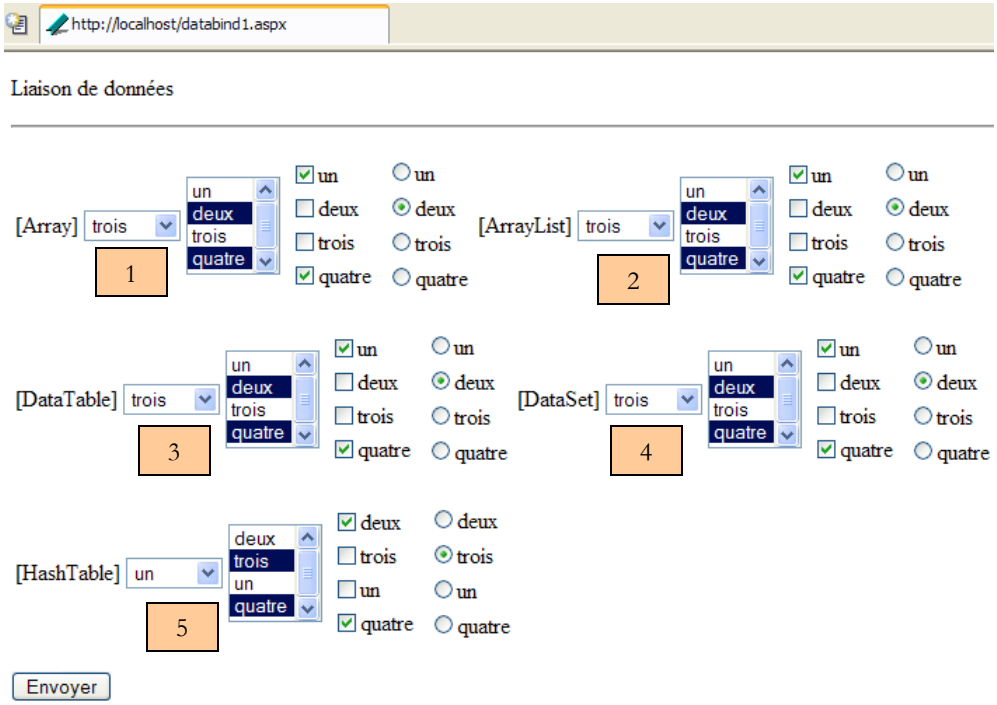
`DataSource`	une source de données [Array], [ArrayList], [DataTable], [DataSet], [HashTable], ...
`DataMember`	dans le cas où la source de données est un [DataSet], représente le nom de la table à utiliser comme source de données. La véritable source de données est alors une table.
`DataTextField`	dans le cas où la source de données est une table ([DataTable],[DataSet]), représente le nom de la colonne de la table qui va donner ses valeurs au champ [Text] des éléments du [ListControl]
`DataValueField`	dans le cas où la source de données est une table ([DataTable],[DataSet]), représente le nom de la colonne de la table qui va donner ses valeurs au champ [Value] des éléments du [ListControl]

Selon la nature de la source de données, l'association de celle-ci à un composant [ListControl] se fera de façon différente :

`Array A`	[ListControl].DataSource=A les champs [Text] et [Value] des éléments de [ListControl] auront pour valeurs les éléments de A
`ArrayList AL`	[ListControl].DataSource=AL les champs [Text] et [Value] des éléments de [ListControl] auront pour valeurs les éléments de AL
`DataTable DT`	[ListControl].DataSource=DT, [ListControl].DataTextField="col1", [ListControl]. DatavalueField ="col2" où col1 et col2 sont deux colonnes de la table DT. Les champs [Text] et [Value] des éléments de [ListControl] auront pour valeurs celles des colonnes col1 et col2 de la table DT
`DataSet DS`	[ListControl].DataSource=DS , [ListControl].DataSource="table" où "table" est le nom d'une des tables de DS. [ListControl].DataTextField="col1", [ListControl]. DatavalueField ="col2" où col1 et col2 sont deux colonnes de la table "table". Les champs [Text] et [Value] des éléments de [ListControl] auront pour valeurs celles des colonnes col1 et col2 de la table "table"
`HashTable HT`	[ListControl].DataSource=HT , [ListControl].DataTextField="key", [ListControl]. DatavalueField ="value" où [key] et [value] sont respectivement les clés et les valeurs de HT.

L'association d'un composant [ListControl] à une source de données n'initialise pas le composant avec les valeurs de la source de données. C'est l'opération **[ListControl].DataBind** qui le fait.

Nous mettons en pratique ces informations sur l'exemple [databind1.aspx] suivant :



Nous avons là cinq liaisons de données avec pour chacune d'elles quatre contrôles de type [DropDownList], [ListBox], [CheckBoxList] et [RadioButtonList]. Les cinq liaisons étudiées diffèrent par leurs sources de données :

liaison	source de données
1	Array
2	ArrayList
3	DataTable
4	DataSet
5	HashTable

2.3.1 Code de présentation des composants

Le code de présentation des contrôles de la liaison 1 est le suivant :

```
<td>
  <asp:DropDownList id="DropDownList1" runat="server"></asp:DropDownList>
</td>
<td>
  <asp:ListBox id="ListBox1" runat="server" SelectionMode="Multiple"></asp:ListBox>
</td>
<td>
  <asp:CheckBoxList id="CheckBoxList1" runat="server"></asp:CheckBoxList>
</td>
<td>
  <asp:RadioButtonList id="RadioButtonList1" runat="server"></asp:RadioButtonList>
</td>
```

Celui des liaisons 2 à 5 est identique au numéro de liaison près.

2.3.2 Liaison à une source de données de type Array

La liaison des quatre contrôles [ListBox] ci-dessus se fait de la façon suivante dans la procédure [Page_Load] du code de contrôle :

```
' les données globales
dim textes() as string={"un","deux","trois","quatre"}
dim valeurs() as string={"1","2","3","4"}
dim myDataListe as new ArrayList
dim myDataTable as new DataTable("table1")
dim myDataSet as new DataSet
dim myHashTable as new HashTable

' procédure exécutée au chargement de la page
Sub page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
```

```
      ' on crée les sources  des données qu'on va lier aux composants
      createDataSources
      ' liaison à un tableau [Array]
      bindToArray
      ' liaison à une liste [ArrayList]
      bindToArrayList
      ' liaison à une table [DataTable]
      bindToDataTable
      ' liaison à un groupe de données [DataSet]
      bindToDataSet
      ' liaison à un dictionnaire [HashTable]
      bindToHashTable
    end if
End Sub

sub createDataSources
  ' crée les sources de données qui seront liées aux composants
  ' arraylist
  dim i as integer
  for i=0 to textes.length-1
    myDataListe.add(textes(i))
  next
  ' datatable
  ' on définit ses deux colonnes
  myDataTable.Columns.Add("id",Type.GetType("System.Int32"))
  myDataTable.Columns.Add("texte",Type.GetType("System.String"))
  ' on remplit la table
  dim ligne as DataRow
  for i=0 to textes.length-1
    ligne=myDataTable.NewRow
    ligne("id")=i
    ligne("texte")=textes(i)
    myDataTable.Rows.Add(ligne)
  next
  ' dataset - une seule table
  myDataSet.Tables.Add(myDataTable)
  ' hashtable
  for i=0 to textes.length-1
    myHashTable.add(valeurs(i),textes(i))
  next
end sub
```

```
' liaison tableau
sub bindToArray
        ' l'association aux composants
        with DropDownList1
            .DataSource=textes
            .DataBind
        end with
        with ListBox1
            .DataSource=textes
            .DataBind
        end with
        with CheckBoxList1
            .DataSource=textes
            .DataBind
        end with
        with RadioButtonList1
            .DataSource=textes
            .DataBind
        end with
        ' la sélection des éléments
        ListBox1.Items(1).Selected=true
        ListBox1.Items(3).Selected=true
        CheckBoxList1.Items(0).Selected=true
        CheckBoxList1.Items(3).Selected=true
        DropDownList1.SelectedIndex=2
        RadioButtonList1.SelectedIndex=1
end sub
```

```
sub bindToArrayList
....
end sub

sub bindToDataTable
...
end sub

sub bindToHashTable
...
end sub
```

```
sub bindToDataSet
...
end sub
```

La liaison des contrôles aux données ne se fait ici qu'à la première requête. Ensuite les contrôles garderont leurs éléments par le mécanisme du [VIEWSTATE]. La liaison est faite dans la procédure [bindToArray]. La source de données étant de type [Array], seul le champ [DataSource] des composants [ListControl] est initialisé. Le remplissage du contrôle avec les valeurs de la source de données associée est réalisée par la méthode [ListControl].DataBind. Ce n'est qu'après, que les objets [ListControl] ont des éléments. On peut alors sélectionner certains d'entre-eux.

2.3.3 Liaison à une source de données de type ArrayList

La source de données [myDataListe] est initialisée dans la procédure [createDataSources] :

```
' les données globales
dim textes() as string={"un","deux","trois","quatre"}
dim myDataListe as new ArrayList
..

' procédure exécutée au chargement de la page
Sub page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
      ' on crée les sources  des données qu'on va lier aux composants
      createDataSources
...
    end if
End Sub

sub createDataSources
  ' crée les sources de données qui seront liées aux composants
  ' arraylist
  dim i as integer
  for i=0 to textes.length-1
    myDataListe.add(textes(i))
  next
...
end sub
```

La liaison des quatre contrôles [ListBox] de la liaison 2 se fait de la façon suivante dans la procédure [bindToArrayList] du code de contrôle :

```
' liaison arraylist
sub bindToArrayList
        ' l'association aux composants
        with DropDownList2
            .DataSource=myDataListe
            .DataBind
        end with
        with ListBox2
            .DataSource=myDataListe
            .DataBind
        end with
        with CheckBoxList2
            .DataSource=myDataListe
            .DataBind
        end with
        with RadioButtonList2
            .DataSource=myDataListe
            .DataBind
        end with
        ' la sélection des éléments
        ListBox2.Items(1).Selected=true
        ListBox2.Items(3).Selected=true
        CheckBoxList2.Items(0).Selected=true
        CheckBoxList2.Items(3).Selected=true
        DropDownList2.SelectedIndex=2
        RadioButtonList2.SelectedIndex=1
end sub
```

La source de données étant de type [ArrayList], seul le champ [DataSource] des composants [ListControl] est initialisé.

2.3.4 Source de données de type DataTable

La source de données [myDataTable] est initialisée dans la procédure [createDataSources] :

```
' les données globales
dim textes() as string={"un","deux","trois","quatre"}
dim myDataTable as new DataTable("table1")
...

' procédure exécutée au chargement de la page
Sub page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
      ' on crée les sources  des données qu'on va lier aux composants
      createDataSources
...
    end if
End Sub

sub createDataSources
  ' crée les sources de données qui seront liées aux composants
...
  ' datatable
  ' on définit ses deux colonnes
  myDataTable.Columns.Add("id",Type.GetType("System.Int32"))
  myDataTable.Columns.Add("texte",Type.GetType("System.String"))
  ' on remplit la table
  dim ligne as DataRow
  for i=0 to textes.length-1
    ligne=myDataTable.NewRow
    ligne("id")=i
    ligne("texte")=textes(i)
    myDataTable.Rows.Add(ligne)
  next
...
end sub
```

On commence par construire un objet [DataTable] avec deux colonnes [id] et [texte]. La colonne [id] alimentera le champ [Value] des éléments des [ListControl] et la colonne [texte] leurs champs [Text]. La construction du [DataTable] avec deux colonnes se fait de la façon suivante :

```
  myDataTable.Columns.Add("id",Type.GetType("System.Int32"))
  myDataTable.Columns.Add("texte",Type.GetType("System.String"))
```

On crée donc une table à deux colonnes :

- la première appelée "id" est de type entier
- la seconde appelée "texte" est de type chaîne de caractères

La structure de la table étant créée, on peut la remplir avec le code suivant :

```
  dim ligne as DataRow
  for i=0 to textes.length-1
    ligne=myDataTable.NewRow
    ligne("id")=i
    ligne("texte")=textes(i)
    myDataTable.Rows.Add(ligne)
  next
```

La colonne [id] va contenir des entiers [0,1,..,n] alors que la colonne [texte] va contenir les valeurs du tableau [textes]. Ceci fait, la table [dataListe] est remplie. La liaison des quatre contrôles [ListBox] de la liaison 3 se fait de la façon suivante dans la procédure [bindToDataTable] du code de contrôle :

```
' liaison datatable
sub bindToDataTable
        ' l'association aux composants
        with DropDownList3
            .DataSource=myDataTable
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        with ListBox3
            .DataSource=myDataTable
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        with CheckBoxList3
            .DataSource=myDataTable
            .DataValueField="id"
            .DataTextField="texte"
```

```
            .DataBind
        end with
        with RadioButtonList3
            .DataSource=myDataTable
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        ' la sélection des éléments
        ListBox3.Items(1).Selected=true
        ListBox3.Items(3).Selected=true
        CheckBoxList3.Items(0).Selected=true
        CheckBoxList3.Items(3).Selected=true
        DropDownList3.SelectedIndex=2
        RadioButtonList3.SelectedIndex=1
end sub
```

Chaque composant [ListControl] est lié à la source [myDataTable] en affectant pour chacun d'entre-eux :

```
            .DataSource= myDataTable
            .DataValueField="id"
            .DataTextField="texte"
```

La table [`myDataTable`] est la source de données. La colonne [id] de cette table va alimenter les champs [Value] des éléments des composants alors que la colonne [texte] va alimenter leurs champs [Text].

2.3.5 Source de données de type DataSet

La source de données [myDataSet] est initialisée dans la procédure [createDataSources] :

```
' les données globales
dim myDataTable as new DataTable("table1")
dim myDataSet as new DataSet
...


' procédure exécutée au chargement de la page
Sub page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
      ' on crée les sources  des données qu'on va lier aux composants
      createDataSources
...
    end if
End Sub

sub createDataSources
  ' crée les sources de données qui seront liées aux composants
  ' dataset - une seule table
  myDataSet.Tables.Add(myDataTable)
...
end sub
```

Un objet [DataSet] représente un ensemble de tables de type [DataTable]. On ajoute la table [myDataTable] construite précédemment au [DataSet]. La liaison des quatre contrôles [ListBox] de la liaison 4 se fait de la façon suivante dans la procédure [bindToDataSet] du code de contrôle :

```
' liaison dataset
sub bindToDataSet
        ' l'association aux composants
        with DropDownList4
            .DataSource=myDataSet
            .DataMember="table1"
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        with ListBox4
            .DataSource=myDataSet
            .DataMember="table1"
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        with CheckBoxList4
            .DataSource=myDataSet
            .DataMember="table1"
            .DataValueField="id"
```

```
            .DataTextField="texte"
            .DataBind
        end with
        with RadioButtonList4
            .DataSource=myDataSet
            .DataMember="table1"
            .DataValueField="id"
            .DataTextField="texte"
            .DataBind
        end with
        ' la sélection des éléments
        ListBox4.Items(1).Selected=true
        ListBox4.Items(3).Selected=true
        CheckBoxList4.Items(0).Selected=true
        CheckBoxList4.Items(3).Selected=true
        DropDownList4.SelectedIndex=2
        RadioButtonList4.SelectedIndex=1
end sub
```

Chaque composant [ListControl] est lié à la source de données de la façon suivante :

```
            .DataSource= myDataSet
            .DataMember="table1"
            .DataValueField="id"
            .DataTextField="texte"
```

Le groupe de données [myDataSet] est la source de données. Comme celle-ci peut comprendre plusieurs tables, on précise dans [DataMember] le nom de celle que l'on va utiliser. La colonne [id] de cette table va alimenter les champs [Value] des éléments des composants alors que la colonne [texte] va alimenter leurs champs [Text].

2.3.6 Source de données de type HashTable

La source de données [myHashTable] est initialisée dans la procédure [createDataSources] :

```
' les données globales
dim textes() as string={"un","deux","trois","quatre"}
dim valeurs() as string={"1","2","3","4"}
dim myHashTable as new HashTable
...

' procédure exécutée au chargement de la page
Sub page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
      ' on crée les sources  des données qu'on va lier aux composants
      createDataSources
...
    end if
End Sub

sub createDataSources
  ' crée les sources de données qui seront liées aux composants
...
  ' hashtable
  for i=0 to textes.length-1
    myHashTable.add(valeurs(i),textes(i))
  next
end sub
```

Le dictionnaire [myHashTable] peut être vu comme une table à deux colonnes appelées "key" et "value". La colonne [key] représente les clés du dictionnaire et la colonne [value] les valeurs associées à celles-ci. La colonne [key] est ici formée du contenu du tableau [valeurs] et la colonne [value] du contenu du tableau [textes]. La liaison de cette source aux contrôles est faite dans la procédure [bindToHashTable] :

```
sub bindToHashTable
        ' l'association aux composants
        with DropDownList5
            .DataSource=myHashTable
            .DataValueField="key"
            .DataTextField="value"
            .DataBind
        end with
        with ListBox5
            .DataSource=myHashTable
            .DataValueField="key"
            .DataTextField="value"
            .DataBind
```

```
        end with
        with CheckBoxList5
            .DataSource=myHashTable
            .DataValueField="key"
            .DataTextField="value"
            .DataBind
        end with
        with RadioButtonList5
            .DataSource=myHashTable
            .DataValueField="key"
            .DataTextField="value"
            .DataBind
        end with
        ' la sélection des éléments
        ListBox5.Items(1).Selected=true
        ListBox5.Items(3).Selected=true
        CheckBoxList5.Items(0).Selected=true
        CheckBoxList5.Items(3).Selected=true
        DropDownList5.SelectedIndex=2
        RadioButtonList5.SelectedIndex=1
end sub
```

Pour chacun des composants, la liaison est faite par les instructions :

```
            .DataSource=myHashTable
            .DataValueField="key"
            .DataTextField="value"
```

La source de données est le dictionnaire [myHashTable]. Les valeurs du contrôle sont fournies par la colonne [key] du dictionnaire et les textes par la colonne [value]. L'insertion des éléments du dictionnaire dans les contrôles se fait dans l'ordre des clés qui est à priori aléatoire.

2.3.7 Les directives d'importation d'espaces de noms

Un certain nombre d'espaces de noms sont automatiquement importés dans une page ASP.NET. Ce n'est pas le cas de "System.Data" où se trouve les classes [DataTable] et [DataSet]. Aussi faut-il importer cette classe. Cela se fait de la façon suivante :

```
<%@ Page Language="VB" %>
<%@ import Namespace="System.Data" %>
<script runat="server">
...
</script>
<html>
...
</html>
```

2.4 Composant DataGrid et liaison de données

Le composant [DataGrid] permet d'afficher des données sous forme de tableaux mais il va bien au-delà de ce simple affichage :

- il offre la possibilité de paramétrer de façon précise le "rendu visuel" du tableau
- il permet la mise à jour de la source de données

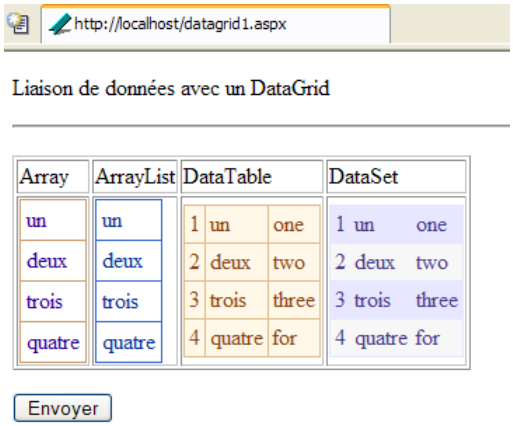
Le composant [DataGrid] est un composant à la fois puissant et complexe. Nous allons le présenter par touches successives.

2.4.1 Affichage d'une source de données Array, ArrayList, DataTable, DataSet

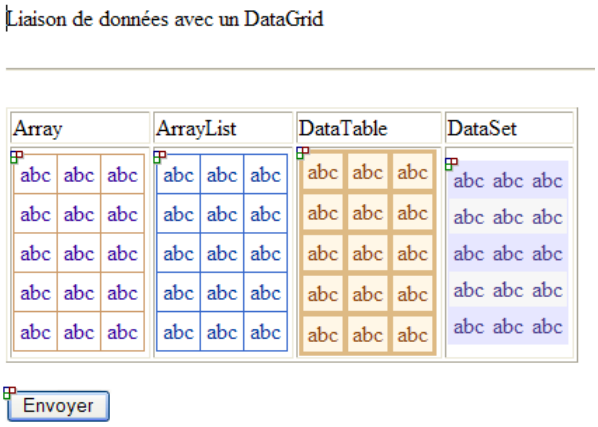
Le composant [DataGrid] permet d'afficher dans un tableau HTML des sources de données de type [Array], [ArrayList], [DataTable], [DataSet]. Pour ces quatre type de données, il suffit d'associer la source à la propriété [DataSource] du composant [DataGrid] :

`DataSource`	une source de données [Array], [ArrayList], [DataTable], [DataSet], ...
`DataMember`	dans le cas où la source de données est un [DataSet], représente le nom de la table à utiliser comme source de données. La véritable source de données est alors une table. Si ce champ n'est pas renseigné, toutes les tables du [DataSet] sont affichées.

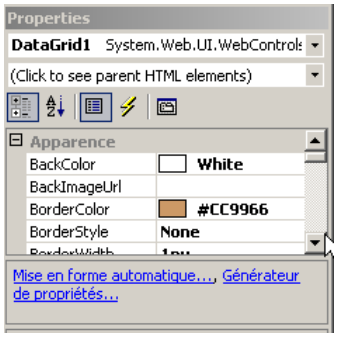
Nous présentons maintenant la page [datagrid1.aspx] qui montre l'association d'un [DataGrid] à quatre sources de données différentes :



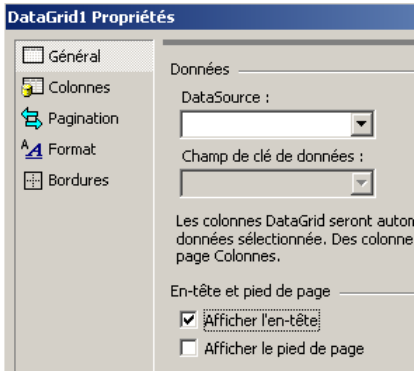
La page contient quatre composants [DataGrid] construits avec [WebMatrix] de la façon suivante. On dépose le composant à son emplacement dans l'onglet [Design] :



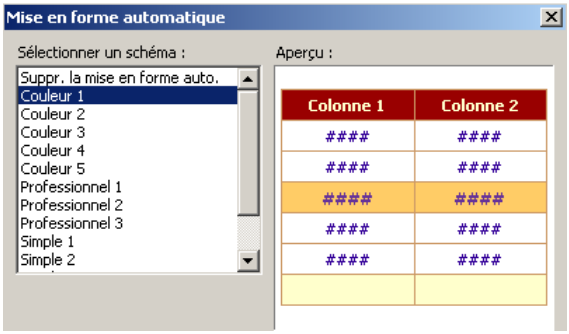
Un tableau HTML générique est alors dessiné. Les propriétés d'un [DataGrid] peuvent être définies à la conception. C'est ce que nous faisons ici pour ses propriétés de mise en forme. Pour cela, nous sélectionnons le [DataGrid] à configurer. Ses propriétés apparaissent dans une fenêtre en bas à droite :



Nous allons utiliser les deux liens ci-dessus. Le lien [Générateur de propriétés] donne accès aux principales propriétés du [DataGrid] :



Nous décochons la phrase [Afficher l'en-tête] pour les quatre composants [DataGrid] et validons la page. L'autre lien [Mise en forme automatique] permet de choisir parmi plusieurs styles pour le tableau HTML qui sera affiché :



Nous choisissons [couleur i] pour le [DataGrid] n° i. Ces choix de conception sont traduits dans le code de présentation de la page :

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            Liaison de données avec un DataGrid
        </p>
        <hr />
        <p>
            <table>
                <tbody>
                </tbody>
            </table>
            <table border="1">
                <tbody>
                    <tr>
                        <td>
                            Array</td>
                        <td>
                            ArrayList</td>
                        <td>
                            DataTable</td>
                        <td>
                            DataSet</td>
                    </tr>
                    <tr>
                        <td>
                            <asp:DataGrid id="DataGrid1" runat="server" ShowHeader="False"
CellPadding="4" BackColor="White" BorderColor="#CC9966" BorderWidth="1px" BorderStyle="None">
                                <FooterStyle forecolor="#330099" backcolor="#FFFFCC"></FooterStyle>
                                <HeaderStyle font-bold="True" forecolor="#FFFFCC"
backcolor="#990000"></HeaderStyle>
                                <PagerStyle horizontalalign="Center" forecolor="#330099"
backcolor="#FFFFCC"></PagerStyle>
                                <SelectedItemStyle font-bold="True" forecolor="#663399"
backcolor="#FFCC66"></SelectedItemStyle>
                                <ItemStyle forecolor="#330099" backcolor="White"></ItemStyle>
                            </asp:DataGrid>
                        </td>
                        <td>
```

```
                            <asp:DataGrid id="DataGrid2" runat="server" ShowHeader="False"
CellPadding="4" BackColor="White" BorderColor="#3366CC" BorderWidth="1px" BorderStyle="None">
                                <FooterStyle forecolor="#003399" backcolor="#99CCCC"></FooterStyle>
                                <HeaderStyle font-bold="True" forecolor="#CCCCFF"
backcolor="#003399"></HeaderStyle>
                                <PagerStyle horizontalalign="Left" forecolor="#003399"
backcolor="#99CCCC" mode="NumericPages"></PagerStyle>
                                <SelectedItemStyle font-bold="True" forecolor="#CCFF99"
backcolor="#009999"></SelectedItemStyle>
                                <ItemStyle forecolor="#003399" backcolor="White"></ItemStyle>
                            </asp:DataGrid>
                        </td>
                        <td>
                            <asp:DataGrid id="DataGrid3" runat="server" ShowHeader="False"
CellPadding="3" BackColor="#DEBA84" BorderColor="#DEBA84" BorderWidth="1px" BorderStyle="None"
CellSpacing="2">
                                <FooterStyle forecolor="#8C4510" backcolor="#F7DFB5"></FooterStyle>
                                <HeaderStyle font-bold="True" forecolor="White"
backcolor="#A55129"></HeaderStyle>
                                <PagerStyle horizontalalign="Center" forecolor="#8C4510"
mode="NumericPages"></PagerStyle>
                                <SelectedItemStyle font-bold="True" forecolor="White"
backcolor="#738A9C"></SelectedItemStyle>
                                <ItemStyle forecolor="#8C4510" backcolor="#FFF7E7"></ItemStyle>
                            </asp:DataGrid>
                        </td>
                        <td>
                            <asp:DataGrid id="DataGrid4" runat="server" ShowHeader="False"
CellPadding="3" BackColor="White" BorderColor="#E7E7FF" BorderWidth="1px" BorderStyle="None"
GridLines="Horizontal">
                                <FooterStyle forecolor="#4A3C8C" backcolor="#B5C7DE"></FooterStyle>
                                <HeaderStyle font-bold="True" forecolor="#F7F7F7"
backcolor="#4A3C8C"></HeaderStyle>
                                <PagerStyle horizontalalign="Right" forecolor="#4A3C8C"
backcolor="#E7E7FF" mode="NumericPages"></PagerStyle>
                                <SelectedItemStyle font-bold="True" forecolor="#F7F7F7"
backcolor="#738A9C"></SelectedItemStyle>
                                <AlternatingItemStyle backcolor="#F7F7F7"></AlternatingItemStyle>
                                <ItemStyle forecolor="#4A3C8C" backcolor="#E7E7FF"></ItemStyle>
                            </asp:DataGrid>
                        </td>
                    </tr>
                </tbody>
            </table>
        </p>
        <asp:Button id="Button1" runat="server" Text="Envoyer"></asp:Button>
    </form>
</body>
</html>
```

En mode conception, nous n'avons fixé que des propriétés de mise en forme. C'est dans le code de contrôle que nous associons des données aux quatre composants :

```
<%@ Page Language="VB" %>
<%@ import Namespace="system.data" %>
<script runat="server">

    ' les données globales
        dim textes1() as string={"un","deux","trois","quatre"}
        dim textes2() as string={"one","two","three","for"}
        dim valeurs() as string={"1","2","3","4"}
        dim myDataListe as new ArrayList
        dim myDataTable as new DataTable("table1")
        dim myDataSet as new DataSet

        ' procédure exécutée au chargement de la page
        Sub page_Load(sender As Object, e As EventArgs)
            if not IsPostBack then
              ' on crée les sources  des données qu'on va lier aux composants
              createDataSources
              ' liaison à un tableau [Array]
              bindToArray
              ' liaison à une liste [ArrayList]
              bindToArrayList
              ' liaison à une table [DataTable]
              bindToDataTable
              ' liaison à un groupe de données [DataSet]
              bindToDataSet
            end if
```

```
        End Sub

        sub createDataSources
          ' crée les sources de données qui seront liées aux composants
          ' arraylist
          dim i as integer
          for i=0 to textes1.length-1
            myDataListe.add(textes1(i))
          next
          ' datatable
          ' on définit ses deux colonnes
          myDataTable.Columns.Add("id",Type.GetType("System.Int32"))
          myDataTable.Columns.Add("texte1",Type.GetType("System.String"))
          myDataTable.Columns.Add("texte2",Type.GetType("System.String"))
          ' on remplit la table
          dim ligne as DataRow
          for i=0 to textes1.length-1
            ligne=myDataTable.NewRow
            ligne("id")=valeurs(i)
            ligne("texte1")=textes1(i)
            ligne("texte2")=textes2(i)
            myDataTable.Rows.Add(ligne)
          next
          ' dataset - une seule table
          myDataSet.Tables.Add(myDataTable)
        end sub

        ' liaison tableau
        sub bindToArray
          with DataGrid1
            .DataSource=textes1
            .DataBind
          end with
        end sub

        ' liaison arraylist
        sub bindToArrayList
          with DataGrid2
            .DataSource=myDataListe
            .DataBind
          end with
        end sub

        ' liaison datatable
        sub bindToDataTable
          with DataGrid3
            .DataSource=myDataTable
            .DataBind
          end with
        end sub

        ' liaison dataset
        sub bindToDataSet
          with DataGrid4
            .DataSource=myDataSet
            .DataBind
          end with
        end sub

</script>
<html>
...
</html>
```

Le code est assez semblable à celui de l'exemple précédent, aussi ne le commenterons-nous pas particulièrement. Remarquons cependant comment la liaison de données est faite. Pour chacun des quatre contrôles, la séquence suivante suffit :

```
          with [DataGrid]
            .DataSource=[source de données]
            .DataBind
          end with
```

où [source de données] est de type [Array] ou [ArrayList] ou [DataTable] ou [DataSet]. On voit donc qu'on a là, un outil puissant d'affichage de données en tableaux :

- la mise en forme est faite avec [WebMatrix] ou tout autre IDE, au moment de la conception
- la liaison de données est faite dans le code. Avec [WebMatrix], elle peut être faite au moment de la conception si la source de données est une base SQL server ou une base ACCESS. Dans ce cas, on placera un composant de type [SqlDataSource] ou [AccessDataSource] sur la feuille. Ce composant peut être relié au moment de la conception à la

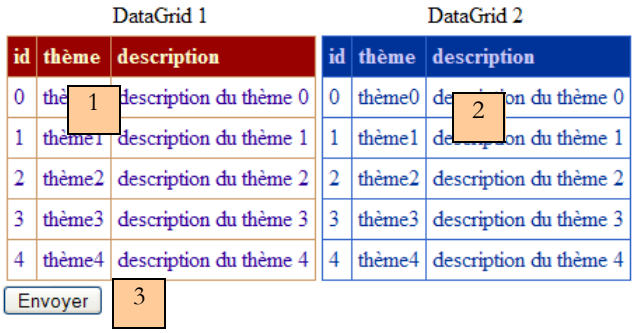
source physique des données, une base SQL Server ou une base ACCESS selon les cas. Si on affecte à la propriété [DataSource] d'un composant [DataGrid] un objet [SqlDataSource] ou [AccessDataSource] relié à une source physique, alors on verra apparaître, en mode conception, les données réelles dans le composant [DataGrid].

2.5 ViewState des composants listes de données

Nous nous proposons ici de mettre en lumière le mécanisme du [VIEWSATE] pour les composants listes de données. On peut hésiter entre deux méthodes pour maintenir l'état d'un composant liste de données entre deux requêtes du client :

- mettre son attribut [VIEWSTATE] à vrai
- mettre son attribut [VIEWSTATE] à faux et mémoriser sa source de données dans la session afin d'être capable de relier le composant à cette source lors de la prochaine requête.

L'exemple suivant illustre certains aspects du mécanisme du [VIEWSTATE] des conteneurs de données. A la première requête du client, l'application présente la vue suivante :



n°	nom	type	propriétés	rôle
1	DataGrid1	DataGrid	EnableViewState=true	affiche une source de données S
2	DataGrid2	DataGrid	EnableViewState=true	affiche la même source S que [DataGrid1]
3	Button1	Button	EnableViewState=false	bouton [submit]

Le composant [DataGrid1] est maintenu par le mécanisme du [VIEWSTATE]. On cherche à savoir si ce mécanisme qui régénère l'affichage de [DataGrid1] à chaque requête, régénère également sa source de données. Pour cela, celle-ci est liée au composant [DataGrid2]. La génération de celui-ci a chaque requête est faite par une liaison explicite à la source de de données de [DataGrid1]. Il a son attribut [EnableViewState] à [vrai] également.

Le code de présentation [main.aspx] de l'application est le suivant :

```
<%@ page src="main.aspx.vb" inherits="main" autoeventwireup="false" %>
<HTML>
 <HEAD>
   <title></title>
 </HEAD>
 <body>
   <form runat="server">
     <table>
       <tr>
         <td align="center">DataGrid 1</td>
         <td align="center">
           DataGrid 2</td>
       </tr>
       <tr>
         <td>
           <asp:DataGrid id="DataGrid1" runat="server" ...>
             <SelectedItemStyle ...></SelectedItemStyle>
....
           </asp:DataGrid></td>
         <td>
           <asp:DataGrid id="Datagrid2" runat="server" ...>
....
           </asp:DataGrid></td>
       </tr>
     </table>
     <asp:Button id="Button1" runat="server" Text="Envoyer"></asp:Button>
   </form>
 </body>
</HTML>
```

Le contrôleur [main.aspx.vb] est le suivant :

```
Imports System.Data

Public Class main
 Inherits System.Web.UI.Page

 Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents Datagrid2 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents Button1 As System.Web.UI.WebControls.Button

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' à la 1ère requête la source de données est définie et liée au 1er datagrid
   If Not IsPostBack Then
     'définir la source de données
     With DataGrid1
       .DataSource = createDataSource()
       .DataBind()
     End With
   End If
   ' à chaque requête, on lie datagrid2 avec la source du 1er datagrid
   With Datagrid2
     .DataSource = DataGrid1.DataSource
     .DataBind()
   End With
 End Sub

 Private Function createDataSource() As DataTable
   ' on initialise la source de données
   Dim thèmes As New DataTable
   ' colonnes
   With thèmes.Columns
     .Add("id", GetType(System.Int32))
     .Add("thème", GetType(System.String))
     .Add("description", GetType(System.String))
   End With
   ' colonne id sera clé primaire
   thèmes.Constraints.Add("cléprimaire", thèmes.Columns("id"), True)
   ' lignes
   Dim ligne As DataRow
   For i As Integer = 0 To 4
     ligne = thèmes.NewRow
     ligne.Item("id") = i.ToString
     ligne.Item("thème") = "thème" + i.ToString
     ligne.Item("description") = "description du thème " + i.ToString
     thèmes.Rows.Add(ligne)
   Next
   Return thèmes
 End Function

 Private Sub InitializeComponent()

 End Sub
End Class
```

La méthode [createDataSource] crée une source S de type [DataTable]. Nous ne nous attarderons pas sur son code qui n'est pas l'objet de l'exemple. C'est la façon utilisée pour construire les deux composants [DataGrid] qui nous intéressent :

- le composant [DataGrid1] est lié une fois à la table S, lors de la première requête. Il ne l'est plus ensuite.
- le composant [DataGrid2] est lié à la source [DataGrid1.DataSource] à chaque nouvelle requête.

Lors de la première requête; nous obtenons la vue suivante :

DataGrid 1

id	thème	description
0	thème0	description du thème 0
1	thème1	description du thème 1
2	thème2	description du thème 2
3	thème3	description du thème 3
4	thème4	description du thème 4

DataGrid 2

id	thème	description
0	thème0	description du thème 0
1	thème1	description du thème 1
2	thème2	description du thème 2
3	thème3	description du thème 3
4	thème4	description du thème 4

Envoyer

Assez logiquement, les deux composants affichent la source de données à laquelle ils ont été liés. Nous utilisons le bouton [Envoyer] pour provoquer un [PostBack] vers le serveur. La vue obtenue est alors la suivante :

DataGrid 1 DataGrid 2

id	thème	description
0	thème0	description du thème 0
1	thème1	description du thème 1
2	thème2	description du thème 2
3	thème3	description du thème 3
4	thème4	description du thème 4

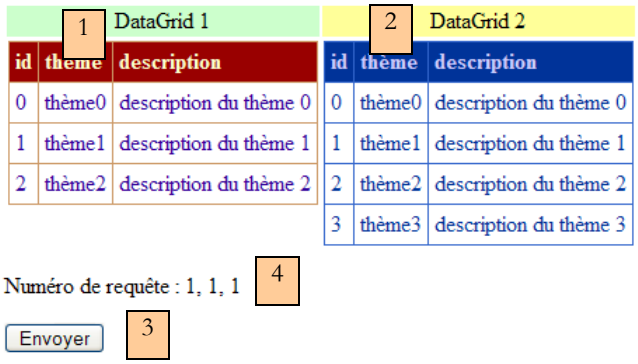
Envoyer

Nous constatons que le composant [DataGrid1] a conservé sa valeur mais pas le composant [DataGrid2]. Explications :

- avant même que la procédure [Page_Load] ne démarre, les objets [DataGrid1] et [DataGrid2] ont récupéré la valeur qu'ils avaient lors de la requête précédente, à cause du mécanisme du [viewstate]. Ils ont en effet, tous les deux, leur propriété [EnableViewState] à [vrai].
- la procédure [Page_Load] s'exécute. Comme on a affaire à une opération [PostBack], le composant [DataGrid1] n'est pas modifié par [Page_Load] (cf code). Aussi garde-t-il la valeur récupérée grâce au [viewstate]. C'est ce que montre l'écran ci-dessus.
- le composant [DataGrid2] est lui relié [DataBind] à la source de données [DataGrid1.DataSource]. Il est donc reconstruit et la valeur qu'il venait de récupérer grâce au [viewstate] est perdue. Il y aurait donc eu intérêt ici à ce qu'il ait sa propriété [EnableViewState] à [faux] afin d'éviter la gestion inutile de son état. L'écran ci-dessus montre que [DataGrid2] a été lié à une source vide. Cette source étant [DataGrid1.DataSource], on en déduit que si le mécanisme du [viewstate] restaure bien l'affichage du composant [DataGrid1], il ne restaure pas pour autant ses propriétés telles que [DataSource].

Que conclure de cet exemple ? Il faut éviter de mettre la propriété [EnableViewState] d'un conteneur de données à [vrai] si celui-ci doit être lié (DataBind) à une source de données à chaque requête. Il y a cependant certains cas, où même dans ce cas de figure, la propriété [EnableViewState] du conteneur doit être maintenue à [vrai], sinon des événements que l'on voudrait gérer ne sont pas déclenchés. Nous aurons l'occasion d'en rencontrer un exemple.

Fréquemment, la source de données d'un conteneur de données évolue au fil des requêtes. Il faut donc, à chaque requête, relier le conteneur à la source de données. Il est fréquent que celle-ci soit mise en session afin que les requêtes y aient accès. Notre second exemple montre ce mécanisme. L'application ne fournit qu'une seule vue :



n°	nom	type	propriétés	rôle
1	DataGrid1	DataGrid	EnableViewState=true	affiche une source de données S
2	DataGrid2	DataGrid	EnableViewState=false	affiche la même source S que [DataGrid1]
3	Button1	Button	EnableViewState=false	bouton [submit]
4	lblInfo1 lblInfo2 lblInfo3	Label	EnableViewState=false	textes d'information

Le composant [DataGrid1] est lié aux données uniquement lors de la première requête. Il gardera sa valeur au fil des requêtes grâce au mécanisme du [viewstate]. Le composant [DataGrid2] est lié à chaque requête à une source de données à qui on ajoute un élément à chaque nouvelle requête. Il faut donc relier (DataBind) le composant [DataGrid2] à chaque requête. On a donc mis son attribut [EnableViewState] à [faux] comme il a été conseillé précédemment. Ainsi lors de la seconde requête (usage du bouton [Envoyer]), on a la réponse suivante :

DataGrid 1

id	thème	description
0	thème0	description du thème 0
1	thème1	description du thème 1
2	thème2	description du thème 2

DataGrid 2

id	thème	description
0	thème0	description du thème 0
1	thème1	description du thème 1
2	thème2	description du thème 2
3	thème3	description du thème 3
4	thème4	description du thème 4

Numéro de requête : 1, 2, 2

Envoyer

Le composant [DataGrid1] a gardé sa valeur initiale. Le composant [DataGrid2] a un élément de plus. Les trois informations [1,2,2] représentent le n° de requête calculé de trois façons différentes. On devrait donc avoir trois fois le même résultat, ici [2, 2, 2]. On voit que l'une des informations est erronée. Nous essaierons de comprendre pourquoi.

Le code de présentation [main.aspx] de l'application est le suivant :

```
<%@ Page src="main.aspx.vb" inherits="main" autoeventwireup="false" Language="vb" %>
<HTML>
 <HEAD>
   <title></title>
 </HEAD>
 <body>
   <form runat="server">
     <table>
       <tr>
         <td align="center" bgColor="#ccffcc">DataGrid 1</td>
         <td align="center" bgColor="#ffff99">DataGrid 2</td>
       </tr>
       <tr>
         <td vAlign="top">
           <asp:DataGrid id="DataGrid1" runat="server" ...>
...
           </asp:DataGrid>
         </td>
         <td vAlign="top">
           <asp:DataGrid id="Datagrid2" runat="server" ... EnableViewState="False">
....
           </asp:DataGrid>
         </td>
       </tr>
     </table>
     <P>Numéro de requête :
       <asp:Label id="lblInfo1" runat="server" EnableViewState="False"></asp:Label>,
       <asp:Label id="lblInfo2" runat="server" EnableViewState="False"></asp:Label>,
       <asp:Label id="lblInfo3" runat="server" EnableViewState="False"></asp:Label></P>
     <P>
       <asp:Button id="Button1" runat="server" Text="Envoyer" EnableViewState="False"></asp:Button></P>
   </form>
 </body>
</HTML>
```

Le code du contrôleur [main.aspx.vb] est le suivant :

```
Imports System.Data
Imports System

Public Class main
 Inherits System.Web.UI.Page

 Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents Datagrid2 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents Button1 As System.Web.UI.WebControls.Button
 Protected WithEvents lblInfo1 As System.Web.UI.WebControls.Label
 Protected WithEvents lblInfo2 As System.Web.UI.WebControls.Label
 Protected WithEvents lblInfo3 As System.Web.UI.WebControls.Label

 Dim dtThèmes As DataTable
 Dim numRequête1 As Integer
 Dim numRequête2 As Integer
 Dim numRequête3 As New entier
```

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' à la 1ère requête la source de données est définie et liée au 1er datagrid
   If Not IsPostBack Then
     'définir la source de données
     dtThèmes = createDataSource()
     With DataGrid1
       .DataSource = dtThèmes
       .DataBind()
     End With
     ' mémoriser des informations dans la session
     Session("source") = dtThèmes
     numRequête1 = 0 : Session("numRequête1") = numRequête1
     numRequête2 = 0 : Session("numRequête2") = numRequête2
     numRequête3.valeur = 0 : Session("numRequête3") = numRequête3
   End If
   ' à chaque requête, on ajoute un nouveau thème
   dtThèmes = CType(Session("source"), DataTable)
   Dim nbThèmes = dtThèmes.Rows.Count
   Dim ligne As DataRow = dtThèmes.NewRow
   With ligne
     .Item("id") = nbThèmes.ToString
     .Item("thème") = "thème" + nbThèmes.ToString
     .Item("description") = "description du thème " + nbThèmes.ToString
   End With
   dtThèmes.Rows.Add(ligne)
   'lie datagrid2 avec la source de données
   With Datagrid2
     .DataSource = dtThèmes
     .DataBind()
   End With
   ' infos nbre de requêtes
   numRequête1 = CType(Session("numRequête1"), Integer)
   numRequête1 += 1
   lblInfo1.Text = numRequête1.ToString
   numRequête2 = CType(Session("numRequête2"), Integer)
   numRequête2 += 1
   lblInfo2.Text = numRequête2.ToString
   numRequête3 = CType(Session("numRequête3"), entier)
   numRequête3.valeur += 1
   lblInfo3.Text = numRequête3.valeur.ToString
   ' on mémorise qqs infos dans la session
   Session("numRequête2") = numRequête2
 End Sub

 Private Function createDataSource() As DataTable
....
 End Function
End Class

Public Class entier
 Private _valeur As Integer
 Public Property valeur() As Integer
   Get
     Return _valeur
   End Get
   Set(ByVal Value As Integer)
     _valeur = Value
   End Set
 End Property
End Class
```

Nous n'avons pas reproduit le code de la méthode [createDataSource]. C'est le même que dans l'application précédente au détail près qu'on ne met que trois lignes dans la source. Intéressons-nous tout d'abord à la gestion de la source de données et des deux conteneurs :

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' à la 1ère requête la source de données est définie et liée au 1er datagrid
   If Not IsPostBack Then
     'définir la source de données
     dtThèmes = createDataSource()
     With DataGrid1
       .DataSource = dtThèmes
       .DataBind()
     End With
     ' mémoriser des informations dans la session
     Session("source") = dtThèmes
...
   End If
   ' à chaque requête, on ajoute un nouveau thème
```

```
    dtThèmes = CType(Session("source"), DataTable)
    Dim nbThèmes = dtThèmes.Rows.Count
    Dim ligne As DataRow = dtThèmes.NewRow
    With ligne
      .Item("id") = nbThèmes.ToString
      .Item("thème") = "thème" + nbThèmes.ToString
      .Item("description") = "description du thème " + nbThèmes.ToString
    End With
    dtThèmes.Rows.Add(ligne)
    'lie datagrid2 avec la source de données
    With Datagrid2
      .DataSource = dtThèmes
      .DataBind()
    End With
...
 End Sub
```

Le composant [DataGrid1] n'est lié à la source S de données que lors de la première requête (not IsPostBack). Celle-ci est alors placée dans la session. Elle ne le sera plus par la suite. La source de données S mise en session est récupérée à chaque requête et se voit ajouter une nouvelle ligne. Le composant [DataGrid2] est lié explicitement, à chaque requête, à la source S. C'est pourquoi son contenu augmente d'une ligne à chaque requête. On constate qu'après avoir modifié le contenu de la source S, on ne remet pas explicitement celle-ci en session par une opération :

```
    Session("source") = S
```

Pourquoi ? Lorsqu'une requête démarre, elle contient un objet Session("source") de type [DataTable] qui est la source de données telle qu'elle était lors de la dernière requête. Lorsque nous écrivons :

```
    dtThèmes = CType(Session("source"), DataTable)
```

[dtThèmes] et [Session(" source ")] sont deux références sur un même objet [DataTable]. L' objet référencé peut être modifié indifféremment par l'une ou l'autre des références. Ceci n'est plus vrai lorsque les données mises en session ne sont pas des objets, telles que les structures [Integer, Float, ...]. C'est ce que montre la gestion des compteurs de requêtes :

```
 Dim numRequête1 As Integer
 Dim numRequête2 As Integer
 Dim numRequête3 As New entier

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' à la 1ère requête la source de données est définie et liée au 1er datagrid
....
    ' mémoriser des informations dans la session
    Session("source") = dtThèmes
    numRequête1 = 0 : Session("numRequête1") = numRequête1
    numRequête2 = 0 : Session("numRequête2") = numRequête2
    numRequête3.valeur = 0 : Session("numRequête3") = numRequête3
   End If
   ' à chaque requête, on ajoute un nouveau thème
....
   ' infos nbre de requêtes
   numRequête1 = CType(Session("numRequête1"), Integer)
   numRequête1 += 1
   lblInfo1.Text = numRequête1.ToString
   numRequête2 = CType(Session("numRequête2"), Integer)
   numRequête2 += 1
   lblInfo2.Text = numRequête2.ToString
   numRequête3 = CType(Session("numRequête3"), entier)
   numRequête3.valeur += 1
   lblInfo3.Text = numRequête3.valeur.ToString
   ' on mémorise qqs infos dans la session
   Session("numRequête2") = numRequête2
 End Sub

 Private Function createDataSource() As DataTable
....
 End Function

 Private Sub InitializeComponent()

 End Sub
End Class

Public Class entier
 Private _valeur As Integer
 Public Property valeur() As Integer
   Get
     Return _valeur
   End Get
```

```
    Set(ByVal Value As Integer)
      _valeur = Value
    End Set
  End Property
```

On mémorise les compteurs de requêtes dans trois éléments :

- numRequête1 et numRequête2 de type [Integer] - [Integer] n'est pas une classe mais une structure
- numRequête3 de type [entier] - [entier] est une classe définie pour l'occasion

Lorsqu'on écrit :

```
    numRequête1 = CType(Session("numRequête1"), Integer)
..
    numRequête2 = CType(Session("numRequête2"), Integer)
..
    numRequête3 = CType(Session("numRequête3"), entier)
..
```

- la structure [Session("numRequête1")] est recopiée dans [numRequête1]. Ainsi lorsque l'élément [numRequête1] est modifié, l'élément [Session("numRequête1")] lui, ne l'est pas
- il en est de même pour [Session("numRequête2")] et [numRequête2]
- les éléments [Session("numRequête3")] et [numRequête3] sont eux deux références à un même objet de type [entier]. L'objet référencé peut être modifié indifféremment par l'une ou l'autre des références.

Ce ceci, on déduit qu'il est inutile d'écrire :

```
    Session("numRequête3") = numRequête3
```

pour mémoriser la nouvelle valeur de [numRequête3] dans la session. En revanche, il faudrait écrire :

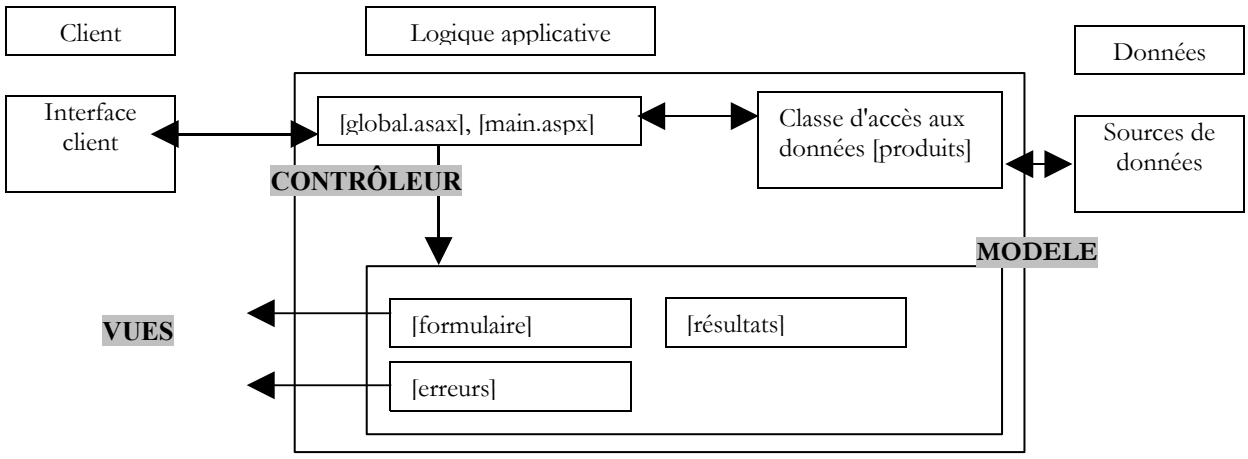
```
    Session("numRequête1") = numRequête1
    Session("numRequête2") = numRequête2
```

pour mémoriser les nouvelles valeurs des structures [numRequête1] et [numRequête2]. Nous ne le faisons ici que pour [numRequête2], ce qui explique que sur la copie d'écran obtenue à l'issue de la seconde requête, le compteur [numRequête1] est erroné.

On retiendra donc qu'une fois mise en session, une donnée n'a pas à y être remise de façon répétée si elle est représentée par un objet. Dans les autres cas, il faut le faire si elle a été modifiée.

2.6 Affichage d'une liste de données à l'aide d'un DataGrid paginé et trié

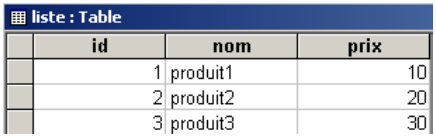
Le composant [DataGrid] permet d'afficher le contenu d'un [DataSet]. Pour l'instant, nous avons toujours construit nos [DataSet] "à la main". Cette fois, nous utilisons un [DataSet] issu d'une base de données. Nous construisons l'application MVC suivante :



Les trois vues seront incorporées dans le code de présentation du contrôleur [main.aspx] sous la forme de conteneurs. Aussi cette application a-t-elle une unique page [main.aspx].

2.6.1 La classe d'accès aux données

La classe [produits] donne accès à la base ACCESS suivante :



liste : Table

id	nom	prix
1	produit1	10
2	produit2	20
3	produit3	30

Le code de la classe [produits] est le suivant :

```
Imports System
Imports System.Data
Imports System.Data.OleDb
Imports System.Xml

Namespace st.istia.univangers.fr

 Public Class produits
   Private chaineConnexionOLEDB As String

   Public Sub New(ByVal chaineConnexionOLEDB As String)
     ' on mémorise la chaîne de connexion
     Me.chaineConnexionOLEDB = chaineConnexionOLEDB
   End Sub

   Public Function getDataSet(ByVal commande As String) As DataSet
     ' on crée un objet DataAdapter pour lire les données de la source OLEDB
     Dim adaptateur As New OleDbDataAdapter(commande, chaineConnexionOLEDB)
     ' on crée une image en mémoire du résultat du select
     Dim contenu As New DataSet
     Try
       adaptateur.Fill(contenu)
     Catch e As Exception
       Throw New Exception("Erreur d'accès à la base de données (" + e.Message + ")")
     End Try
     ' on rend le résultat
     Return contenu
   End Function
 End Class
End Namespace
```

La base ACCESS sera gérée via un pilote OLEDB. On donne au constructeur de la classe [produits], la chaîne de connexion identifiant et le pilote OLEDB et la base que l'on doit gérer. La classe [produits] a une méthode [getDataSet] qui rend un [DataSet] obtenu par exécution d'une requête SQL [select] dont le texte est passé en paramètre. Au cour de la méthode, plusieurs opérations ont lieu : création de la connexion à la base, exécution du [select], fermeture de la connexion. Tout ceci peut générer une exception qui est ici gérée. Un programme de test pourrait être le suivant :

```
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.VisualBasic

Namespace st.istia.univangers.fr

 ' pg de test
 Module testproduits
   Sub Main(ByVal arguments() As String)
     ' affiche le contenu d'une table de produits
     ' la table est dans une base ACCESS dont le pg reçoit le nom de fichier
     Const syntaxe1 As String = "pg bdACCESS"

     ' vérification des paramètres du programme
     If arguments.Length <> 1 Then
       ' msg d'erreur
       Console.Error.WriteLine(syntaxe1)
       ' fin
       Environment.Exit(1)
     End If

     ' on prépare la chaîne de connexion
     Dim chaineConnexion As String = "Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data Source="
+ arguments(0)

     ' création d'un objet produits
```

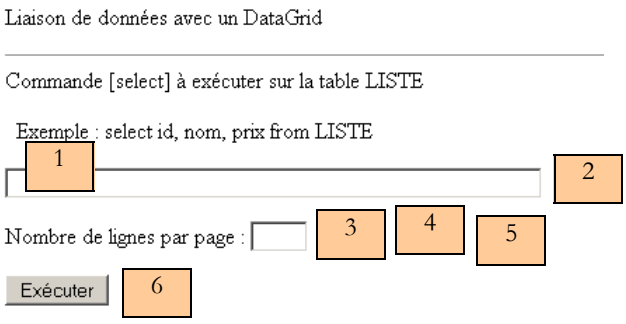
```
    Dim objProduits As produits = New produits(chaineConnexion)

    ' on récupère la table des produits dans un dataset
    Dim contenu As DataSet
    Try
      contenu = objProduits.getDataSet("select id,nom,prix from liste")
    Catch ex As Exception
      Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
      Environment.Exit(2)
    End Try

    ' on affiche son contenu
    Dim lignes As DataRowCollection = contenu.Tables(0).Rows
    For i As Integer = 0 To lignes.Count - 1
      ' ligne i de la table
      Console.Out.WriteLine(lignes(i).Item("id").ToString + "," + lignes(i).Item("nom").ToString + _
      "," + lignes(i).Item("prix").ToString)
    Next
  End Sub
 End Module
End Namespace
```

La compilation des différents éléments de cette application se fait de la façon suivante :

```
dos>vbc /t:library /r:system.dll /r:system.data.dll /r:system.xml.dll produits.vb

dos>vbc /r:produits.dll /r:system.data.dll /r:system.xml.dll /r:system.dll testproduits.vb

dos>dir
06/05/2004  16:52               118 784 produits.mdb
07/05/2004  11:07                   902 produits.vb
07/04/2004  07:01                 1 532 testproduits.vb
07/05/2004  14:21                 3 584 produits.dll
07/05/2004  14:22                 4 608 testproduits.exe

dos>testproduits produits.mdb
1,produit1,10
2,produit2,20
3,produit3,30
```

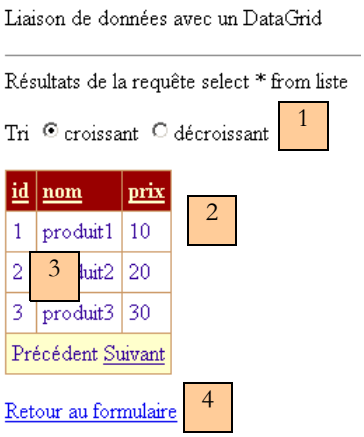
2.6.2 Les vues

Maintenant que nous avons la classe d'accès aux données, nous écrivons les contrôleurs et vues de cette application web. Voyons tout d'abord son fonctionnement. La première vue est la suivante :



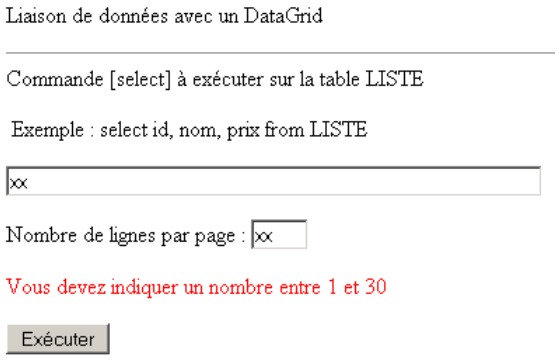
n°	nom	type	propriétés	rôle
1	txtSelect	TextBox	EnableViewState=true	champ de saisie de la requête select
2	RequiredFieldValidator1	RequiredFieldValidator	EnableViewState=false	vérifie la présence de 1
3	txtPages	TextBox	EnableViewState=true	champ de saisie - indique le nombre de lignes de données à afficher par page de résultats
4	RequiredFieldValidator2	RequiredFieldValidator	EnableViewState=false	vérifie la présence de 3
5	RangeValidator1	RangeValidator	EnableViewState=false	vérifie que (3) est dans l'intervalle [1,30]
6	btnExécuter	Button CausesValidation=true	EnableViewState=false	bouton [submit]

Nous appellerons cette vue, la vue [formulaire]. Elle permet à l'utilisateur de faire exécuter une requête SQL select sur la base [produits.mdb]. L'exécution de la requête donne naissance à une nouvelle vue :

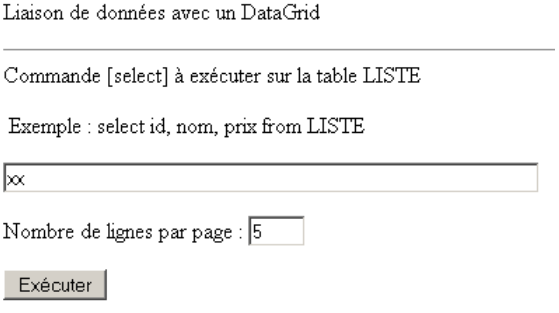


n°	nom	type	propriétés	rôle
1	lblSelect	Label	EnableViewState=false	champ d'information
2	rdCroissant rdDécroissant	RadioButton	EnableViewState=false GroupName=rdTri	permet de choisir un sens de tri
3	DataGrid1	DataGrid	EnableViewState=true AllowPaging=true AllowSorting=true	table d'affichage du résultat du select
4	lnkRésultats	LinkButton	EnableViewState=false	lien-bouton [submit]

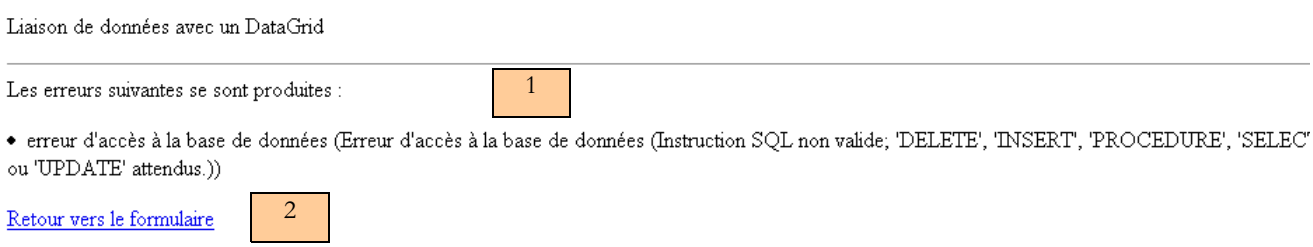
Nous appellerons cette vue, la vue [résultats]. C'est elle qui contient le [DataGrid] qui va afficher le résultat de la commande SQL [select]. L'utilisateur peut se tromper dans sa requête. Certaines erreurs lui seront signalées dans la vue [erreurs] grâce aux contrôles de validation.



D'autres erreurs lui seront signalées par la vue [erreurs] :



La vue [erreurs] est affichée :



n°	nom	type	propriétés	rôle
1	erreursHTML	variable		code HTML nécessaire à l'affichage des erreurs
3	lnkForm2	LinkButton	EnableViewState=false	lien-bouton [submit]

Les trois vues de l'application sont trois conteneurs (panel) différents au sein de la même page [main.aspx]. Le code de présentation de celle-ci est le suivant :

```
<%@ Page src="main.aspx.vb" inherits="main" autoeventwireup="false" Language="vb" %>
<HTML>
 <HEAD>
 </HEAD>
 <body>
   <P>Liaison de données avec un DataGrid</P>
   <HR width="100%" SIZE="1">
   <form runat="server">
     <asp:panel id="vueFormulaire" runat="server">
       <P>Commande [select] à exécuter sur la table LISTE</P>
       <P> Exemple : select id, nom, prix from LISTE
       </P>
       <P>
         <asp:TextBox id="txtSelect" runat="server" Columns="60"></asp:TextBox></P>
       <P>
         <asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server" EnableViewState="False"
EnableClientScript="False"
           ErrorMessage="Indiquez la requête [select] à exécuter" ControlToValidate="txtSelect"
Display="Dynamic"></asp:RequiredFieldValidator></P>
       <P>Nombre de lignes par page :
         <asp:TextBox id="txtPages" runat="server" Columns="3"></asp:TextBox></P>
       <P>
         <asp:RequiredFieldValidator id="RequiredFieldValidator2" runat="server" EnableViewState="False"
EnableClientScript="False"
           ErrorMessage="Indiquez le nombre de lignes par page désirées" ControlToValidate="txtPages"
Display="Dynamic"></asp:RequiredFieldValidator>
         <asp:RangeValidator id="RangeValidator1" runat="server" EnableViewState="False"
EnableClientScript="False"
           ErrorMessage="Vous devez indiquer un nombre entre 1 et 30" ControlToValidate="txtPages"
Display="Dynamic"
           Type="Integer" MinimumValue="1" MaximumValue="30"></asp:RangeValidator></P>
       <P>
         <asp:Button id="btnExécuter" runat="server" EnableViewState="False"
Text="Exécuter"></asp:Button></P>
     </asp:panel>

     <asp:Panel id="vueRésultats" runat="server">
       <P>Résultats de la requête
         <asp:Label id="lblSelect" runat="server"></asp:Label></P>
       <P>Tri
         <asp:RadioButton id="rdCroissant" runat="server" Text="croissant" Checked="True"
GroupName="rdTri"></asp:RadioButton>
         <asp:RadioButton id="rdDécroissant" runat="server" Text="décroissant"
GroupName="rdTri"></asp:RadioButton></P>
       <P>
         <asp:DataGrid id="DataGrid1" runat="server" BorderColor="#CC9966" BorderStyle="None"
BorderWidth="1px"
           BackColor="White" CellPadding="4" AllowPaging="True" PageSize="4" AllowSorting="True">
           <SelectedItemStyle Font-Bold="True" ForeColor="#663399"
BackColor="#FFCC66"></SelectedItemStyle>
           <ItemStyle ForeColor="#330099" BackColor="White"></ItemStyle>
           <HeaderStyle Font-Bold="True" ForeColor="#FFFFCC" BackColor="#990000"></HeaderStyle>
           <FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>
           <PagerStyle NextPageText="Suivant" PrevPageText="Pr&#233;c&#233;dent" HorizontalAlign="Center"
             ForeColor="#330099" BackColor="#FFFFCC"></PagerStyle>
         </asp:DataGrid></P>
       <P>
```

```
        <asp:LinkButton id="lnkRésultats" runat="server" EnableViewState="False">Retour au
formulaire</asp:LinkButton></P>
      </asp:Panel>

      <asp:Panel id="vueErreurs" runat="server">
        <P>Les erreurs suivantes se sont produites :</P>
        <% =erreursHTML %>
        <P>
          <asp:LinkButton id="lnkErreurs" runat="server" EnableViewState="False">Retour vers le
formulaire</asp:LinkButton></P>
      </asp:Panel>
    </form>
  </body>
</HTML>
```

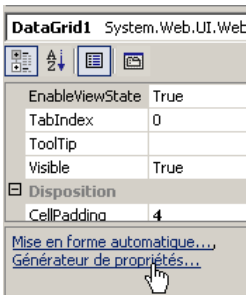
2.6.3 Configuration du DataGrid

Attardons-nous sur la pagination du composant [DataGrid], option que nous rencontrons pour la première fois. Dans le code ci-dessus, cette pagination est contrôlée par les attributs suivants :

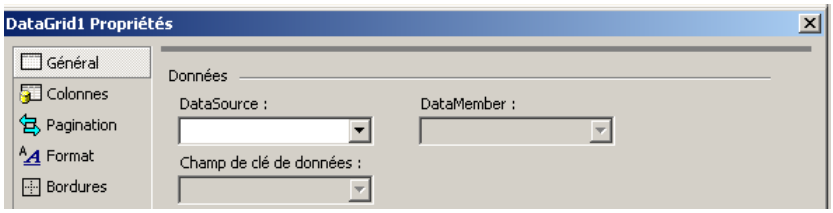
```
        <asp:DataGrid id="DataGrid1" runat="server" PageSize="4" AllowPaging="True" ...>
...
          <PagerStyle NextPageText="Suivant" PrevPageText="Pr&#233;c&#233;dent" ...></PagerStyle>
        </asp:DataGrid></P>
```

`AllowPaging="true"`	autorise la pagination
`PageSize="4"`	quatre lignes de données par page
`NextPageText="Suivant"`	Le texte du lien pour aller à la page suivante de la source de données
`PrevPageText="Précédent"`	Le texte du lien pour aller à la page précédente de la source de données

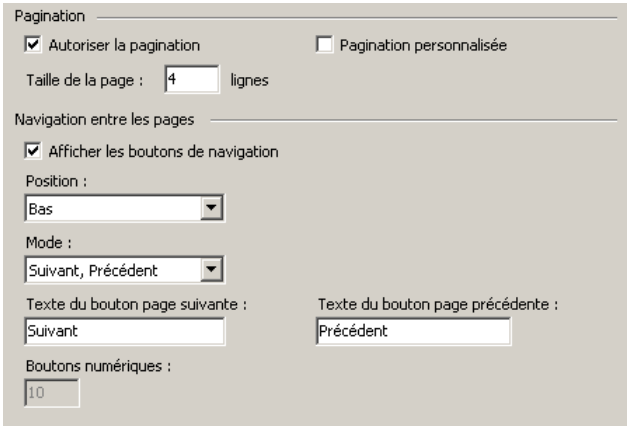
Ces informations peuvent être saisies directement dans les attributs de la balise <asp:datagrid>. On peut également s'aider de [WebMatrix]. Dans la fenêtre de propriétés du [DataGrid], on suit le lien [Générateur de propriétés] :



On obtient l'assistant suivant :



On prend l'option [Pagination] :



Nous retrouvons ci-dessus, les valeurs des attributs de pagination du [DataGrid] du code de présentation.

Par ailleurs, nous autorisons le tri des données sur l'une des colonnes du [DataGrid]. Il y a différentes façons de faire cela. L'une d'elles est de définir la propriété [AllowSorting=true] dans la fenêtre de propriétés du [DataGrid]. On peut aussi utiliser le générateur de propriétés. Quelque soit la méthode utilisée, cela se traduit par la présence de l'attribut [AllowSorting=true] dans la balise <asp:DataGrid> du composant :

```
<asp:DataGrid id="DataGrid1" runat="server" ... AllowPaging="True" PageSize="4" AllowSorting="True">
```

2.6.4 Les contrôleurs

Le contrôleur [global.asax, global.asax.vb] est le suivant :

[global.asax]

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

[global.asax.vb]

```
Imports st.istia.univangers.fr
Imports System.Configuration

Public Class Global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on crée un objet produits
   Dim objProduits As produits
   Try
     objProduits = New produits(ConfigurationSettings.AppSettings("OLEDBStringConnection"))
     ' on met l'objet dans l'application
     Application("objProduits") = objProduits
     ' pas d'erreur
     Application("erreur") = False
   Catch ex As Exception
     'il y a eu erreur, on le note dans l'application
     Application("erreur") = True
     Application("message") = ex.Message
   End Try
 End Sub
End Class
```

Le fichier [web.config] de configuration de l'application pourrait ressembler à ceci :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.    <appSettings>
4.      <add key="OLEDBStringConnection" value="Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4;
    Data Source=D:\data\devel\aspnet\poly\webforms2\vs\dataset5\produits.mdb" />
5.    </appSettings>
6.  </configuration>
```

Lorsque l'application démarre (Application_Start), nous construisons un objet [produits] et le mettons dans l'application afin qu'il soit disponible pour toutes les requêtes de tous les clients. S'il y a exception lors de cette construction, elle est notée dans

l'application. La procédure [Application_Start] ne s'exécutera qu'une fois. Ensuite le contrôleur [global.asax] n'interviendra plus. C'est le contrôleur [main.aspx.vb] qui fera alors le travail :

```
Imports System.Collections
Imports Microsoft.VisualBasic
Imports System.Data
Imports st.istia.univangers.fr
Imports System
Imports System.Xml

Public Class main
 Inherits System.Web.UI.Page

 ' composants page
 Protected WithEvents txtSelect As System.Web.UI.WebControls.TextBox
 Protected WithEvents RequiredFieldValidator1 As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents txtPages As System.Web.UI.WebControls.TextBox
 Protected WithEvents RequiredFieldValidator2 As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents RangeValidator1 As System.Web.UI.WebControls.RangeValidator
 Protected WithEvents btnExécuter As System.Web.UI.WebControls.Button
 Protected WithEvents vueFormulaire As System.Web.UI.WebControls.Panel
 Protected WithEvents lblSelect As System.Web.UI.WebControls.Label
 Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents lnkRésultats As System.Web.UI.WebControls.LinkButton
 Protected WithEvents vueRésultats As System.Web.UI.WebControls.Panel
 Protected WithEvents lnkErreurs As System.Web.UI.WebControls.LinkButton
 Protected WithEvents vueErreurs As System.Web.UI.WebControls.Panel
 Protected WithEvents rdCroissant As System.Web.UI.WebControls.RadioButton
 Protected WithEvents rdDécroissant As System.Web.UI.WebControls.RadioButton
 ' données page
 Protected erreursHTML As String

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' on regarde si l'application est en erreur
   If CType(Application("erreur"), Boolean) Then
     ' l'application ne s'est pas initialisée correctement
     Dim erreurs As New ArrayList
     erreurs.Add("Application momentanément indisponible (" + CType(Application("message"), String) +
")")
     afficheErreurs(erreurs, False)
     Exit Sub
   End If
   '1ère requête
   If Not IsPostBack Then
     ' on affiche le formulaire vide
     afficheFormulaire()
   End If
 End Sub

 Private Sub afficheErreurs(ByVal erreurs As ArrayList, ByVal afficheLien As Boolean)
   ' affiche la vue erreurs
   erreursHTML = ""
   For i As Integer = 0 To erreurs.Count - 1
     erreursHTML += "<li>" + erreurs(i).ToString + "</li>" + ControlChars.CrLf
   Next
   lnkErreurs.Visible = afficheLien
   ' on affiche la vue [erreurs]
   vueErreurs.Visible = True
   vueFormulaire.Visible = False
   vueRésultats.Visible = False
 End Sub

 Private Sub afficheFormulaire()
   ' on affiche la vue [formulaire]
   vueFormulaire.Visible = True
   vueErreurs.Visible = False
   vueRésultats.Visible = False
 End Sub

 Private Sub afficheRésultats(ByVal sqlTexte As String, ByVal données As DataView)
   ' on initialise les contrôles
   lblSelect.Text = sqlTexte
   With DataGrid1
     .DataSource = données
     .PageSize = CType(txtPages.Text, Integer)
     .CurrentPageIndex = 0
     .DataBind()
   End With
   ' on affiche la vue [résultats]
   vueRésultats.Visible = True
   vueFormulaire.Visible = False
```

```
    vueErreurs.Visible = False
  End Sub

  Private Sub btnExécuter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnExécuter.Click
    ' page valide ?
    If Not Page.IsValid Then
      afficheFormulaire()
      Exit Sub
    End If

    ' exécution de la requête SELECT client
    Dim données As DataView
    Try
      données = CType(Application("objProduits"),
produits).getDataSet(txtSelect.Text.Trim).Tables(0).DefaultView
    Catch ex As Exception
      Dim erreurs As New ArrayList
      erreurs.Add("erreur d'accès à la base de données (" + ex.Message + ")")
      afficheErreurs(erreurs, True)
      Exit Sub
    End Try
    ' tout va bien - on affiche les résultats
    afficheRésultats(txtSelect.Text.Trim, données)
    ' on met les données dans la session
    Session("données") = données
  End Sub

  Private Sub retourFormulaire(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkErreurs.Click, lnkRésultats.Click
    ' jeu de vues
    vueErreurs.Visible = False
    vueFormulaire.Visible = True
    vueRésultats.Visible = False
  End Sub

  Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles DataGrid1.PageIndexChanged
    ' chgt de page
    With DataGrid1
      .CurrentPageIndex = e.NewPageIndex
      .DataSource = CType(Session("données"), DataView)
      .DataBind()
    End With
  End Sub

  Private Sub DataGrid1_SortCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridSortCommandEventArgs) Handles DataGrid1.SortCommand
    ' on trie le dataview
    Dim données As DataView = CType(Session("données"), DataView)
    données.Sort = e.SortExpression + " " + CType(IIf(rdCroissant.Checked, "asc", "desc"), String)
    ' on l'affiche
    With DataGrid1
      .CurrentPageIndex = 0
      .DataSource = données
      .DataBind()
    End With
  End Sub
End Class
```

Au chargement de la page [Page_Load], nous regardons tout d'abord si l'application a pu s'initialiser correctement. Si ce n'est pas le cas, on affiche la vue [erreurs] sans lien de retour vers le formulaire car ce lien est alors inutile. En effet, seule la vue [erreurs] peut s'afficher si l'application n'a pas pu s'initialiser correctement. Sinon, on affiche la vue [formulaire] si on a affaire à la première requête du client. Pour le reste, nous laissons au lecteur le soin de comprendre le code. Nous nous attardons seulement sur trois procédures, la procédure [btnExécuter_Click] qui s'exécute lorsque l'utilisateur a demandé l'exécution de la requête SQL saisie dans la vue [formulaire], la procédure [DataGrid1_PageIndexChanged] qui est exécutée lorsque l'utilisateur utilise les liens [Suivant] et [Précédent] du [DataGrid] et la procédure [DataGrid1_SortCommand] qui est exécutée lorsque l'utilisateur clique sur le titre d'une colonne pour trier les données selon l'ordre de celle-ci. Le sens de l'ordre, croissant ou décroissant est fixé par les deux boutons radio de tri.

Dans la procédure [btnExécuter_Click], on commence donc par vérifier si la page est valide ou non. Lorsque la procédure [btnExécuter_Click] s'exécute, les vérifications liées aux différents contrôles de validation de la page ont été faites. Pour chaque contrôle de validation, deux attributs ont été positionnés :

`IsValid`	à vrai si la donnée vérifiée est valide, à faux sinon

`ErrorMessage`	le message d'erreur si la donnée vérifiée s'est avérée invalide

Pour la page elle-même, un attribut [IsValid] a été positionné. Il est à vrai uniquement si tous les contrôles de validation ont leur attribut [IsValid] à vrai. Si ce n'est pas le cas, il faut afficher la vue [formulaire]. Celle-ci contient les contrôles de validation qui afficheront leur attribut [ErrorMessage]. Si la page est valide , on utilise l'objet de type [produits] créé par [Application_Start] afin d'obtenir le [DataSet] correspondant à l'exécution de la requête SQL select. On transforme celui-ci en objet [DataView] :

```
    Dim données As DataView
    Try
      données = CType(Application("objProduits"),
produits).getDataSet(txtSelect.Text.Trim).Tables(0).DefaultView
...
```

On aurait pu travailler simplement avec le [DataSet] et écrire :

```
    Dim données As DataSet
    Try
      données = CType(Application("objProduits"), produits).getDataSet(txtSelect.Text.Trim)
...
```

Un objet [DataSet] est, à la base, un ensemble de tables liées par des relations. Dans notre application particulière, le [DataSet] obtenu de la classe [produits] ne contient qu'une table, celle issue du résultat de l'instruction [select]. Une table peut être triée alors qu'un [DataSet] ne le peut pas, or on est intéressé par trier les données obtenues. Pour travailler avec la table résultat du [select], on aurait pu écrire :

```
    Dim données As DataTable
    Try
      données = CType(Application("objProduits"), produits).getDataSet(txtSelect.Text.Trim).Tables(0)
...
```

L'objet [DataTable], bien que représentant une table de base de données n'a pas de méthode de tri. Il faut pour cela avoir une vue sur la table. Une vue est un objet de type [DataView]. On peut avoir différentes vues sur une même table à l'aide de filtres. Une table a une vue par défaut qui est celle où aucun filtre n'est défini. Elle représente donc la totalité de la table. Cette vue par défaut est obtenue par [DataTable.DefaultView]. On peut trier une vue à l'aide de sa propriété [sort] sur laquelle nous reviendrons.

Si l'obtention du [DataSet] de la classe [produits] se passe bien, la vue [résultats] est affichée sinon c'est la vue [erreurs]. L'affichage de la vue [résultats] se fait par la procédure [afficheRésultats] à qui on passe deux paramètres :

- le texte à placer dans le label [lblSelect]
- le [DataView] à lier à [DataGrid1]

Cet exemple nous montre la grande souplesse du composant [DataGrid]. Il sait reconnaître la structure du [DataView] auquel on le lie et s'adapter à celle-ci. Enfin, la procédure [btnExécuter_Click] mémorise le [DataView] qu'elle vient d'obtenir dans la session de l'utilisateur afin d'en disposer lorsque celui-ci va demander d'autres pages du même [DataView].

La procédure [DataGrid1_PageIndexChanged] est exécutée lorsque l'utilisateur utilise les liens [Suivant] et [Précédent] du [DataGrid]. Elle reçoit deux paramètres :

```
 Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As System.Web.UI.WebControls.)
Handles DataGrid1.PageIndexChanged
```

`source`	l'objet origine de l'événement - ici, l'un des liens [Suivant] ou [Précédent]
`e`	des informations sur l'événement. L'attribut [e.NewPageIndex] est le n° de page à afficher pour répondre à la demande du client

Le code complet du gestionnaire d'événement est le suivant :

```
 Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles DataGrid1.PageIndexChanged
   ' chgt de page
   With DataGrid1
     .CurrentPageIndex = e.NewPageIndex
     .DataSource = CType(Session("données"), DataView)
     .DataBind()
   End With
 End Sub
```

Le composant [DataGrid] a un attribut [CurrentPageIndex] qui indique le n° de la page qu'il affiche ou va afficher. On donne à cet attribut la valeur [NewPageIndex] du paramètre [e]. Le [DataGrid] est ensuite associé au [DataView] qui avait été mémorisé dans la session, par la procédure [btnExécuter_Click].

On peut se demander si le [DataGrid] a besoin de l'attribut [EnableViewState=true] puisque son contenu est calculé par le code à chaque nouvelle exécution de la page. On pouvait penser que non. Or, si le [DataGrid] a l'attribut [EnableViewState=false], on constate que l'événement [DataGrid1.PageIndexChanged] n'est jamais déclenché. C'est pour cela qu'on a laissé [EnableViewState=true]. On sait que cela entraîne que le contenu du [DataGrid] va être mis dans le champ caché [__VIEWSTATE] de la page. Cela peut entraîner une surcharge importante de la page si le [DataGrid] est important. Si ce fait est gênant, on peut alors gérer la pagination soi-même sans utiliser la pagination automatique du [DataGrid].

La procédure [DataGrid1_SortCommand] est exécutée lorsque l'utilisateur clique sur le titre d'une des colonnes affichées par le [DataGrid] pour demander le tri des données dans l'ordre de cette colonne. Elle reçoit deux paramètres :

```
 Private Sub DataGrid1_SortCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridSortCommandEventArgs) Handles DataGrid1.SortCommand
```

source	l'objet origine de l'événement
e	des informations sur l'événement. L'attribut [e.SortExpression] est le nom de la colonne cliquée pour demander le tri

Le code complet du gestionnaire d'événement est le suivant :

```
 Private Sub DataGrid1_SortCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridSortCommandEventArgs) Handles DataGrid1.SortCommand
   ' on trie le dataview
   Dim données As DataView = CType(Session("données"), DataView)
   données.Sort = e.SortExpression + " " + CType(IIf(rdCroissant.Checked, "asc", "desc"), String)
   ' on l'affiche
   With DataGrid1
     .CurrentPageIndex = 0
     .DataSource = données
     .DataBind()
   End With
 End Sub
```

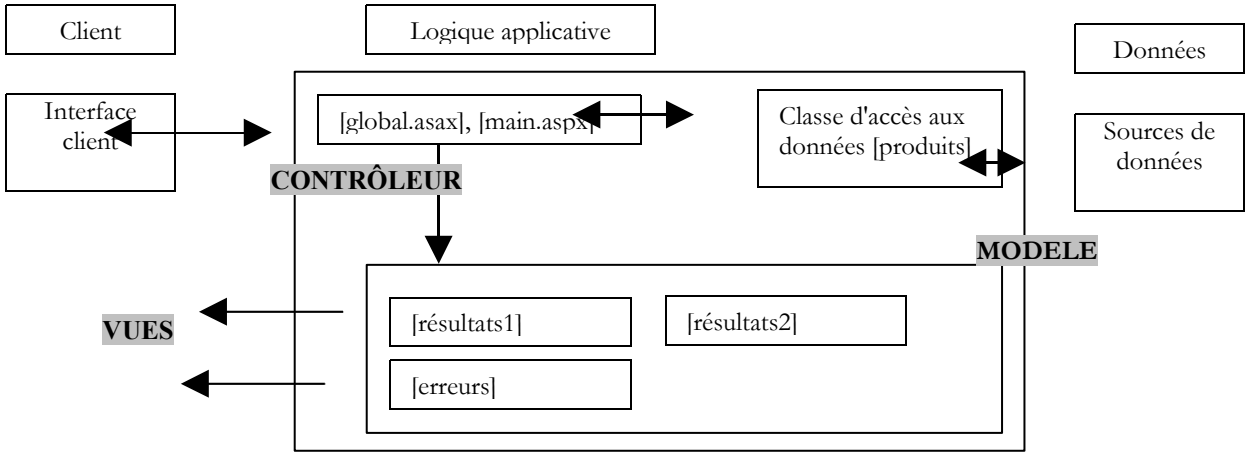
On récupère le [DataView] visualisé par le [DataGrid] dans la session courante. C'est la procédure [btnExécuter_Click] qui l'y avait mis. Le composant [DataView] a une propriété [Sort] à qui on attribue l'expression de tri. Celle-ci obéit à la syntaxe [select ... order by expr1, expr2, ...] où chaque [expri] peut être suivi du mot clé [asc] pour un tri croissant ou [desc] pour un tri décroissant. L'expression [order by] utilisée ici, est [order by colonne asc/desc]. La propriété [e.SortExpression] nous donne le nom de la colonne du [DataGrid] qui a été cliquée pour le tri. La chaîne [asc/desc] est fixée d'après les valeurs des boutons radio du groupe [rdTri]. Une fois fixée l'expression de tri du [DataView], le [DataGrid] est associé à celui-ci. On positionne le [DataGrid] sur sa première page.

2.7 Composant DataList et liaison de données

Nous nous intéressons maintenant au composant [DataList]. Il offre plus de possibilités de mise en forme que le [DataGrid] mais est moins souple. Ainsi, il ne sait pas s'adapter automatiquement à la source de données à laquelle il est lié. Il faut faire cette adaptation par code si celle-ci est souhaitée. Si la structure de la source de données est connue d'avance, alors ce composant offre des possibilités de mise en forme qui peuvent le faire préférer au [DataGrid].

2.7.1 Application

Afin d'illustrer l'utilisation du [DataList], nous construisons une application MVC analogue à la précédente :



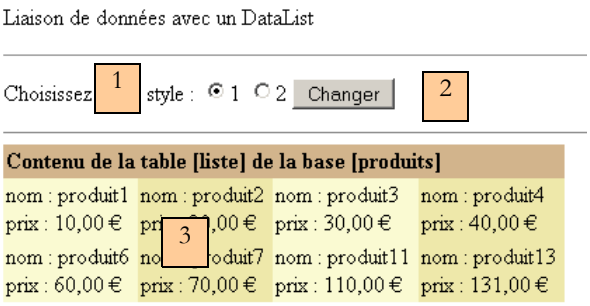
Les trois vues seront incorporées dans le code de présentation du contrôleur [main.aspx] sous la forme de conteneurs. Aussi cette application a-t-elle une unique page [main.aspx].

2.7.2 La classe d'accès aux données

La classe [produits] est la même que précédemment.

2.7.3 Les vues

Lorsque l'utilisateur fait sa première requête à l'application, il obtient la vue [résultats1] suivante :

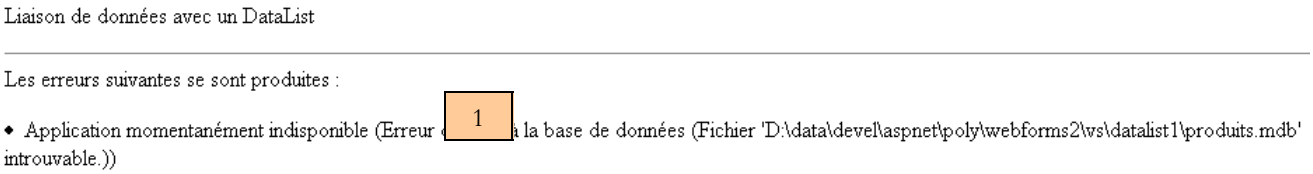


n°	nom	type	propriétés	rôle
1	`RadioButton1` `RadioButton2`	RadioButton	EnableViewState=false	permet de choisir un style de [DataList] parmi deux
2	`btnChanger`	Button	EnableViewState=false	bouton [submit]
3	`DataList1`	DataList	EnableViewState=true	champ d'affichage de la liste de données

Si l'utilisateur choisit le style n° 2, il obtient la vue [résultats2] suivante :

n°	nom	type	propriétés	rôle
1	DataList2	DataList	EnableViewState=true	champ d'affichage de la liste de données

La vue [erreurs] signale un problème d'accès à la source de données :



n°	nom	type	propriétés	rôle
1	erreursHTML	variable		code HTML nécessaire à l'affichage des erreurs

Les trois vues de l'application sont trois conteneurs (panel) différents au sein de la même page [main.aspx]. Le code de présentation de celle-ci est le suivant :

```
<%@ Page src="main.aspx.vb" inherits="main" autoeventwireup="false" Language="vb" %>
<HTML>
 <HEAD>
 </HEAD>
 <body>
   <P>Liaison de données avec un DataList</P>
   <HR width="100%" SIZE="1">
   <form runat="server" ID="Form1">
```

```
     <asp:Panel Runat="server" ID="bandeau">
       <P>Choisissez votre style :
         <asp:RadioButton id="RadioButton1" runat="server" EnableViewState="False" Text="1"
GroupName="rdstyle"
          Checked="True"></asp:RadioButton>
         <asp:RadioButton id="RadioButton2" runat="server" EnableViewState="False" Text="2"
GroupName="rdstyle"></asp:RadioButton>
         <asp:Button id="btnChanger" runat="server" EnableViewState="False"
Text="Changer"></asp:Button></P>
       <HR width="100%" SIZE="1">
     </asp:Panel>
```

```
     <asp:Panel id="vueRésultats1" runat="server">
       <P>
         <asp:DataList id="DataList1" runat="server" BorderColor="Tan" BorderWidth="1px"
BackColor="LightGoldenrodYellow"
          CellPadding="2" RepeatDirection="Horizontal" RepeatColumns="4" ForeColor="Black">
          <SelectedItemStyle ForeColor="GhostWhite" BackColor="DarkSlateBlue"></SelectedItemStyle>
          <HeaderTemplate>
            Contenu de la table [liste] de la base [produits]
```

```
          </HeaderTemplate>
          <AlternatingItemStyle BackColor="PaleGoldenrod"></AlternatingItemStyle>
          <ItemTemplate>
            nom :
            <%# Container.DataItem("nom") %>
            <br>
            prix :
            <%# databinder.eval(Container.DataItem,"prix","{0:C}") %>
            <br>
          </ItemTemplate>
          <FooterStyle BackColor="Tan"></FooterStyle>
          <HeaderStyle Font-Bold="True" BackColor="Tan"></HeaderStyle>
        </asp:DataList></P>
    </asp:Panel>
```

```
    <asp:Panel id="vueRésultats2" runat="server">
      <P>
        <asp:DataList id="DataList2" runat="server">
          <HeaderTemplate>
            Contenu de la table [liste] de la base [produits]
            <HR width="100%" SIZE="1">
          </HeaderTemplate>
          <AlternatingItemStyle BackColor="Teal"></AlternatingItemStyle>
          <SeparatorStyle BackColor="LightSkyBlue"></SeparatorStyle>
          <ItemStyle BackColor="#C0C000"></ItemStyle>
          <ItemTemplate>
            nom :
            <%# Container.DataItem("nom") %>
            , prix :
            <%# databinder.eval(Container.DataItem,"prix","{0:C}") %>
            <BR>
          </ItemTemplate>
          <SeparatorTemplate>
            <HR width="100%" SIZE="1">
          </SeparatorTemplate>
          <HeaderStyle BackColor="#C0C0FF"></HeaderStyle>
        </asp:DataList></P>
    </asp:Panel>
    <asp:Panel id="vueErreurs" runat="server">
      <P>Les erreurs suivantes se sont produites :</P>
      <%= erreursHTML %>
    </asp:Panel>
```

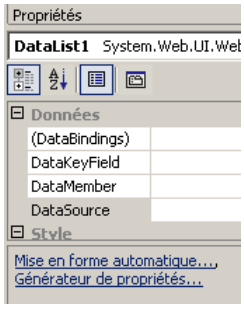
```
  </form>
 </body>
</HTML>
```

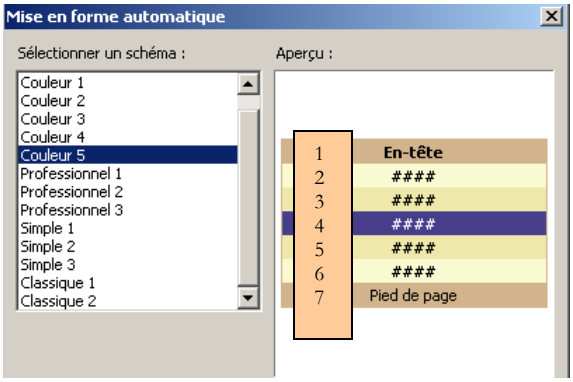
2.7.4 Configuration des composants [DataList]

Intéressons-nous aux différents attributs d'un composant [DataList]. Ils sont très nombreux et nous n'en présentons qu'une faible partie. On peut définir jusqu'à sept modèles d'affichage à l'intérieur d'un [DataList] :

`HeaderTemplate`	modèle de l'entête du [DataList]
`ItemTemplate`	modèle des lignes affichant les éléments de la liste de données associée. Seul ce modèle est obligatoire.
`AlternatingItemTemplate`	pour différentier visuellement les éléments successifs affichés, on peut utiliser deux modèles : ItemTemplate pour l'élément n, AlternatingItemTemplate pour l'élément n+1
`SelectedItemTemplate`	modèle de l'élément sélectionné dans le [DataList]
`SeparatorTemplate`	modèle du séparateur entre deux éléments du [DataList]
`EditItemTemplate`	un [DataList] rend possible la modification des valeurs qu'il affiche. [EditItemTemplate] est le modèle d'un élément du [DataList] pour lequel on est en mode "édition"
`FooterTemplate`	modèle du pied de page du [DataList]

Le composant [DataList1] a été construit avec [WebMatrix]. Dans sa fenêtre de propriétés on a sélectionné le lien [Mise en forme automatique] :





Ci-dessus, le schéma [Couleur 5] va générer un [DataList] avec des styles pour les modèles suivants : HeaderTemplate (1), ItemTemplate (2, 6), AlternatingTemplate (3, 5), SelectedItemTemplate (4), FooterTemplate (7). Le code généré est le suivant :

```
        <asp:DataList id="DataList1" runat="server" BorderColor="Tan" BorderWidth="1px"
BackColor="LightGoldenrodYellow"
          CellPadding="2" ForeColor="Black">
          <SelectedItemStyle ForeColor="GhostWhite" BackColor="DarkSlateBlue"></SelectedItemStyle>
          <AlternatingItemStyle BackColor="PaleGoldenrod"></AlternatingItemStyle>
          <FooterStyle BackColor="Tan"></FooterStyle>
          <HeaderStyle Font-Bold="True" BackColor="Tan"></HeaderStyle>
        </asp:DataList></P>
```

Aucun modèle n'est défini. C'est à nous de le faire. Nous définissons les modèles suivants :

HeaderTemplate
```
<HeaderTemplate>
  Contenu de la table [liste] de la base [produits]
</HeaderTemplate>
```

ItemTemplate
```
<ItemTemplate>
  nom :
  <%# Container.DataItem("nom") %>
  <br>
  prix :
  <%# DataBinder.Eval(Container.DataItem,"prix","{0:C}") %>
  <br>
</ItemTemplate>
```

Rappelons que le modèle [ItemTemplate] est celui de l'affichage des éléments de la source de données liée au [DataList]. Cette source de données est un ensemble de lignes de données, chacune comportant une ou plusieurs valeurs. La ligne courante de la source de données est représentée par l'objet [Container.DataItem]. Une telle ligne a des colonnes. [Container.DataItem("col1")] est la valeur de la colonne "col1" de la ligne courante. Pour inclure cette valeur dans le code de présentation, on écrit **<%# Container.DataItem("col") %>**. Parfois, on veut présenter un élément de la ligne courante sous un format spécial. Ici, on veut afficher la colonne "prix" de la ligne courante en euros. On utilise alors la fonction [DataBinder.Eval] qui admet trois paramètres :

- la ligne courante [Container.DataItem]
- le nom de la colonne à formater
- la chaîne de formatage sous la forme {0:format} où [format] est l'un des formats acceptés par la méthode [string.format].

Ainsi le code **<%# DataBinder.Eval(Container.DataItem,"prix",{0:C}) %>** va faire afficher la colonne [prix] de la ligne courante sous forme monétaire (format C=Currency).

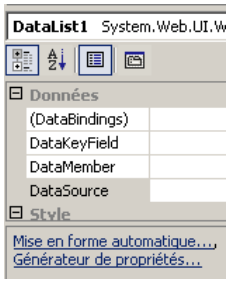
Nous aurons donc un [DataList] qui va ressembler à ceci :

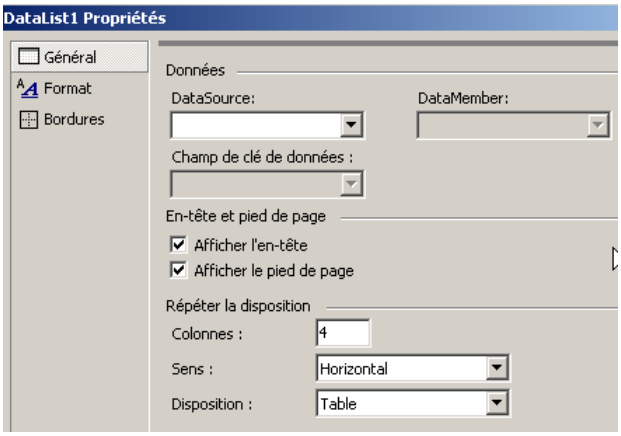
Contenu de la table [liste] de la base [produits]			
nom : produit1 prix : 10,00 €	nom : produit2 prix : 20,00 €	nom : produit3 prix : 30,00 €	nom : produit4 prix : 40,00 €
nom : produit6 prix : 60,00 €	nom : produit7 prix : 70,00 €	nom : produit11 prix : 110,00 €	nom : produit13 prix : 131,00 €

Ci-dessus, les données ont été placées à raison de quatre données par ligne. Cela est obtenu avec les attributs suivants de [DataList] :

RepeatDirection	Horizontal
RepeatColumns	nombre de colonnes désirées

Au final, le code de [DataList1] est celui qui a été présenté dans le code de présentation un peu plus haut. Nous laissons au lecteur le soin d'étudier le code de présentation de [DataList2]. Comme pour le composant [DataGrid], la plupart des propriétés de [DataList] peuvent être fixées à l'aide d'un assistant de [WebMatrix]. On utilise pour cela, le lien [Générateur de propriétés] de la fenêtre de propriétés du [DataList] :





2.7.5 Les contrôleurs

Le contrôleur [global.asax, global.asax.vb] est le suivant :

[global.asax]

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

[global.asax.vb]

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration
Imports System.Data
Imports System.Collections

Public Class Global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on crée un objet produits
   Try
     Dim données As DataSet = New
produits(ConfigurationSettings.AppSettings("OLEDBStringConnection")).getDataSet("select * from LISTE")
     ' on met l'objet dans l'application
     Application("données") = données
     ' pas d'erreur
     Application("erreur") = False
   Catch ex As Exception
     'il y a eu erreur, on le note dans l'application
     Application("erreur") = True
     Application("message") = ex.Message
   End Try
 End Sub
End Class
```

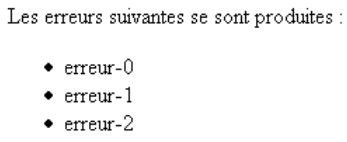
Lorsque l'application démarre (Application_Start), nous construisons un [dataset] à partir de la classe métier [produits] et le mettons dans l'application afin qu'il soit disponible pour toutes les requêtes de tous les clients. S'il y a exception lors de cette construction, elle est notée dans l'application. La procédure [Application_Start] ne s'exécutera qu'une fois. Ensuite le contrôleur [global.asax] n'interviendra plus. C'est le contrôleur [main.aspx.vb] qui fera alors le travail :

```
Imports System.Collections
Imports Microsoft.VisualBasic
Imports System.Data
Imports st.istia.univangers.fr
```

```
Imports System
Imports System.Xml

Public Class main
 Inherits System.Web.UI.Page

 ' composants page
 Protected WithEvents vueErreurs As System.Web.UI.WebControls.Panel
 Protected WithEvents DataList1 As System.Web.UI.WebControls.DataList
 Protected WithEvents DataList2 As System.Web.UI.WebControls.DataList
 Protected WithEvents vueRésultats1 As System.Web.UI.WebControls.Panel
 Protected WithEvents vueRésultats2 As System.Web.UI.WebControls.Panel
 Protected WithEvents RadioButton1 As System.Web.UI.WebControls.RadioButton
 Protected WithEvents RadioButton2 As System.Web.UI.WebControls.RadioButton
 Protected WithEvents btnChanger As System.Web.UI.WebControls.Button
 Protected WithEvents bandeau As System.Web.UI.WebControls.Panel
 ' données page
 Protected erreursHTML As String

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' on regarde si l'application est en erreur
   If CType(Application("erreur"), Boolean) Then
     ' l'application ne s'est pas initialisée correctement
     Dim erreurs As New ArrayList
     erreurs.Add("Application momentanément indisponible (" + CType(Application("message"), String) +
")")
     afficheErreurs(erreurs, False)
     Exit Sub
   End If
   '1ère requête
   If Not IsPostBack Then
     ' on initialise les contrôles
     With DataList1
       .DataSource = CType(Application("données"), DataSet)
       .DataBind()
     End With
     With DataList2
       .DataSource = CType(Application("données"), DataSet)
       .DataBind()
     End With
     ' on affiche le formulaire vide
     afficheRésultats(True, False)
   End If
 End Sub

 Private Sub afficheErreurs(ByVal erreurs As ArrayList, ByVal afficheLien As Boolean)
   ' affiche la vue erreurs
   erreursHTML = ""
   For i As Integer = 0 To erreurs.Count - 1
     erreursHTML += "<li>" + erreurs(i).ToString + "</li>" + ControlChars.CrLf
   Next
   ' on affiche la vue [erreurs]
   vueErreurs.Visible = True
   vueRésultats1.Visible = False
   vueRésultats2.Visible = False
   bandeau.Visible = False
 End Sub

 Private Sub afficheRésultats(ByVal visible1 As Boolean, ByVal visible2 As Boolean)
   ' on affiche la vue [résultats]
   vueRésultats1.Visible = visible1
   vueRésultats2.Visible = visible2
   vueErreurs.Visible = False
 End Sub

 Private Sub btnChanger_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnChanger.Click
   ' on change le style
   afficheRésultats(RadioButton1.Checked, RadioButton2.Checked)
 End Sub
End Class
```

2.8 Composant Repeater et liaison de données

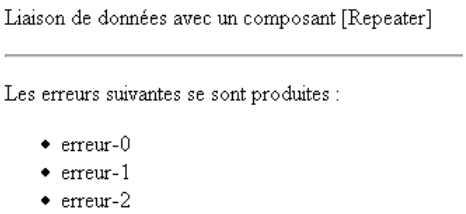
Le composant [Repeater] permet de répéter le code HTML de chacun des éléments d'une liste de données. Supposons qu'on veuille afficher une liste d'erreurs sous la forme suivante :



Nous avons déjà rencontré ce problème et nous l'avons résolu en mettant dans le code de présentation une variable sous la forme **<% =erreursHTML %>
, la valeur de **erreursHTML étant calculée par le contrôleur. Cette valeur contient du code HTML, celle d'une liste. Cela présente l'inconvénient que si on veut modifier la présentation de cette liste HTML on est obligé d'aller dans la partie contrôleur, ce qui va à l'encontre de la séparation contrôleur/présentation. Le composant [Repeater] nous apporte une solution. Comme pour le [DataList], on peut définir les modèles <HeaderTemplate> pour l'entête, <ItemTemplate> pour l'élément courant de la liste de données et <FooterTemplate> pour la fin des données. Ici, nous pourrions avoir la définition suivante pour le composant [Repeater] :

```
            <asp:Repeater id="Repeater1" runat="server" EnableViewState="False">
                <HeaderTemplate>
                    Les erreurs suivantes se sont produites :
                    <ul>
                </HeaderTemplate>
                <ItemTemplate>
                    <li>
                        <%# Container.DataItem %>
                    </li>
                </ItemTemplate>
                <FooterTemplate>
                    </ul>
                </FooterTemplate>
            </asp:Repeater>
```

On rappelle que [Container.DataItem] représente une ligne de données si la source de données a plusieurs colonnes. Elle représente une donnée si la source n'a qu'une colonne. Ce sera le cas ici. Pour exemple, nous construisons l'application suivante :



Le code de présentation de la page est le suivant :

```
<html>
<head>
</head>
<body>
    <form runat="server">
        <p>
            Liaison de données avec un composant [Repeater]
        </p>
        <hr />
        <p>
            <asp:Repeater id="Repeater1" runat="server" EnableViewState="False">
                <ItemTemplate>
                    <li>
                        <%# Container.DataItem %>
                    </li>
                </ItemTemplate>
                <HeaderTemplate>
                    Les erreurs suivantes se sont produites :
                    <ul>
                </HeaderTemplate>
                <FooterTemplate>
                    </ul>
                </FooterTemplate>
            </asp:Repeater>
        </p>
    </form>
</body>
</html>
```

Le code de contrôle est le suivant :

```
<%@ Page Language="VB" %>
<script runat="server">

    ' procédure exécutée au chargement de la page
    Sub page_Load(sender As Object, e As EventArgs)
        if not IsPostBack then
          ' on crée une source de données
          with Repeater1
              .DataSource=createDataSource
              .DataBind
          end with
        end if
    End Sub

    function createDataSource as ArrayList
      ' crée un arraylist
      dim erreurs as new ArrayList
      dim i as integer
      for i=0 to 5
        erreurs.add("erreur-"+i.ToString)
      next
      return erreurs
    end function

</script>
<html>
...
</html>
```

A la première requête du client, nous associons un objet [ArrayList] au composant [Repeater] supposé représenter une liste d'erreurs.

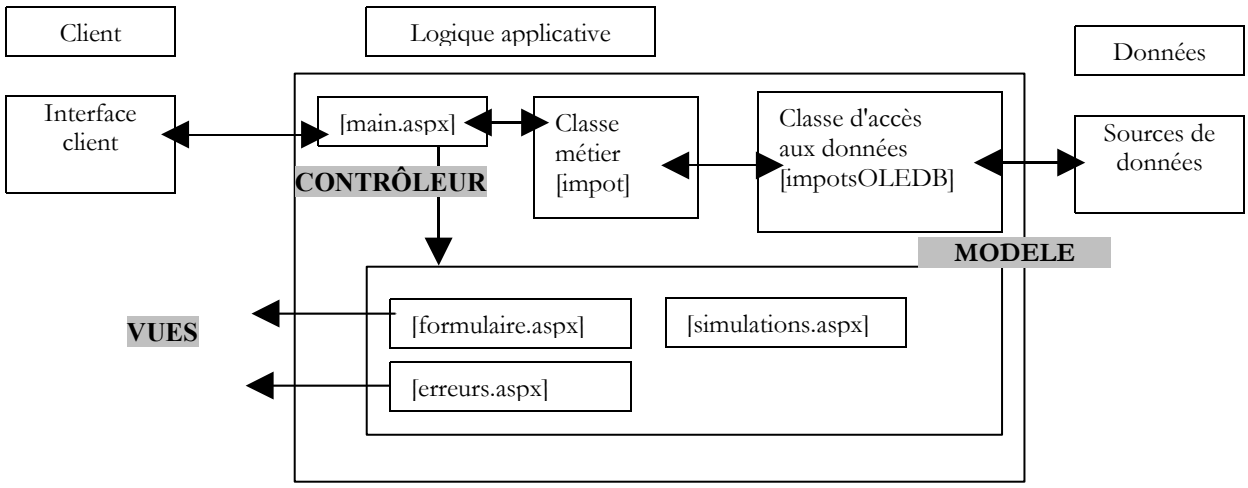
2.9 Application

Nous reprenons ici, une application déjà traitée avec des composants serveur. L'application permet de faire des simulations de calculs d'impôt. Elle s'appuie sur une classe [impot] qu'on ne rappellera pas ici. Cette classe a besoin de données qu'elle trouve dans une source de données OLEDB. Pour l'exemple, ce sera une source ACCESS. Nous amenons les nouveautés suivantes dans cette nouvelle version :

- l'usage de composants de validation pour vérifier la validité des données
- l'usage de composants serveur liés à des sources de données pour l'affichage des résultats

2.9.1 La structure MVC de l'application

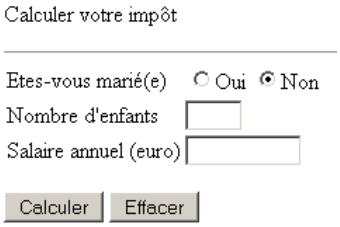
La structure MVC de l'application est la suivante :



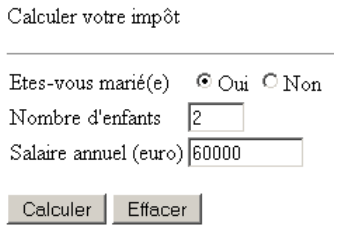
Les trois vues seront incorporées dans le code de présentation du contrôleur [main.aspx] sous la forme de conteneurs. Aussi cette application a-t-elle une unique page [main.aspx].

2.9.2 Les vues de l'application

La vue [formulaire] est le formulaire de saisie des informations permettant le calcul de l'impôt d'un utilisateur :

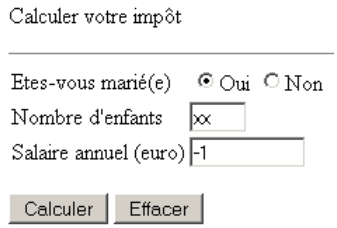


L'utilisateur remplit le formulaire :

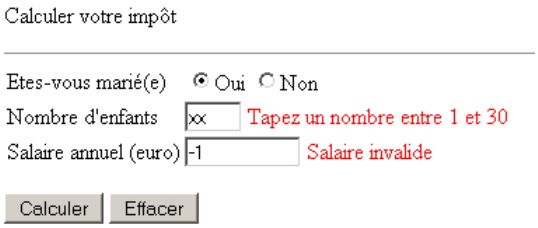


Il utilise le bouton [Envoyer] pour demander le calcul de son impôt. Il obtient la vue [simulations] suivante :

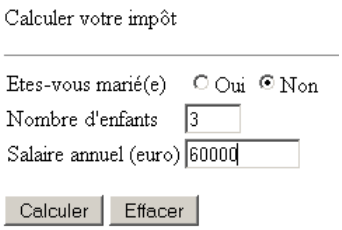
Il retourne au formulaire par le lien ci-dessus. Il le retrouve dans l'état où il l'a saisi. Il peut faire des erreurs de saisie :



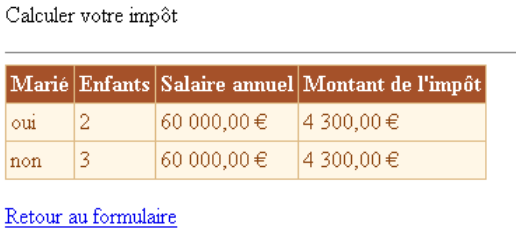
Celles-ci lui sont signalées sur la vue [formulaire] :



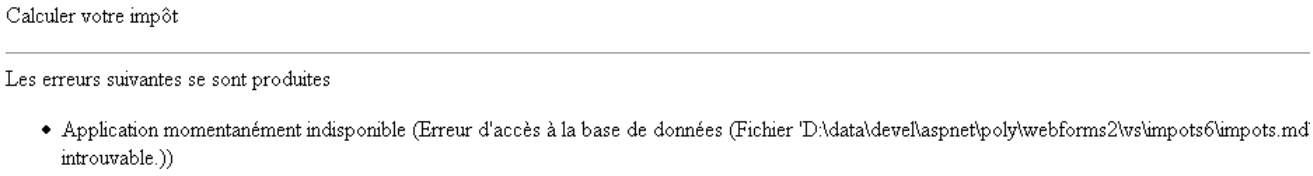
L'utilisateur corrige alors ses erreurs. Il peut faire de nouvelles simulations :



Il obtient alors la vue [simulations] avec une simulation de plus :



Enfin, si la source de données n'est pas disponible, cela est signalé à l'utilisateur dans la vue [erreurs] :

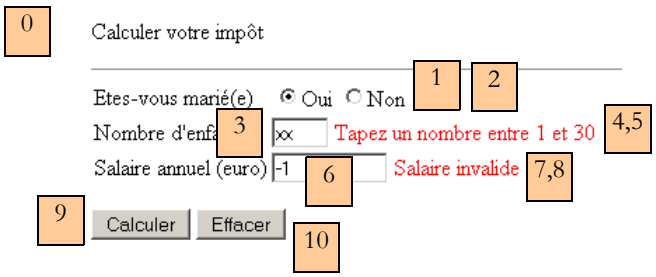


2.9.2.1 Le code de présentation

Rappelons que la page [main.aspx] rassemble toutes les vues. C'et un unique formulaire avec trois conteneurs :

- [panelform] pour la vue [formulaire]
- [panelerreurs] pour la vue [erreurs]
- [panelsimulations] pour la vue [simulations]

Nous détaillons maintenant les composants de ces trois conteneurs. Le conteneur [panelform] a la représentation visuelle suivante :

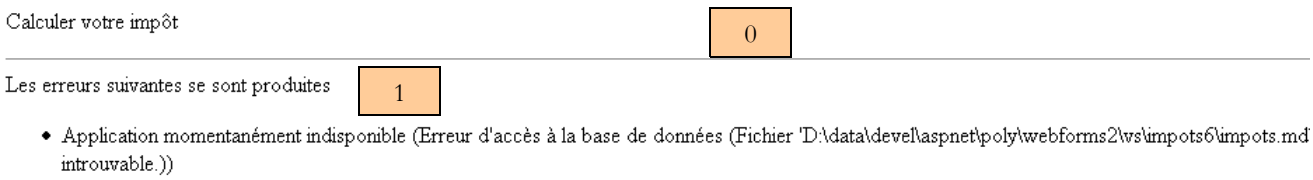


n°	nom	type	propriétés	rôle
0	panelform	Panel		vue formulaire
1	rdOui rdNon	RadioButton	GroupName=rdmarie	boutons radio
2	cvMarie	CustomValidator	ErrorMessage=Vous n'avez pas indiqué votre état marital EnableClientScript=false	vérifie que le programme client a bien envoyé le statut marital attendu
3	txtEnfants	TextBox		nombre d'enfants
4	rfvEnfants	RequiredFieldValidator	ErrorMessage=Indiquez le nombre d'enfants	vérifie que le champ [txtEnfants] n'est pas vide

n°	nom	type	propriétés	rôle
			ControlToValidate=txtEnfants	
5	rvEnfants	RangeValidator	ErrorMessage=Tapez un nombre entre 1 et 30 ControlToValidate=txtEnfants	vérifie que le champ [txtEnfants] est dans l'intervalle [1,30]
6	txtSalaire	TextBox	EnableViewState=true	salaire annuel
7	rfvSalaire	RequiredFieldValidator	ControlToValidate=txtSalaire ErrorMessage=Indiquez le montant de votre salaire	vérifie que le champ [txtSalaire] n'est pas vide
8	revSalaire	RegularExpressionValidator	ControlToValidate=txtSalaire ErrorMessage=Salaire invalide RegularExpression=\s*\d+\s*	vérifie que le champ [txtSalaire] est une suite de chiffes
9	btnCalculer	Button	CausesValidation=true	bouton [submit] du formulaire - lance le calcul de l'impôt
10	btnEffacer	Button	CausesValidation=false	bouton [submit] du formulaire - vide le formulaire

On pourra s'étonner du contrôle [cvMarié] chargé de vérifier que l'utilisateur a bien coché l'un des deux boutons radio. Il ne peut en effet en être autrement s'il utilise bien le formulaire envoyé par le serveur. Comme rien ne peut l'assurer, on est obligé de vérifier tous les paramètres postés. On notera également l'attribut [CausesValidation=false] du bouton [btnEffacer]. En effet, lorsque l'utilisateur utilise ce bouton on ne doit pas vérifier les données postées puisqu'elles vont être ignorées.

Tous les composants du conteneur ont la propiété [EnableViewState=false] sauf [panelForm, txtEnfants, txtSalaire, rdOui, rdNon]. Le conteneur [panelerreurs] a la représentation visuelle suivante :

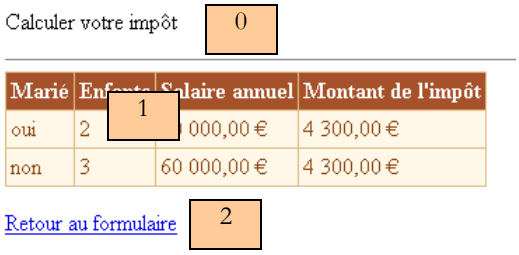


n°	nom	type	propriétés	rôle
0	panelerreurs	Panel	EnableViewState=false	vue erreurs
1	rptErreurs	Repeater	EnableViewState=false	affiche une liste d'erreurs

Le composant [rptErreurs] est défini de la façon suivante :

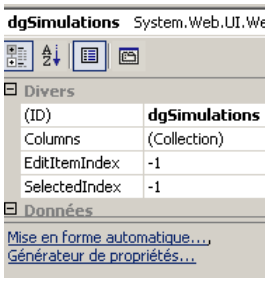
```
<asp:Repeater id="rptErreurs" runat="server" EnableViewState="False">
  <ItemTemplate>
    <li>
      <%# Container.Dataitem%>
    </li>
  </ItemTemplate>
  <HeaderTemplate>
    Les erreurs suivantes se sont produites
    <ul>
  </HeaderTemplate>
  <FooterTemplate>
    </ul>
  </FooterTemplate>
</asp:Repeater>
```

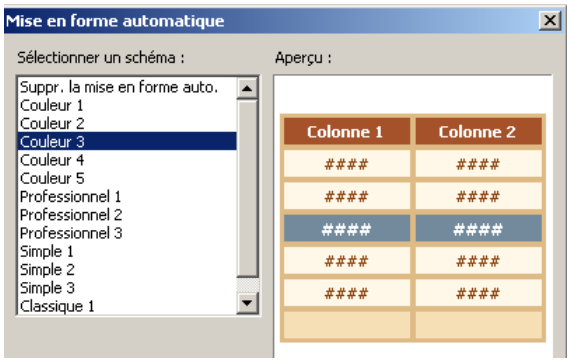
La liste de données associée au composant [rptErreurs] sera un objet [ArrayList] contenant une liste de messages d'erreurs. Ainsi ci-dessus, <%# Container.Dataitem%> désigne le message d'erreur courant. Le conteneur [panelsimulations] a la représentation visuelle suivante :



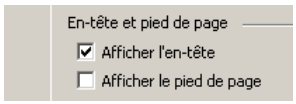
n°	nom	type	propriétés	rôle
0	panelsimulations	Panel	EnableViewState=false	vue simulations
2	lnkForm2	LinkButton	EnableViewState=false	lien vers le formulaire
1	dgSimulations	DataGrid	EnableViewState=false	chargé d'afficher les simulations placées dans un objet [DataTable]

Le composant [dgSimulations] est configuré de la façon suivante sous [Webmatrix]. Dans la fenêtre de propriétés de [dgSimulations], nous sélectionnons le lien [Mise en forme automatique] et sélectionnons le schéma [Couleur 3] :

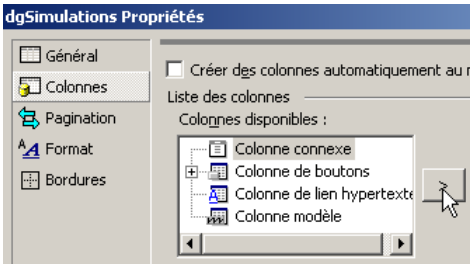




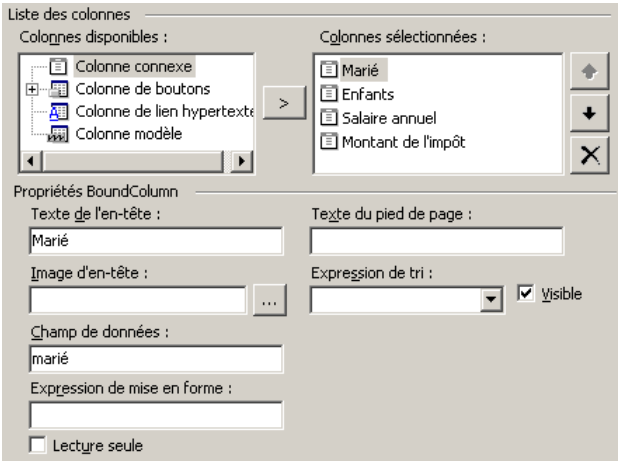
Puis, toujours dans la fenêtre de propriétés de [dlgSimulations], nous sélectionnons le lien [Générateur de propriétés]. Nous demandons l'affichage de l'en-tête des colonnes :



Nous sélectionnons l'option [Colonnes] pour définir les quatre colonnes du [DataGrid] :



Nous décochons la phrase [Créer des colonnes automatiquement ...]. Nous allons définir nous-mêmes les quatre colonnes du [DataGrid]. Celui-ci sera associé à un objet [DataTable] à quatre colonnes nommées respectivement "marié", "enfants", "salaire", "impôt". Pour créer une colonne dans le [DataGrid], nous sélectionnons l'option [Colonnes connexe] et utilisons le bouton de création comme indiqué ci-dessus. Une colonne est créée et nous pouvons définir ses caractéristiques. La première colonne du tableau des simulations contiendra la colonne "marié" de l'objet [DataTable] qui sera associé au [DataGrid]. Cette première colonne du [DataGrid] est définie comme suit dans l'assistant :



`Texte de l'en-tête`	titre de la colonne, ici "Marié"
`Champ de données`	nom de la colonne de la source de données qui sera visualisée par cette colonne du [DataGrid]. Ici, c'est la colonne "marié" du [DataTable].

La deuxième colonne est définie comme suit :

`Texte de l'en-tête`	Enfants
`Champ de données`	enfants

La troisème colonne est définie comme suit :

`Texte de l'en-tête`	Salaire annuel
`Champ de données`	salaire
`Expression de mise en forme`	{0:C} - affichage monétaire pour obtenir le signe euro

La quatrième colonne est définie comme suit :

`Texte de l'en-tête`	Montant de l'impôt
`Champ de données`	impôt
`Expression de mise en forme`	{0:C} - affichage monétaire pour obtenir le signe euro

Nous mettons le code de de présentation et le code de contrôle dans deux fichiers séparés. Le premier sera dans [main.aspx] et le second dans [main.aspx.vb]. Le code de [main.aspx] est le suivant :

```
<%@ page src="main.aspx.vb" inherits="main" AutoEventWireUp="false" %>
<HTML>
 <HEAD>
   <title>Calculer votre impôt</title>
 </HEAD>
 <body>
   <P>Calculer votre impôt</P>
   <HR width="100%" SIZE="1">
   <FORM id="Form1" runat="server">

     <asp:panel id="panelform" Runat="server">
       <TABLE id="Table1" cellSpacing="1" cellPadding="1" border="0">
         <TR>
           <TD height="19">Etes-vous marié(e)</TD>
           <TD height="19">
             <asp:RadioButton id="rdOui" runat="server" GroupName="rdMarie"></asp:RadioButton>Oui
             <asp:RadioButton id="rdNon" runat="server" GroupName="rdMarie"
Checked="True"></asp:RadioButton>Non
             <asp:CustomValidator id="cvMarie" runat="server" ErrorMessage="Vous n'avez pas indiqué votre
état marital"
               Display="Dynamic" EnableClientScript="False" Visible="False"
EnableViewState="False"></asp:CustomValidator></TD>
```

```
        </TR>
        <TR>
          <TD>Nombre d'enfants</TD>
          <TD>
            <asp:TextBox id="txtEnfants" runat="server" Columns="3" MaxLength="3"></asp:TextBox>
            <asp:RequiredFieldValidator id="rfvEnfants" runat="server" ErrorMessage="Indiquez le nombre
d'enfants" Display="Dynamic"
              ControlToValidate="txtEnfants" EnableViewState="False"></asp:RequiredFieldValidator>
            <asp:RangeValidator id="rvEnfants" runat="server" ErrorMessage="Tapez un nombre entre 1 et
30" Display="Dynamic"
              ControlToValidate="txtEnfants" Type="Integer" MaximumValue="30" MinimumValue="0"
EnableViewState="False"></asp:RangeValidator></TD>
        </TR>
        <TR>
          <TD>Salaire annuel (euro)</TD>
          <TD>
            <asp:TextBox id="txtSalaire" runat="server" Columns="10" MaxLength="10"></asp:TextBox>
            <asp:RequiredFieldValidator id="rfvSalaire" runat="server" ErrorMessage="Indiquez le montant
de votre salaire"
              Display="Dynamic" ControlToValidate="txtSalaire"
EnableViewState="False"></asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator id="revSalaire" runat="server" ErrorMessage="Salaire
invalide" Display="Dynamic"
              ControlToValidate="txtSalaire" ValidationExpression="\s*\d+\s*"
EnableViewState="False"></asp:RegularExpressionValidator></TD>
        </TR>
      </TABLE>
      <P>
        <asp:Button id="btnCalculer" runat="server" Text="Calculer"></asp:Button>
        <asp:Button id="btnEffacer" runat="server" Text="Effacer"
CausesValidation="False"></asp:Button></P>
    </asp:panel>
```

```
      <asp:panel id="panelerreurs" runat="server" EnableViewState="False">
      <asp:Repeater id="rptErreurs" runat="server" EnableViewState="False">
        <ItemTemplate>
          <li>
            <%# Container.Dataitem%>
          </li>
        </ItemTemplate>
        <HeaderTemplate>
          Les erreurs suivantes se sont produites
          <ul>
        </HeaderTemplate>
        <FooterTemplate>
          </ul>
        </FooterTemplate>
      </asp:Repeater>
    </asp:panel>
```

```
    <asp:panel id="panelsimulations" runat="server" EnableViewState="False">
      <P>
        <asp:DataGrid id="dgSimulations" runat="server" BorderColor="#DEBA84" BorderStyle="None"
CellSpacing="2"
          BorderWidth="1px" BackColor="#DEBA84" CellPadding="3" AutoGenerateColumns="False"
EnableViewState="False">
          <SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#738A9C"></SelectedItemStyle>
          <ItemStyle ForeColor="#8C4510" BackColor="#FFF7E7"></ItemStyle>
          <HeaderStyle Font-Bold="True" ForeColor="White" BackColor="#A55129"></HeaderStyle>
          <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>
          <Columns>
            <asp:BoundColumn DataField="mari&#233;" HeaderText="Mari&#233;"></asp:BoundColumn>
            <asp:BoundColumn DataField="enfants" HeaderText="Enfants"></asp:BoundColumn>
            <asp:BoundColumn DataField="salaire" HeaderText="Salaire annuel"
DataFormatString="{0:C}"></asp:BoundColumn>
            <asp:BoundColumn DataField="imp&#244;t" HeaderText="Montant de l'imp&#244;t"
DataFormatString="{0:C}"></asp:BoundColumn>
          </Columns>
          <PagerStyle HorizontalAlign="Center" ForeColor="#8C4510" Mode="NumericPages"></PagerStyle>
        </asp:DataGrid></P>
      <P>
        <asp:LinkButton id="lnkForm2" runat="server" EnableViewState="False">Retour au
formulaire</asp:LinkButton></P>
    </asp:panel>
```

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

2.9.3 Le code de contrôle de l'application

Le code de contrôle de l'application se trouve réparti dans les fichiers [global.asax.vb] et [main.aspx.vb]. Le fichier [global.asax] est défini comme suit :

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

Le fichier [global.asax.vb] est le suivant :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration
Imports System.Collections
Imports System.Data

Public Class Global
    Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on crée un objet impot
   Dim objImpot As impot
   Try
     objImpot = New impot(New impotsOLEDB(ConfigurationSettings.AppSettings("chaineConnexion")))
     ' on met l'objet dans l'application
     Application("objImpot") = objImpot
     ' pas d'erreur
     Application("erreur") = False
   Catch ex As Exception
     'il y a eu erreur, on le note dans l'application
     Application("erreur") = True
     Application("message") = ex.Message
   End Try
 End Sub

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' début de session - on crée une liste de simulations vides
   Dim simulations As New DataTable("simulations")
   simulations.Columns.Add("marié", Type.GetType("System.String"))
   simulations.Columns.Add("enfants", Type.GetType("System.String"))
   simulations.Columns.Add("salaire", Type.GetType("System.Int32"))
   simulations.Columns.Add("impôt", Type.GetType("System.Int32"))
   Session.Item("simulations") = simulations
 End Sub
End Class
```

Lorsque l'application démarre (1ère requête faite à l'application), la procédure [Application_Start] est exécutée. Elle cherche à créer un objet de type [impot] prenant ses données dans une source OLEDB. Le lecteur est invité à lire le chapitre 5 où cette classe a été définie, s'il l'a oubliée. La construction de l'objet [impot] peut échouer si la source de données n'est pas disponible. Dans ce cas, l'erreur est mémorisée dans l'application afin que toutes les requêtes ultérieures sachent que celle-ci n'a pu s'initialiser correctement. Si la construction se passe bien, l'objet [impot] créé est lui aussi mémorisé dans l'application. Il sera utilisé par toutes les requêtes de calcul d'impôt. Lorsqu'un client fait sa première requête, une session est créée pour lui par la procédure [Application_Start]. Cette session est destinée à mémoriser les différentes simulations de calcul d'impôt qu'il va faire. Celles-ci seront mémorisées dans un objet [DataTable] associé à la clé de session "simulations". Lorsque la session démarre, cette clé est associée à un objet [DataTable] vide mais dont la structure a été définie. Les informations nécessaires à l'application sont placées dans son fichier de configuration [web.config] :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <appSettings>
   <add key="chaineConnexion" value="Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data
Source=D:\data\devel\aspnet\poly\webforms2\vs\impots6\impots.mdb" />
 </appSettings>
</configuration>
```

La clé [chaineConnexion] désigne la chaîne de connexion à la source OLEDB. L'autre partie du code de contrôle se trouve dans [main.aspx.vb] :

```
Imports System.Collections
Imports Microsoft.VisualBasic
Imports st.istia.univangers.fr
Imports System
Imports System.Web.UI.Control
Imports System.Data
```

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

 Protected WithEvents rdOui As System.Web.UI.WebControls.RadioButton
 Protected WithEvents rdNon As System.Web.UI.WebControls.RadioButton
 Protected WithEvents txtEnfants As System.Web.UI.WebControls.TextBox
 Protected WithEvents txtSalaire As System.Web.UI.WebControls.TextBox
 Protected WithEvents btnCalculer As System.Web.UI.WebControls.Button
 Protected WithEvents btnEffacer As System.Web.UI.WebControls.Button
 Protected WithEvents panelform As System.Web.UI.WebControls.Panel
 Protected WithEvents lnkForm2 As System.Web.UI.WebControls.LinkButton
 Protected WithEvents panelerreurs As System.Web.UI.WebControls.Panel
 Protected WithEvents rfvEnfants As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents rvEnfants As System.Web.UI.WebControls.RangeValidator
 Protected WithEvents rfvSalaire As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents rptErreurs As System.Web.UI.WebControls.Repeater
 Protected WithEvents dgSimulations As System.Web.UI.WebControls.DataGrid
 Protected WithEvents revSalaire As System.Web.UI.WebControls.RegularExpressionValidator
 Protected WithEvents cvMarie As System.Web.UI.WebControls.CustomValidator
 Protected WithEvents panelsimulations As System.Web.UI.WebControls.Panel

 ' variables locales

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

 Private Sub afficheErreurs(ByRef erreurs As ArrayList)
...
 End Sub

 Private Sub afficheFormulaire()
...
 End Sub

 Private Sub afficheSimulations(ByRef simulations As DataTable, ByRef lien As String)
...
 End Sub

 Private Sub btnCalculer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCalculer.Click
....
 End Sub

 Private Sub lnkForm1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
...
 End Sub

 Private Sub lnkForm2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm2.Click
...
 End Sub

 Private Sub btnEffacer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnEffacer.Click
...
 End Sub

 Private Sub razForm()
...
 End Sub

 Private Sub cvMarie_ServerValidate(ByVal source As System.Object, ByVal args As
System.Web.UI.WebControls.ServerValidateEventArgs)
...
 End Sub
End Class
```

Le premier événement traité par le code est [Page_Load] :

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' tout d'abord, on regarde dans quel état est l'application
   If CType(Application("erreur"), Boolean) Then
     ' l'application n'a pas pu s'initialiser
     ' on affiche la vue erreurs
     Dim erreurs As New ArrayList
     erreurs.Add("Application momentanément indisponible (" + CType(Application("message"), String) +
")")
     afficheErreurs(erreurs)
     Exit Sub
```

```
    End If
    ' pas d'erreurs - à la 1ère requête, on présente le formulaire
    If Not IsPostBack Then afficheFormulaire()
  End Sub
```

Avant de commencer à traiter la requête, nous nous assurons que l'application a pu s'initialiser correctement. Si ce n'est pas le cas, on affiche la vue [erreurs] avec la procédure [afficheErreurs]. Si c'est la première requête (IsPostBack=false), nous faisons afficher la vue [formulaire] avec [afficheFormulaire].

La procédure affichant la vue [erreurs] est la suivante :

```
  Private Sub afficheErreurs(ByRef erreurs As ArrayList)
    ' construit la liste des erreurs
    With rptErreurs
      .DataSource = erreurs
      .DataBind()
    End With
    ' affichage des conteneurs
    panelerreurs.Visible = True
    panelform.Visible = False
    panelsimulations.Visible = False
    Exit Sub
  End Sub
```

La procédure reçoit comme paramètre une liste de messages d'erreurs dans [erreurs] de type [ArrayList]. Nous nous contentons de lier cette source de données au composant [rptErreurs] chargé de l'afficher.

La procédure affichant la vue [formulaire] est la suivante :

```
  Private Sub afficheFormulaire()
    ' affiche le formulaire
    panelform.Visible = True
    ' les autres conteneurs sont cachés
    panelerreurs.Visible = False
    panelsimulations.Visible = False
  End Sub
```

Cette procédure se contente de rendre visible le conteneur [panelform]. Les composants sont affichés avec leur valeur postée ou précédente (VIEWSTATE).

Lorsque l'utilisateur clique sur le bouton [Calculer] de la vue [formulaire], un POST vers [main.aspx] est fait. La procédure [Page_Load] est exécutée puis la procédure [btnCalculer_Click] :

```
  Private Sub btnCalculer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnCalculer.Click
    ' on vérifie la validité des données saisies, s'il y a des erreurs, on le signale
    If Not Page.IsValid Then
      afficheFormulaire()
      Exit Sub
    End If
    ' pas d'erreurs - on calcule l'impôt
    Dim impot As Long = CType(Application("objImpot"), impot).calculer( _
    rdOui.Checked, CType(txtEnfants.Text, Integer), CType(txtSalaire.Text, Long))
    ' on rajoute le résultat aux simulations existantes
    Dim simulations As DataTable = CType(Session.Item("simulations"), DataTable)
    Dim simulation As DataRow = simulations.NewRow
    simulation("marié") = CType(IIf(rdOui.Checked, "oui", "non"), String)
    simulation("enfants") = txtEnfants.Text.Trim
    simulation("salaire") = CType(txtSalaire.Text.Trim, Long)
    simulation("impôt") = impot
    simulations.Rows.Add(simulation)
    ' on affiche la page des simulations
    afficheSimulations(simulations, "Retour au formulaire")
  End Sub
```

La procédure commence par vérifier la validité de la page. Rappelons ici que lorsque la procédure [btnCalculer_Click] s'exécute, les contrôles de validation ont fait leur travail et l'attribut [IsValid] de la page a été positionné. Si la page n'est pas valide, alors la vue [formulaire] est affichée de nouveau et la procédure est terminée. Les composants de validation de la vue [formulaire] ayant leur attribut [IsValid] à [false] afficheront leur attribut [ErrorMessage]. Si la page est valide, alors le montant de l'impôt est calculé grâce à l'objet de type [impot] qui avait été stocké dans l'application au démarrage de celle-ci. Cette nouvelle simulation est ajoutée à la liste des simulations déjà faites et stockée dans la session.

Enfin la vue [simulations] est affichée par la procédure [afficheSimulations] suivante :

```
Private Sub afficheSimulations(ByRef simulations As DataTable, ByRef lien As String)
  ' on lie le datagrid à la source simulations
  With dgSimulations
    .DataSource = simulations
    .DataBind()
  End With
  ' lien
  lnkForm2.Text = lien
  ' les vues
  panelsimulations.Visible = True
  panelerreurs.Visible = False
  panelform.Visible = False
End Sub
```

La procédure a deux paramètres :

- une liste de simulations dans [simulations] de type [DataTable]
- un texte de lien dans [lien]

La source de données [simulations] est liée au composant chargé de l'afficher. Le texte du lien est lui placé dans la propriété [Text] de l'objet [LinkButton] de la vue.

Lorsque l'utilisateur clique sur le bouton [Effacer] de la vue [formulaire], c'est la procédure [btnEffacer_click] qui s'exécute (toujours après [Page_Load]) :

```
Private Sub btnEffacer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnEffacer.Click
  ' affiche le formulaire vide
  razForm()
  afficheFormulaire()
End Sub

Private Sub razForm()
  ' vide le formulaire
  rdOui.Checked = False
  rdNon.Checked = True
  txtEnfants.Text = ""
  txtSalaire.Text = ""
End Sub
```

Le code ci-dessus est suffisamment simple pour ne pas avoir à être commenté. Il nous reste à gérer le clic sur les liens des vues [erreurs] et [simulations] :

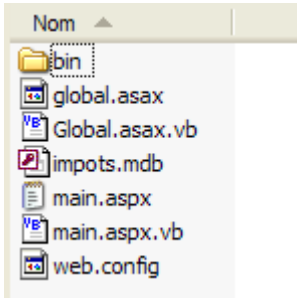
```
Private Sub lnkForm1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm1.Click
  ' affiche le formulaire
  afficheFormulaire()
End Sub

Private Sub lnkForm2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkForm2.Click
  ' affiche le formulaire
  afficheFormulaire()
End Sub
```

Les deux procédures se contentent de faire afficher la vue [formulaire]. Nous savons que les champs de celle-ci vont obtenir une valeur qui est soit la valeur postée pour eux soit leur valeur précédente. Comme ici, le POST du client n'envoie aucune valeur pour les champs du formulaire, ceux-ci vont retrouver leur valeur précédente. Le formulaire est donc affiché avec les valeurs saisies par l'utilisateur.

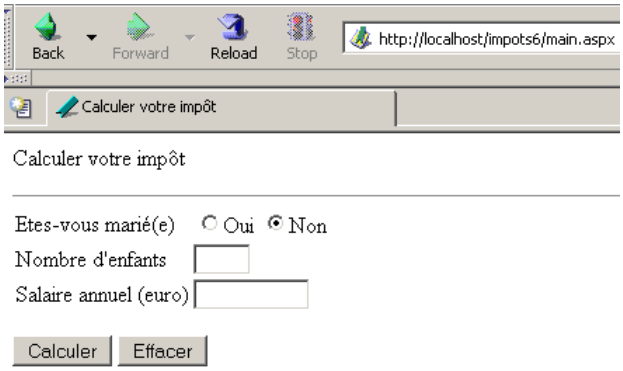
2.9.4 Tests

Tous les fichiers nécessaires à l'application sont placés dans un dossier <application-path> :

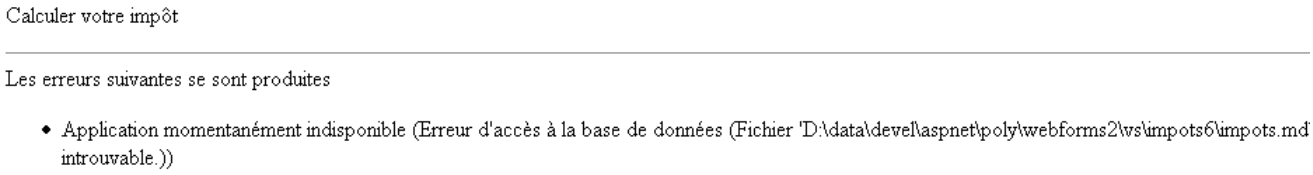
Le dossier [bin] contient la dll contenant les classes [impot], [impotsData], [impotsOLEDB] nécessaires à l'application :



Le lecteur pourra, s'il le souhaite, relire le chapitre 5 où est expliquée la façon de créer le fichier [impot.dll] ci-dessus. Ceci fait, le serveur Cassini est lancé avec les paramètres (<application-path>,/impots6). On demande l'url [http://impots6/main.aspx] avec un navigateur :



Si on renomme le fichier ACCESS [impots.mdb] en [impots1.mdb], on aura la page suivante :



2.9.5 Conclusion

Nous avons une application MVC n'utilisant que des composants serveurs. L'usage de ceux-ci a permis de séparer complètement la partie présentation de l'application de sa partie contrôle. Cela n'avait pas encore été possible jusqu'à maintenant où le code de présentation contenait des variables calculées par le code de contrôle et ayant une valeur contenant du code HTML. Maintenant ce n'est plus le cas.

3 Composants serveur ASP - 3

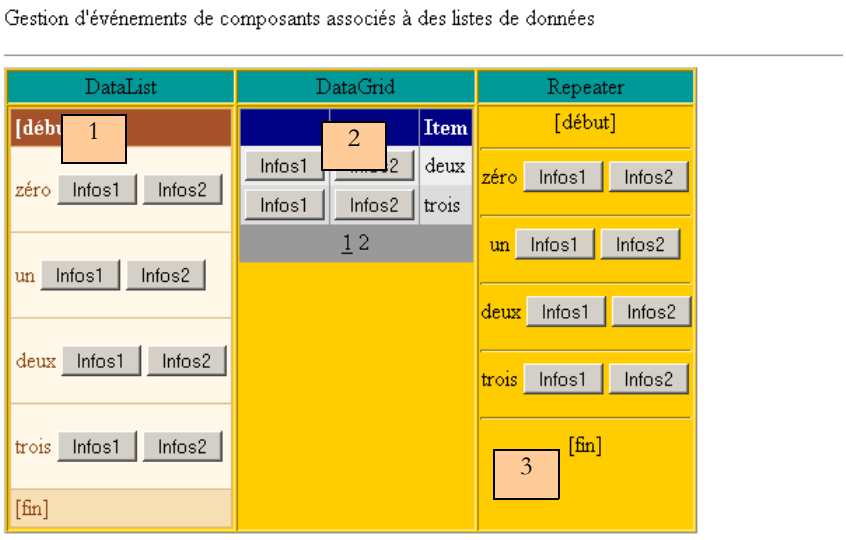
3.1 Introduction

Nous continuons notre travail sur l'interface utilisateur en approfondissant les capacités des composants [DataList], [DataGrid], notamment dans le domaine de la mise à jour des données qu'ils affichent

3.2 Gérer les événements associés aux données des composants à liaison de données

3.2.1 L'exemple

Considérons la page suivante :



La page comprend trois composants associés à une liste de données :

- un composant [DataList] nommé [DataList1]
- un composant [DataGrid] nommé [DataGrid1]
- un composant [Repeater] nommé [Repeater1]

La liste de données associée est le tableau {"zéro","un","deux","trois"}. A chacune de ces données est associé un groupe de deux boutons libellés [Infos1] et [Infos2]. L'utilisateur clique sur l'un des boutons et un texte affiche le nom du bouton cliqué. On cherche à montrer ici comment gérer une liste de boutons ou de liens.

3.2.2 La configuration des composants

Le composant [DataList1] est configuré de la façon suivante :

```
<asp:datalist id="DataList1" ... runat="server">
  <SelectedItemStyle ...</SelectedItemStyle>
  <HeaderTemplate>
    [début]
  </HeaderTemplate>
  <FooterTemplate>
    [fin]
  </FooterTemplate>
  <ItemStyle ...></ItemStyle>
  <ItemTemplate>
    <P><%# Container.DataItem %>
      <asp:Button runat="server" Text="Infos1" CommandName="infos1"></asp:Button>
      <asp:Button runat="server" Text="Infos2" CommandName="infos2"></asp:Button></P>
  </ItemTemplate>
  <FooterStyle ...></FooterStyle>
  <HeaderStyle ...></HeaderStyle>
</asp:datalist>
```

Nous avons omis tout ce qui correspondait à l'apparence du [DataList] pour ne s'intéresser qu'à son contenu :

- la section <HeaderTemplate> définit l'en-tête du [DataList] et la section <FooterTemplate> son pied-de-page.
- la section <ItemTemplate> est le modèle d'affichage utilisé pour chacune des données de la liste de données associée. On y trouve les éléments suivants :
  - la valeur de la donnée courante de la iste des données associée au composant : `<%# Container.DataItem %>`
  - deux boutons libellés respectivement [Infos1] et [Infos2]. La classe [Button] a un attribut [CommandName] qui est utilisé ici. Il va nous permettre de déterminer quel est le bouton à l'origine d'un événement dans le [DataList]. Pour gérer le clic sur les boutons, on n'aura qu'un seul gestionnaire d'événement qui sera lié au [DataList] lui-même et non aux boutons. Ce gestionnaire recevra une information lui indiquant sur quelle ligne du [DataList] s'est produit le clic. L'attribut [CommandName] nous permettra de savoir sur quel bouton de la ligne il s'est produit.

Le composant [Repeater1] est configuré de façon très analogue :

```
<asp:repeater id="Repeater1" runat="server">
```

```
            <HeaderTemplate>
              [début]<hr />
            </HeaderTemplate>
            <FooterTemplate>
              <hr />
              [fin]
            </FooterTemplate>
            <SeparatorTemplate>
              <hr />
            </SeparatorTemplate>
            <ItemTemplate>
              <%# Container.DataItem %>
              <asp:Button runat="server" Text="Infos1" CommandName="infos1"></asp:Button>
              <asp:Button runat="server" Text="Infos2" CommandName="infos2"></asp:Button></P>
            </ItemTemplate>
          </asp:repeater>
```

On a simplement ajouté une section <SeparatorTemplate> pour que les données successives affichées par le composant soient séparées par une barre horizontale.

Enfin, le composant [DataGrid1] est configuré comme suit :

```
          <asp:datagrid id="DataGrid1" ... runat="server" PageSize="2" AllowPaging="True">
            <SelectedItemStyle ...></SelectedItemStyle>
            <AlternatingItemStyle ...></AlternatingItemStyle>
            <ItemStyle ...></ItemStyle>
            <HeaderStyle ...></HeaderStyle>
            <FooterStyle ....></FooterStyle>
            <Columns>
              <asp:ButtonColumn Text="Infos1" ButtonType="PushButton" CommandName="Infos1">
              </asp:ButtonColumn>
              <asp:ButtonColumn Text="Infos2" ButtonType="PushButton" CommandName="Infos2">
              </asp:ButtonColumn>
            </Columns>
            <PagerStyle .... Mode="NumericPages"></PagerStyle>
          </asp:datagrid>
```

Là également, nous avons omis les informations de style (couleurs, largeurs, ...). Nous sommes en mode de génération automatique des colonnes qui est le mode par défaut du [DataGrid]. Cela signifie qu'il y aura autant de colonnes qu'il y en a dans la source de données. Ici, il y en a une. Nous avons ajouté deux autres colonnes balisées par <asp:ButtonColumn>. Nous y définissons des informations analogues à celles définies pour les deux autres composants ainsi que le type de bouton, ici [PushButton]. Le type par défaut est [LinkButton], c.a.d. un lien. Par ailleurs, les données seront paginées [AllowPaging=true] avec une taille de page de deux éléments [PageSize=2].

3.2.3 Le code de présentation de la page

Le code de présentation de notre page exemple a été placé dans un fichier [main.aspx] :

```
<%@ page codebehind="main.aspx.vb" inherits="vs.main" autoeventwireup="false" %>
<HTML>
 <HEAD>
 </HEAD>
 <body>
   <form runat="server">
     <P>Gestion d'événements de composants associés à des listes de données</P>
     <HR width="100%" SIZE="1">
     <table cellSpacing="1" cellPadding="1" bgColor="#ffcc00" border="1">
       <tr>
         <td ...>DataList</td>
         <td ...>DataGrid</td>
         <td ...>Repeater</td>
       </tr>
       <tr>
         <td ...>
```

```
          <asp:datalist id="DataList1" ... runat="server">
            <HeaderTemplate>
              [début]
            </HeaderTemplate>
            <FooterTemplate>
              [fin]
            </FooterTemplate>
            <ItemStyle ...></ItemStyle>
            <ItemTemplate>
              <P><%# Container.DataItem %>
```

```
            <asp:Button runat="server" Text="Infos1" CommandName="infos1"></asp:Button>
            <asp:Button runat="server" Text="Infos2" CommandName="infos2"></asp:Button></P>
          </ItemTemplate>
          <FooterStyle ...></FooterStyle>
          <HeaderStyle ....></HeaderStyle>
        </asp:datalist>

        <P></P>
      </td>
      <td ...>

          <asp:datagrid id="DataGrid1" ... runat="server" PageSize="2" AllowPaging="True">
          <AlternatingItemStyle ...></AlternatingItemStyle>
          <ItemStyle ...></ItemStyle>
          <HeaderStyle ...></HeaderStyle>
          <FooterStyle ...></FooterStyle>
          <Columns>
            <asp:ButtonColumn Text="Infos1" ButtonType="PushButton" CommandName="Infos1">
            </asp:ButtonColumn>
            <asp:ButtonColumn Text="Infos2" ButtonType="PushButton" CommandName="Infos2">
              </asp:ButtonColumn>
          </Columns>
          <PagerStyle ... Mode="NumericPages"></PagerStyle>
        </asp:datagrid>

      </td>
      <td ...>

          <asp:repeater id="Repeater1" runat="server">
          <HeaderTemplate>
            [début]<hr />
          </HeaderTemplate>
          <FooterTemplate>
            <hr />
            [fin]
          </FooterTemplate>
          <SeparatorTemplate>
            <hr />
          </SeparatorTemplate>
          <ItemTemplate>
            <%# Container.DataItem %>
            <asp:Button runat="server" Text="Infos1" CommandName="infos1"></asp:Button>
            <asp:Button runat="server" Text="Infos2" CommandName="infos2"></asp:Button></P>
          </ItemTemplate>
        </asp:repeater>

      </td>
    </tr>
  </table>
  <P><asp:label id="lblInfo" runat="server"></asp:label></P>
  <P></P>
 </form>
</body>
</HTML>
```

Dans le code ci-dessus, nous avons omis le code de mise en forme (couleurs, lignes, tailles, ...)

3.2.4 Le code de contrôle de la page

La code de contrôle de l'application a été placé dans le fichier [main.aspx.vb] :

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

 ' composants page
 Protected WithEvents DataList1 As System.Web.UI.WebControls.DataList
 Protected WithEvents lblInfo As System.Web.UI.WebControls.Label
 Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents Repeater1 As System.Web.UI.WebControls.Repeater

 ' la source de données
 Protected textes() As String = {"zéro", "un", "deux", "trois"}

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
     'liaisons avec source de données
     DataList1.DataSource = textes
```

```
    DataGrid1.DataSource = textes
    Repeater1.DataSource = textes
    Page.DataBind()
  End If
 End Sub

 Private Sub DataList1_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataListCommandEventArgs) Handles DataList1.ItemCommand
  ' un évt s'est produit sur une des lignes du [datalist]
  lblInfo.Text = "Vous avez cliqué sur le bouton [" + e.CommandName + "] de l'élément [" +
e.Item.ItemIndex.ToString + "] du composant [DataList]"
 End Sub

 Private Sub Repeater1_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.RepeaterCommandEventArgs) Handles Repeater1.ItemCommand
  ' un évt s'est produit sur une des lignes du [repeater]
  lblInfo.Text = "Vous avez cliqué sur le bouton [" + e.CommandName + "] de l'élément [" +
e.Item.ItemIndex.ToString + "] du composant [Repeater]"
 End Sub

 Private Sub DataGrid1_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.ItemCommand
  ' un évt s'est produit sur une des lignes du [datagrid]
  lblInfo.Text = "Vous avez cliqué sur le bouton [" + e.CommandName + "] de l'élément [" +
e.Item.ItemIndex.ToString + "] de la page [" + DataGrid1.CurrentPageIndex.ToString() + "] du composant
[DataGrid]"
 End Sub

 Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles DataGrid1.PageIndexChanged
  ' chgt de page
  With DataGrid1
    .CurrentPageIndex = e.NewPageIndex
    .DataSource = textes
    .DataBind()
  End With
 End Sub
End Class
```

Commentaires :

- la source de données [textes] est un simple tableau de chaînes de caractères. Elle sera liée aux trois composants présents sur la page
- cette liaison est faite dans la procédure [Page_Load] lors de la première requête. Pour les requêtes suivantes, les trois composants retrouveront leurs valeurs par le mécanisme du [VIEW_STATE].
- les trois composants ont un gestionnaire pour l'événement [ItemCommand]. Celui-ci se produit lorsqu'un bouton ou un lien est cliqué dans l'une des lignes du composant. Le gestionnaire reçoit deux informations :
  - **source** : la référence de l'objet (bouton ou lien) à l'origine de l'événement
  - **a** : information sur l'événement de type [DataListCommandEventArgs], [RepeaterCommandEventArgs], [DataGridCommandEventArgs] selon les cas. L'argument a apporte diverses informations avec lui. Ici, deux d'entre-elles nous intéressent :
    - **a.Item** : représente la ligne sur laquelle s'est produit l'événement, de type [DataListItem], [DataGridItem] ou [RepeaterItem]. Quelque soit le type exact, l'élément [Item] a un attribut [**ItemIndex**] indiquant le numéro de la ligne [Item] du le conteneur à laquelle il appartient. Nous affichons ici ce numéro de ligne
    - **a.CommandName** : est l'attribut [CommandName] du bouton (Button, LinkButton, ImageButton) à l'origine de l'événement. Cette information conjuguée à la précédente nous permet de savoir quel bouton du conteneur est à l'origine de l'événement [ItemCommand]

3.3 Application - gestion d'une liste d'abonnements

Maintenant que nous savons comment intercepter les événements qui se produisent à l'intérieur d'un conteneur de données, nous présentons un exemple qui montre comment on peut les gérer.

3.3.1 Introduction

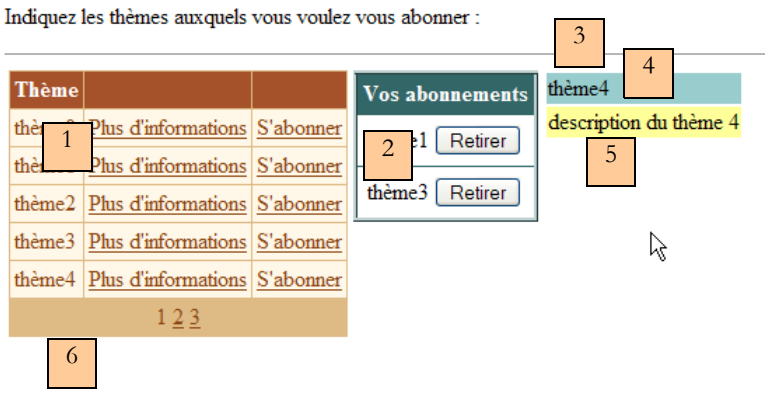
L'application présentée simule une application d'abonnements à des listes de diffusion. Celles-ci sont définies par un objet [DataTable] à trois colonnes :

nom	type	rôle
id	string	clé primaire
thème	string	nom du thème de la liste
description	string	une description des sujets traités par la liste

Pour ne pas alourdir notre exemple, l'objet [DataTable] ci-dessus sera construit par code de façon arbitraire. Dans une application réelle, il serait probablement fourni par une méthode d'une classe d'accès aux données. La construction de la table des listes se fera dans la procédure [Application_Start] et la table résultante sera placée dans l'application. Nous l'appellerons la table [dtThèmes]. L'utilisateur va s'abonner à certains des thèmes de cette table. La liste de ses abonnements sera conservée là encore dans un objet [DataTable] appelé [dtAbonnements] et dont la structure sera la suivante :

nom	type	rôle
id	string	clé primaire
thème	string	nom du thème de la liste

L'application à page unique est la suivante :

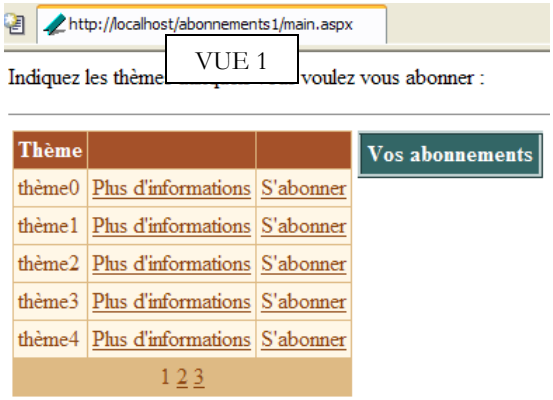


n°	nom	type	propriétés	rôle
1	dgThèmes	DataGrid		listes de diffusion proposées à l'abonnement
2	dlAbonnements	DataList		liste des abonnements de l'utilisateur aux listes précédentes
3	panelInfos	panel		panneau d'informations sur le thème sélectionné par l'utilisateur avec un lien [Plus d'informations]
4	lblThème	Label	fait partie de [panelInfos]	nom du thème
5	lblDescription	Label	fait partie de [panelInfos]	description du thème
6	lblInfo	Label		message d'information de l'application

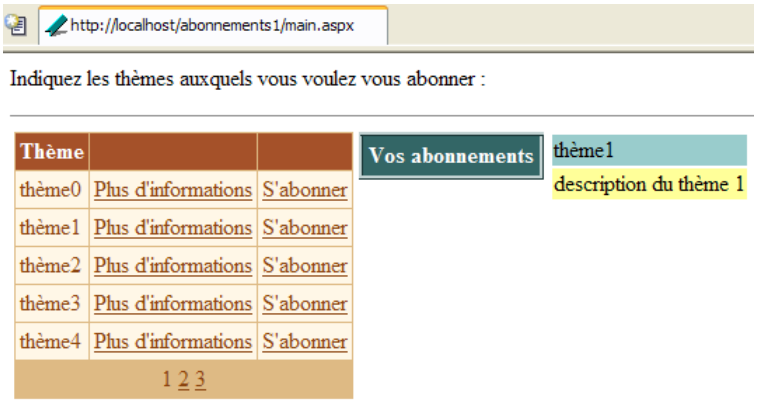
Notre exemple vise à illustrer l'utilisation des composants [DataGrid] et [DataList] notamment la gestion des événements qui se produisent au niveau des lignes de ces conteneurs de données. Aussi l'application n'a-t-elle pas de bouton de validation qui mémoriserait dans une base de données, par exemple, les choix faits par l'utilisateur. Néanmoins, il est réaliste. Nous sommes proches d'une application de commerce électronique où un utilisateur mettrait des produits (abonnements) dans son panier.

3.3.2 Fonctionnement

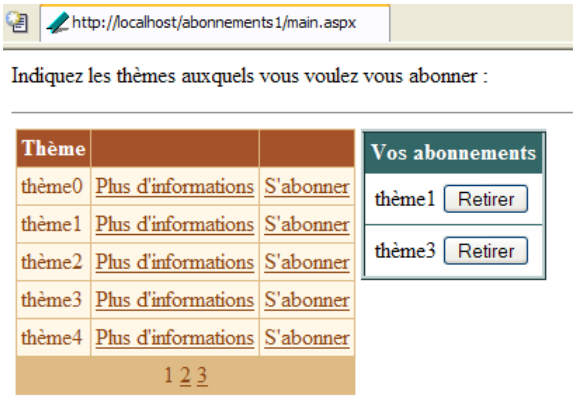
La première vue qu'a l'utilisateur est la suivante :



L'utilisateur clique sur les liens [Plus d'informations] pour avoir des renseignements sur un thème. Ceux-ci sont affichés dans [panelInfos] :

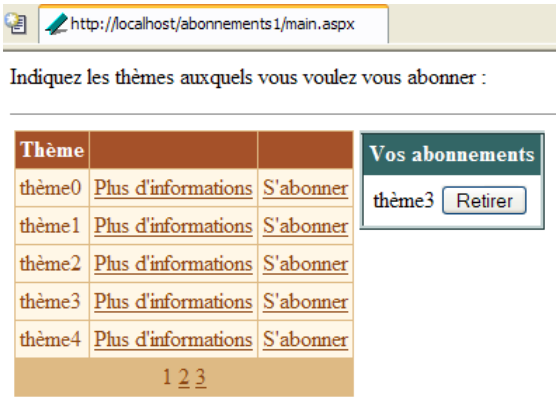


Il clique sur les liens [S'abonner] pour s'abonner à un thème. Les thèmes choisis viennent s'inscrire dans le composant [dlAbonnements] :



L'utilisateur peut vouloir s'abonner à une liste à laquelle il est déjà abonné. Un message le lui signale :

Enfin, il peut retirer son abonnement à l'un des thèmes avec les boutons [Retirer] ci-dessus. Aucune confirmation n'est demandée. Ce n'est pas utile ici car l'utilisateur peut se réabonner aisément. Voici la vue après suppression de l'abonnement à [thème1] :



3.3.3 Configuration des conteneurs de données

Le composant [dgThèmes] de type [DataGrid] est lié à une source de type [DataTable]. Sa mise en forme a été faite avec le lien [Mise en forme automatique] du panneau des propriétés du [DataGrid]. La définition de ses propriétés a elle été faite avec le lien [Générateur de propriétés] du même panneau. Le code généré est le suivant (le code de mise en forme est omis) :

```
<asp:datagrid id="dgThèmes" AutoGenerateColumns="False" AllowPaging="True" PageSize="5"
  runat="server">
  <ItemStyle ...></ItemStyle>
  <HeaderStyle ...></HeaderStyle>
  <FooterStyle ...></FooterStyle>
  <Columns>
    <asp:BoundColumn DataField="th&#232;me" HeaderText="Th&#232;me"></asp:BoundColumn>
    <asp:ButtonColumn Text="Plus d'informations" CommandName="infos"></asp:ButtonColumn>
    <asp:ButtonColumn Text="S'abonner" CommandName="abonner"></asp:ButtonColumn>
  </Columns>
  <PagerStyle HorizontalAlign="Center" ... Mode="NumericPages"></PagerStyle>
</asp:datagrid>
```

On notera les points suivants :

`AutoGenerateColumns=false`	nous définissons nous-mêmes les colonnes à afficher dans la section <columns>...</columns>
`AllowPaging=true PageSize=5`	pour la pagination des données
`<asp:BoundColumn>`	définit la colonne [thème] (HeaderText) du [DataGrid] qui sera liée à la colonne [thème] de la source de données (DataField)
`<asp:ButtonColumn>`	définit deux colonnes de boutons (ou liens). Pour différentier les deux liens d'une même ligne, on utilisera leur propriété [CommandName].

Le composant [DataGrid] n'est pas totalement paramétré. Il le sera dans le code du contrôleur.

Le composant [dlAbonnements] de type [DataList] est lié à une source de type [DataTable]. Sa mise en forme a été faite avec le lien [Mise en forme automatique] du panneau des propriétés du [DataList]. La définition de ses propriétés a elle été faite directement dans le code de présentation. Ce code est le suivant (le code de mise en forme est omis) :

```
          <asp:DataList id="dlAbonnements" ... runat="server" >
            <HeaderTemplate>
              <div align="center">
                Vos abonnements</div>
            </HeaderTemplate>
            <ItemStyle ...></ItemStyle>
            <ItemTemplate>
              <TABLE>
                <TR>
                  <TD><%#Container.DataItem("thème")%></TD>
                  <TD>
                    <asp:Button id="lnkRetirer" CommandName="retirer" runat="server" Text="Retirer"
/></TD>
                </TR>
              </TABLE>
            </ItemTemplate>
            <HeaderStyle ...></HeaderStyle>
          </asp:DataList>
```

`<HeaderTemplate>`	définit le texte de l'entête du [DataList]
`<ItemTemplate>`	définit l'élément courant du [DataList] - ici on a placé un tableau à deux colonnes et une ligne. La première cellule servira à contenir le nom du thème auquel l'utilisateur veut s'abonner, l'autre le bouton [Retirer] qui lui permet d'annuler son choix.

3.3.4 La page de présentation

Le code de présentation [main.aspx] est le suivant :

```
<%@ page src="main.aspx.vb" inherits="main" autoeventwireup="false" %>
<HTML>
 <HEAD>
   <title></title>
 </HEAD>
 <body>
   <P>Indiquez les thèmes auxquels vous voulez vous abonner :</P>
   <HR width="100%" SIZE="1">
   <form runat="server">
     <table>
       <tr>
         <td vAlign="top">
           <asp:datagrid id="dgThèmes" ... runat="server">
...
           </asp:datagrid>
         </td>
         <td vAlign="top">
           <asp:DataList id="dlAbonnements" ... runat="server" GridLines="Horizontal" ShowFooter="False">
....
           </asp:DataList>
           </td>
         <td vAlign="top">
           <asp:Panel ID="panelInfo" Runat="server" EnableViewState="False">
             <TABLE>
               <TR>
                 <TD bgColor="#99cccc">
                   <asp:Label id="lblThème" runat="server"></asp:Label></TD>
               </TR>
               <TR>
                 <TD bgColor="#ffff99">
                   <asp:Label id="lblDescription" runat="server"></asp:Label></TD>
               </TR>
             </TABLE>
           </asp:Panel>
         </td>
       </tr>
     </table>
     <P>
       <asp:Label id="lblInfo" runat="server" EnableViewState="False"></asp:Label></P>
   </form>
 </body>
</HTML>
```

3.3.5 Les contrôleurs

Le contrôle se répartit entre les fichiers [global.asax] et [main.aspx]. Le fichier [global.asax] est le suivant :

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

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

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

Public Class global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on initialise la source de données
   Dim thèmes As New DataTable
   ' colonnes
   With thèmes.Columns
     .Add("id", GetType(System.Int32))
     .Add("thème", GetType(System.String))
     .Add("description", GetType(System.String))
   End With
   ' colonne id sera clé primaire
   thèmes.Constraints.Add("cléprimaire", thèmes.Columns("id"), True)
   ' lignes
   Dim ligne As DataRow
   For i As Integer = 0 To 10
     ligne = thèmes.NewRow
     ligne.Item("id") = i.ToString
     ligne.Item("thème") = "thème" + i.ToString
     ligne.Item("description") = "description du thème " + i.ToString
     thèmes.Rows.Add(ligne)
   Next
   ' on met la source de données dans l'application
   Application("thèmes") = thèmes
 End Sub

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' début de session - on crée une table d'abonnements vide
   Dim dtAbonnements As New DataTable
   With dtAbonnements
     ' les colonnes
     .Columns.Add("id", GetType(String))
     .Columns.Add("thème", GetType(String))
     ' la clé primaire
     .PrimaryKey = New DataColumn() {.Columns("id")}
   End With
   ' la table est mise dans la session
   Session.Item("abonnements") = dtAbonnements
 End Sub

End Class
```

La procédure [Application_Start], exécutée lorsque l'application reçoit sa toute première requête, construit le [DataTable] des thèmes auxquels on peut s'abonner. Il est construit arbitrairement avec du code. Rappelons la technique que nous avons déjà rencontrée. On construit dans l'ordre :

- un objet [DataTable] vide, sans structure et sans données
- la structure de la table en définissant ses colonnes (nom et type de données qu'elle contient)
- les lignes de la table qui représentent les données utiles

Nous avons ici ajouté une clé primaire. C'est la colonne "id" qui sert de clé primaire. Il y a plusieurs façons de le dire. Ici, nous avons utilisé une contrainte. En SQL, une contrainte est une règle que doivent observer les données d'une ligne pour que celle-ci puisse être ajoutée à une table. Il existe toutes sortes de contraintes possibles. La contrainte "Primary Key" force la colonne à laquelle elle est imposée à avoir des valeurs uniques et non vides. Une clé primaire peut être en fait constituée d'une expression faisant intervenir des valeurs de plusieurs colonnes. [DataTable].Constraints est la collection des contraintes d'une table donnée. Pour ajouter une contrainte, on utilise la méthode [DataTable.Constraints.Add]. Celle-ci a plusieurs signatures. Ici, on a utilisé la méthode [Add(Byval nom as String, Byval colonne as DataColumn, Byval cléPrimaire as Boolean)] :

`nom`	nom de la contrainte - peut être quelconque
`colonne`	colonne qui sera clé primaire - de type [DataColumn]
`cléPrimaire`	doit être à [vrai] pour faire de [colonne] une clé primaire. Si [cléPrimaire=false], on a seulement la contrainte de valeurs uniques sur [colonne]

Pour faire de la colonne nommée "id" la clé primaire de la table [dtAbonnements], on écrit donc :

```
dtAbonnements.Constraints.Add("xxx",dtAbonnements.Columns("id"),true)
```

La procédure [Session_Start], exécutée lorsque l'application reçoit la première requête d'un client. Elle sert à créer des objets propres à chaque client qui doivent persister au travers des différentes requêtes de celui-ci. La procédure construit le [DataTable] des abonnements du client. Seule sa structure est construire puisqu'au départ cette table est vide. Elle va se remplir au fil des requêtes. Là également, la colonne "id" sert de clé primaire. On a utilisé une technique différente pour déclarer cette contrainte :

`[DataTable].PrimaryKey`	est le tableau des colonnes formant la clé primaire - ici on a déclaré un tableau à un élément : la colonne de nom "id"

Lorsque la requête du client atteint le contrôleur [main.aspx], les deux objets [DataTable] sont disponibles dans l'application pour la table des thèmes et dans la session pour la table des abonnements. Le contrôleur [main.aspx.vb] est le suivant :

```
Imports System.Data

Public Class main
 Inherits System.Web.UI.Page

 Protected WithEvents dgThèmes As System.Web.UI.WebControls.DataGrid
 Protected WithEvents lblThème As System.Web.UI.WebControls.Label
 Protected WithEvents lblDescription As System.Web.UI.WebControls.Label
 Protected WithEvents dlAbonnements As System.Web.UI.WebControls.DataList
 Protected WithEvents lblInfo As System.Web.UI.WebControls.Label
 Protected WithEvents panelInfo As System.Web.UI.WebControls.Panel

 Protected dtThèmes As DataTable
 Protected dtAbonnements As DataTable

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

 Private Sub liaisons()
...
 End Sub

 Private Sub dgThèmes_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles dgThèmes.PageIndexChanged
...
 End Sub

 Private Sub dgThèmes_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles dgThèmes.ItemCommand
...
 End Sub

 Private Sub infos(ByVal id As String)
...
 End Sub

 Private Sub abonner(ByVal id As String)
...
 End Sub

 Private Sub dlAbonnements_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataListCommandEventArgs) Handles dlAbonnements.ItemCommand
...
 End Sub
End Class
```

La procédure [Page_Load] a pour rôle essentiel de :

- récupérer les deux tables [dtThèmes] et [dtAbonnements] qui sont respectivement dans l'application et la session afin de les rendre disponibles à toutes les méthodes de la page

- faire la liaison de données de ces deux sources avec leurs conteneurs respectifs. Ceci est fait seulement lors de la première requête. Lors des requêtes suivantes, la liaison n'est pas à faire systématiquement et lorsqu'elle est à faire, il faut parfois attendre un événement postérieur à [Page_Load] pour la faire.

Le code de [Page_Load] est le suivant :

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
  ' on récupère les sources de données
  dtThèmes = CType(Application("thèmes"), DataTable)
  dtAbonnements = CType(Session("abonnements"), DataTable)
  ' liaison de données
  If Not IsPostBack Then
    liaisons()
  End If
  ' on cache certaines informations
  panelInfo.Visible = False
End Sub

Private Sub liaisons()
  'on lie la source de données au composant [datagrid]
  With dgThèmes
    .DataSource = dtThèmes
    .DataKeyField = "id"
  End With
  ' on lie la source de données au composant [datalist]
  With dlAbonnements
    .DataSource = dtAbonnements
    .DataKeyField = "id"
  End With
  ' on assigne les données aux composants
  Page.DataBind()
End Sub
```

Dans la procédure [liaisons], nous utilisons la propriété [DataKeyField] des composants [DataList] et [DataGrid] pour définir la colonne de la source de données qui servira à identifier de façon unique les lignes des conteneurs. Classiquement, cette colonne est la clé primaire de la source de données mais ce n'est pas obligatoire. Il suffirait que la colonne soit exempte de doublons et de valeurs vides. Pour le conteneur [dgThèmes], c'est la colonne "id" de la source [dtThèmes] qui servira de clé primaire et pour le conteneur [dlAbonnements] ce sera la colonne "id" de la source [dtAbonnements]. Il n'y a pas nécessité à ce que la colonne qui sert de clé primaire au conteneur soit affichée par celui-ci. Ici, aucun des deux conteneurs n'affiche la colonne clé primaire. L'intérêt qu'un conteneur ait une clé primaire est que celle-ci permet de retrouver aisément dans la source de données, les informations liées à la ligne du conteneur sur laquelle s'est produit un événement. En effet, il est fréquent qu'à partir de la ligne d'un conteneur sur laquelle s'est produit un événement, on doive agir sur la ligne correspondante de la source de données qui lui est liée. La clé primaire facilite ce travail.

La pagination du [DataGrid] est gérée de façon classique :

```
Private Sub dgThèmes_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles dgThèmes.PageIndexChanged
  ' chgt de page
  dgThèmes.CurrentPageIndex = e.NewPageIndex
  ' liaison
  liaisons()
End Sub
```

Les actions sur les liens [Plus d'informations] et [S'abonner] sont gérées par la procédure [dgThèmes_ItemCommand] :

```
Private Sub dgThèmes_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles dgThèmes.ItemCommand
  ' évt sur une ligne du [datagrid]
  Dim commande As String = e.CommandName
  Select Case commande
    Case "infos"
      infos(dgThèmes.DataKeys(e.Item.ItemIndex))
    Case "abonner"
      abonner(dgThèmes.DataKeys(e.Item.ItemIndex))
  End Select
  ' liaison
  liaisons()
End Sub
```

On utilise le fait que les deux liens ont un attribut [CommandName] pour les différentier. Selon la valeur de cet attribut, on appelle la procédure [infos] ou [abonner] en passant dans les deux cas la clé "id" associée à l'élément du [DataGrid] où s'est produit l'événement. Munie de cette information, la procédure [info] va afficher les informations du thème choisi par l'utilisateur :

```
Private Sub infos(ByVal id As String)
  ' informations sur le thème de clé id
  ' on récupère la ligne du [datatable] correspondant à la clé
  Dim ligne As DataRow
  ligne = dtThèmes.Rows.Find(id)
  If Not ligne Is Nothing Then
    ' on affiche l'info
    lblThème.Text = CType(ligne("thème"), String)
    lblDescription.Text = CType(ligne("description"), String)
    panelInfo.Visible = True
  End If
End Sub
```

La table [dtThèmes] ayant une clé primaire, la méthode [dtThèmes.Rows.Find("P")] permet de trouver la ligne ayant la clé primaire P. Si elle est trouvée, on obtient un objet [DataRow]. Ici, nous devons chercher la ligne ayant la clé primaire [id] où [id] est passé en paramètre. Si la ligne est trouvée, on met les informations [thème] et [description] de cette ligne dans le panneau d'informations qu'on rend ensuite visible.

La procédure [abonner(id)] doit ajouter le thème de clé [id] à la liste des abonnements. Son code est le suivant :

```
Private Sub abonner(ByVal id As String)
  ' abonnement au thème de clé id
  ' on récupère la ligne du [datatable] correspondant à la clé
  Dim ligne As DataRow
  ligne = dtThèmes.Rows.Find(id)
  If Not ligne Is Nothing Then
    ' on vérifie si on n'est pas déjà abonné
    Dim abonnement As DataRow
    abonnement = dtAbonnements.Rows.Find(id)
    If Not abonnement Is Nothing Then
      ' on signale l'erreur
      lblInfo.Text = "Vous êtes déjà abonné au thème [" + ligne("thème") + "]"
    Else
      ' on ajoute le thème aux abonnements
      abonnement = dtAbonnements.NewRow
      abonnement("id") = id
      abonnement("thème") = ligne("thème")
      dtAbonnements.Rows.Add(abonnement)
      ' on fait les liaisons
      liaisons()
    End If
  End If
End Sub
```

Dans la liste des thèmes [dtThèmes], on cherche tout d'abord la ligne de clé [id]. Si on la trouve, on vérifie que ce thème n'est pas déjà présent dans la liste des abonnements pour éviter de l'inscrire deux fois. Si c'est le cas, on affiche un mesage d'erreur. Sinon, on ajoute un nouvel abonnement à la table [dtAbonnements] et on fait les liaisons des composants de liste de données à leurs sources respectives.

Lorsque l'utilisateur clique sur un bouton [Retirer], on doit supprimer un élément de la table [dtAbonnements]. Cela est fait par la procédure suivante :

```
Private Sub dlAbonnements_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataListCommandEventArgs) Handles dlAbonnements.ItemCommand
  ' on retire un abonnement
  Dim commande As String = e.CommandName
  If commande = "retirer" Then
    ' on retire l'abonnement du [datatable]
    With dtAbonnements.Rows
      .Remove(.Find(dlAbonnements.DataKeys(e.Item.ItemIndex)))
    End With
    ' liaisons
    liaisons()
  End If
End Sub
```

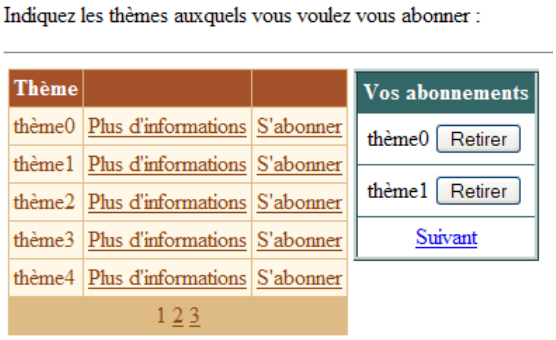
On vérifie tout d'abord l'attribut [CommandName] de l'élément à l'origne de l'événement. C'est en fait assez inutile puisque le bouton [Retirer] est le seul contrôle capable de générer un événement dans le composant [DataList]. Il n'y a donc pas d'ambiguïté. Pour supprimer une ligne d'un objet [DataTable], on utilise la méthode [DataList.Remove(DataRow)] qui enlève de la table, la ligne de type [DataRow] passée en paramètre. Celle-ci est trouvée par la méthode [DataList.Find] à qui on a passé la clé primaire de la ligne cherchée. Une fois la ligne détruite, on fait la liaison des données aux composants

3.4 Gérer un [DataList] paginé

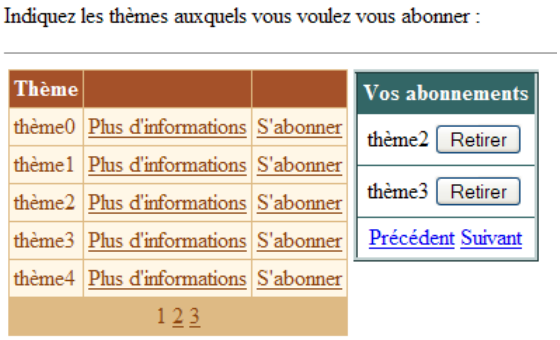
Nous reprenons l'exemple précédent afin de paginer le composant [DataList] représentant la liste des abonnements de l'utilisateur. Contrairement au composant [DataGrid], le composant [DataList] n'offre aucune facilité pour la pagination. Nous allons voir que la mise en place de celle-ci est complexe, ce qui nous fera apprécier à sa juste valeur la pagination automatique du [DataGrid].

3.4.1 Fonctionnement

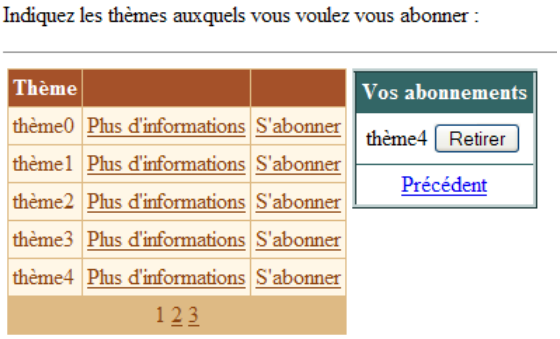
La seule différence est la pagination du [DataList]. Les pages auront deux abonnements. Si l'utilisateur a cinq abonnements, on aura trois pages. La première sera la suivante :



La deuxième page est obtenue via le lien [Suivant] :



La troisième page :



On remarquera que les liens [Précédent] et [Suivant] ne sont visibles que s'il y a respectivement une page qui précède et une page qui suit la page courante.

3.4.2 Code de présentation

Les liens [Précédent] et [Suivant] sont obtenus en ajoutant une balise <FooterTemplate> au [DataList] :

```
          <asp:datalist id="dlAbonnements" runat="server" ...>
....
            <FooterTemplate>
              <asp:LinkButton id="lnkPrecedent" runat="server"
CommandName="precedent">Précédent</asp:LinkButton>
```

```
            <asp:LinkButton id="lnkSuivant" runat="server"
CommandName="suivant">Suivant</asp:LinkButton>
          </FooterTemplate>
....
        </asp:datalist>
```

3.4.3 Code de contrôle

Le fichier associé [global.asax.vb] évolue de la façon suivante :

```
...

Public Class global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
...
 End Sub

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' début de session - on crée une table d'abonnements vide
   Dim dtAbonnements As New DataTable
   With dtAbonnements
     ' les colonnes
     .Columns.Add("id", GetType(String))
     .Columns.Add("thème", GetType(String))
     ' la clé primaire
     .PrimaryKey = New DataColumn() {.Columns("id")}
   End With
   ' la table est mise dans la session
   Session.Item("abonnements") = dtAbonnements
   ' la page courante est la page 0
   Session.Item("pAC") = 0
   ' le nombre d'abonnements dans cette page est 0
   Session.Item("nbAC") = 0
 End Sub
```

Outre la table des abonnements [dtAbonnements], on met dans la session deux autres informations :

pAC	de type [Integer] - c'est le n° de la page courante affichée lors de la dernière requête
nbAC	de type [Integer] - nbre de lignes affichées dans la page courante précédente

Au début d'une session, le n° de page courante ainsi que le nombre de lignes de cette page sont nuls.

Le contrôleur [main.aspx.vb] évolue de la façon suivante :

```
....

Public Class main
 Inherits System.Web.UI.Page

....

 ' données de l'application
 Protected dtThèmes As DataTable
 Protected dtAbonnements As DataTable
 Protected dtPA As DataTable  ' la page d'abonnements affichée
 Protected Const nbAP As Integer = 2 ' nbre abonnements par page
 Protected pAC As Integer ' page abonnement courant
 Protected nbAC As Integer  ' nombre d'abonnements dans page courante

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

 Private Sub terminer()
...
 End Sub

 Private Sub dgThèmes_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles dgThèmes.PageIndexChanged
...
 End Sub
```

```
 Private Sub dgThèmes_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles dgThèmes.ItemCommand
...
 End Sub

 Private Sub infos(ByVal id As String)
...
 End Sub

 Private Sub abonner(ByVal id As String)
...
 End Sub

 Private Sub dlAbonnements_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataListCommandEventArgs) Handles dlAbonnements.ItemCommand
...
 End Sub

 Private Sub changePAC()
...
 End Sub

 Private Sub setLiens(ByVal ctl As Control, ByVal blPrec As Boolean, ByVal blSuivant As Boolean)
...
 End Sub
End Class
```

On y définit de nouvelles données liées à la pagination des abonnements :

```
 Protected dtPA As DataTable  ' la page d'abonnements affichée
 Protected Const nbAP As Integer = 2 ' nbre abonnements par page
 Protected pAC As Integer ' page abonnement courant
 Protected nbAC As Integer  ' nombre d'abonnements dans page courante
```

Certaines de ces informations sont stockées dans la session et sont récupérées à chaque requête dans la procédure [Page_Load] :

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' on récupère les sources de données
   dtThèmes = CType(Application("thèmes"), DataTable)
   dtAbonnements = CType(Session("abonnements"), DataTable)
   ' et les informations d'affichage des abonnements
   pAC = CType(Session("pAC"), Integer)
   nbAC = CType(Session("nbAC"), Integer)
   ' liaison de données
   If Not IsPostBack Then
     ' affichage d'une liste d'abonnements vide
     terminer()
   End If
   ' on cache certaines informations
   panelInfo.Visible = False
 End Sub
```

Les informations récupérées [pAC] et [nbAC] sont des informations sur la page d'abonnements affichée lors de la précédente requête :

pAC	c'est le n° de la page courante affichée lors de la précédente requête
nbAC	nombre de lignes affichées dans cette page courante

La méthode [terminer] fait les liaisons des composants avec leurs sources de données comme le faisait la méthode [liaisons] dans l'application précédente. La nouveauté, ici, est la liaison du [DataList] avec la table [dtPA] qui est la page d'abonnements à afficher :

```
 Private Sub terminer()
   ' on lie la source de données au composant [datagrid]
   With dgThèmes
     .DataSource = dtThèmes
     .DataKeyField = "id"
   End With
   ' on lie la page d'abonnements au composant [datalist] en tenant compte de la page courante pAC
   changePAC()
   ' on visualise la page p
   With dlAbonnements
     .DataSource = dtPA
     .DataKeyField = "id"
   End With
   ' on assigne les données aux composants
```

```
    Page.DataBind()
    ' gestion des liens [précédent] et [suivant] du [datalist]
    Dim blprec As Boolean = pAC <> 0
    Dim blsuivant As Boolean = pAC <> (dtAbonnements.Rows.Count - 1) \ nbAP
    Dim nbLiensTrouvés As Integer = 0
    setLiens(dlAbonnements, blprec, blsuivant, nbLiensTrouvés)
    ' on sauvegarde les informations de page courante dans la session
    Session("pAC") = pAC
    Session("nbAC") = dtPA.Rows.Count
  End Sub
```

Les points suivants sont à noter :

- la source [dtPA] dépend du n° de page courante [pAC] à afficher. La variable [pAC] est une variable globale de la classe, manipulée par les méthodes qui ont à faire évoluer ce n° de page courante. C'est la méthode [changePAC] qui fait le travail de construction de la table [dtPA] qui sera liée au composant [dlAbonnements].
- la méthode [setLiens] a pour rôle d'afficher ou de cacher les liens [Précédent] et [Suivant] selon que la page courante [pAC] affichée est précédée et suivie d'une page. Elle a quatre paramètres :
  - [dlAbonnements] : le contrôle [DataList] dont on va explorer l'arborescence des contrôles pour trouver les deux liens. En effet, bien que ces deux liens soient dans un endroit précis qui est le pied-de-page du [DataList], il ne semble pas qu'il y ait un moyen simple de les référencer directement. En tout cas, il n'a pas été trouvé ici.
  - [blPrecedent] : booléen à affecter à la propriété [visible] du lien [Precedent] - est à vrai si la page courante est différente de 0
  - [blSuivant] : booléen à affecter à la propriété [visible] du lien [Suivant] - est à vrai si la page courante n'est pas la dernière page de la liste des abonnements
  - [nbLiensTrouvés] : un paramètre de sortie qui compte le nombre de liens trouvés. Dès que cette quantité est égale à deux, la méthode est terminée.
- les informations [pAC] et [nbAC] sont sauvegardées dans la session pour la prochaine requête.

La méthode [changePAC] construit la table [dtPA] qui sera liée au composant [dlAbonnements]. Elle le fait d'après le n° [pAC] de la page courante à afficher. La table [dtPA] doit afficher certaines lignes de la table des abonnements [dtAbonnements]. Rappelons que cette table est mémorisée dans la session et mise à jour (augmentée ou diminuée) au fil des requêtes. On commence par fixer l'intervalle [premier,dernier] des n°s de lignes de la table [dtAbonnements] que la table [dtPA] doit afficher :

```
Private Sub changePAC()
  ' fait de la page pAC, la page courante des abonnements
  ' gestion des pages du [datalist]
  Dim nbAbonnements = dtAbonnements.Rows.Count
  Dim dernièrePage = (nbAbonnements - 1) \ nbAP
  ' premier et dernier abonnement
  If pAC < 0 Then pAC = 0
  If pAC > dernièrePage Then pAC = dernièrePage
  Dim premier As Integer = pAC * nbAP
  Dim dernier As Integer = (pAC + 1) * nbAP - 1
  If dernier > nbAbonnements - 1 Then dernier = nbAbonnements - 1
```

Ceci fait, on peut construire la table [dtPA]. On définit tout d'abord sa structure [id,thème] puis on la remplit en y recopiant les lignes de [dtAbonnements] dont le n° est dans l'intervalle [premier,dernier] calculé précédemment.

```
  ' création du datatable dtpa
  dtPA = New DataTable
  With dtPA
    ' les colonnes
    .Columns.Add("id", GetType(String))
    .Columns.Add("thème", GetType(String))
    ' la clé primaire
    .PrimaryKey = New DataColumn() {.Columns("id")}
  End With
  Dim abonnement As DataRow
  For i As Integer = premier To dernier
    abonnement = dtPA.NewRow
    With abonnement
      .Item("id") = dtAbonnements.Rows(i).Item("id")
      .Item("thème") = dtAbonnements.Rows(i).Item("thème")
    End With
    dtPA.Rows.Add(abonnement)
  Next
End Sub
```

A la fin de la méthode [changePAC], la table [dtPA] a été construite et peut être liée au composant [DataList]. C'est fait dans la méthode [terminer]. Dans cette même méthode, on utilise la procédure [setLiens] pour fixer l'état des liens [Précédent] et [Suivant] du [DataList]. Le code de cette procédure est le suivant :

```
 Private Sub setLiens(ByVal ctl As Control, ByVal blPrec As Boolean, ByVal blSuivant As Boolean, ByRef
nbLiensTrouvés As Integer)
   ' on recherche les liens [précédent] et [suivant]
   ' dans l'arborescence des contrôles du [datalist]
   ' a-t-on trouvé tous les liens ?
   If nbLiensTrouvés = 2 Then Exit Sub
   ' examen des contrôles enfant
   Dim c As Control
   For Each c In ctl.Controls
     ' on travaille d'abord en profondeur - les liens sont au fond de l'arbre
     setLiens(c, blPrec, blSuivant, nbLiensTrouvés)
     ' lien [Précédent] ?
     If c.ID = "lnkPrecedent" Then
       CType(c, LinkButton).Visible = blPrec
       nbLiensTrouvés += 1
     End If
     ' lien [Suivant] ?
     If c.ID = "lnkSuivant" Then
       CType(c, LinkButton).Visible = blSuivant
       nbLiensTrouvés += 1
     End If
   Next
 End Sub
```

La procédure est récursive. Elle cherche tout d'abord, parmi les contrôles enfants du composant [dlAbonnements] des composants s'appelant [lnkPrecedent] et [lnkSuivant] qui sont les identités (attribut ID) des deux liens de pagination. Elle les cherche d'abord en partant du fond de l'arbre des contrôles car c'est là qu'ils sont. Dès qu'un lien a été trouvé, le compteur [nbLiensTrouvés] est incrémenté et la propriété [visible] du lien est renseignée avec une valeur passée en paramètre de la procédure. Dès que les deux liens ont été trouvés, l'arbre des contrôles n'est plus exploré et la procédure récursive se termine.

Nous avons dit que la méthode [changePAC] qui fixe la source de données [dtPA] pour le composant [dlAbonnements] travaillait avec le numéro [pAC] de la page courante à afficher. Plusieurs procédures modifient ce numéro :

```
 Private Sub abonner(ByVal id As String)
   ' abonnement au thème de clé id
..
     ' on ajoute le thème aux abonnements
..
     ' mise à jour du n° de page courante - c'est maintenant la dernière page
     pAC = (dtAbonnements.Rows.Count - 1) \ nbAP
     ' liaisons de données
     terminer()
   End If
 End Sub
```

Après l'ajout d'un abonnement, celui-ci se retrouve en fin de liste des abonnements. Aussi se positionne-t-on sur la dernière page des abonnements afin que l'utilisateur voit l'ajout qui a été opéré.

```
 Private Sub dlAbonnements_ItemCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataListCommandEventArgs) Handles dlAbonnements.ItemCommand
   ' on retire un abonnement
   Dim commande As String = e.CommandName
   Select Case commande
     Case "retirer"
       ' on retire l'abonnement du [datatable]
       With dtAbonnements.Rows
         .Remove(.Find(dlAbonnements.DataKeys(e.Item.ItemIndex)))
       End With
       ' faut-il changer de page courante ?
       nbAC -= 1
       If nbAC = 0 Then pAC -= 1
     Case "precedent"
       ' chgt de page courante
       pAC -= 1
     Case "suivant"
       ' chgt de page courante
       pAC += 1
   End Select
   ' liaisons de données
   terminer()
 End Sub
```

- [nbAC] est le nombre de lignes affichées dans la page courante avant le retrait d'un abonnement. Si le nouveau nombre de lignes de la page est égal à 0, le n° de page courante [pAC] est décrémenté d'une unité.
- en cas de clic sur le lien [Precedent], le le n° de page courante [pAC] est décrémenté d'une unité.
- en cas de clic sur le lien [Suivant], le n° de page courante [pAC] est incrémenté d'une unité.

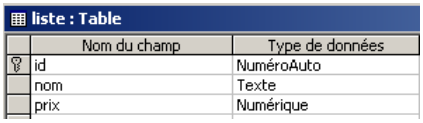
Les autres procédures restent identiques à ce qu'elles étaient précédemment.

3.4.4 Conclusion

Nous avons montré sur cet exemple que nous pouvions paginer un composant [DataList]. Cette pagination est délicate et il est préférable de s'en remettre à la pagination automatique du composant [DataGrid] quand cela est possible. Cet exemple nous a également montré comment atteindre les composants présents dans le pied-de-page du composant [DataList].

3.5 Classe d'accès à une base de produits

Nous nous intéressons de nouveau à la base de données ACCESS [produits] déjà utilisée. Rappelons qu'elle a une unique table appelée [liste] dont la structure est la suivante :



id	nom	prix
1	produit1	10
2	produit2	20
3	produit3	30

Nous allons construire une classe d'accès à la table [liste] qui permettra de la lire et de la mettre à jour. Nous construirons également un client console qui se servira de la classe précédente pour mettre à jour la table. Dans un deuxième temps, nous construirons un client web pour faire ce même travail.

3.5.1 La classe ExceptionProduits

La classe [Exception] a un constructeur qui admet un message d'erreur comme paramètre. Nous souhaitons ici disposer d'une classe d'exception disposant d'un constructeur admettant une liste de messages d'erreurs plutôt qu'un message d'erreur unique. Ce sera la classe [ExceptionProduits] ci-dessous :

```
Public Class ExceptionProduits
  Inherits Exception

  ' msg d'erreurs liées à l'exception
  Private _erreurs As ArrayList

  ' constructeur
  Public Sub New(ByVal erreurs As ArrayList)
    Me._erreurs = erreurs
  End Sub

  ' propriété
  Public ReadOnly Property erreurs() As ArrayList
    Get
      Return _erreurs
    End Get
  End Property
End Class
```

3.5.2 La structure [sProduit]

La structure [produit] représentera un produit [id, nom, prix] :

```
' structure sProduit
Public Structure sProduit
  ' les champs
  Private _id As Integer
  Private _nom As String
  Private _prix As Double

  ' propriété id
  Public Property id() As Integer
    Get
      Return _id
    End Get
    Set(ByVal Value As Integer)
      _id = Value
    End Set
  End Property

  ' propriété nom
  Public Property nom() As String
    Get
```

```
      Return _nom
    End Get
    Set(ByVal Value As String)
      If IsNothing(Value) OrElse Value.Trim = String.Empty Then Throw New Exception
      _nom = Value
    End Set
  End Property

  ' propriété prix
  Public Property prix() As Double
    Get
      Return _prix
    End Get
    Set(ByVal Value As Double)
      If IsNothing(Value) OrElse Value < 0 Then Throw New Exception
      _prix = Value
    End Set
  End Property
End Structure
```

La structure n'admet que des données valides pour les champs [nom] et [prix].

3.5.3 La classe Produits

La classe [Produits] est la classe qui va nous permettre de mettre à jour la table [liste] de la base de produits. Sa structure est la suivante :

```
Public Class produits
  ' données d'instance

  Public Sub New(ByVal chaineConnexionOLEDB As String)
....
  End Sub

  Public Function getProduits() As DataTable
....
  End Function

  Public Sub ajouterProduit(ByVal produit As sProduit)
...
  End Sub

  Public Sub modifierProduit(ByVal produit As sProduit)
...
  End Sub

  Public Sub supprimerProduit(ByVal id As Integer)
...
  End Sub

End Class
```

Les données d'instance

Les données partagées par les différentes méthodes de la classe sont les suivantes :

```
  Private connexion As OleDbConnection
  Private Const selectText As String = "select id,nom,prix from liste"
  Private Const insertText As String = "insert into liste(nom,prix) values(?,?)"
  Private Const updateText As String = "update liste set nom=?,prix=? where id=?"
  Private Const deleteText As String = "delete from liste where id=?"
  Private selectCommand As New OleDbCommand
  Dim insertCommand As New OleDbCommand
  Dim updateCommand As New OleDbCommand
  Dim deleteCommand As New OleDbCommand
  Dim adaptateur As New OleDbDataAdapter
```

`connexion`	la connexion à la base de données - sera ouverte pour l'exécution d'une commande SQL puis refermée aussitôt après
`selectText`	requête SQL [select] obtenant toute la table [liste]
`insertText`	requête permettant l'insertion d'une ligne (nom, prix) dans la table [liste]. On remarquera qu'on ne précise pas le champ [id]. En effet, ce champ est auto-incrémenté par le SGBD et donc nous n'avons pas à le spécifier.
`updateText`	requête permettant la mise à jour des champs (nom,prix) de la ligne de la table [liste] ayant la clé [id]

deleteText	requête permettant la suppression de la ligne de la table [liste] ayant la clé [id]
selectCommand	objet [OleDbCommand] exécutant la requête [selectText] sur la connexion [connexion]
updateCommand	objet [OleDbCommand] exécutant la requête [updateText] sur la connexion [connexion]
insertCommand	objet [OleDbCommand] exécutant la requête [insertText] sur la connexion [connexion]
deleteCommand	objet [OleDbCommand] exécutant la requête [deleteText] sur la connexion [connexion]
adaptateur	objet permettant de récupérer le résultat de l'exécution de [selectCommand] dans un objet [DataSet]

Le constructeur

Le constructeur reçoit un unique paramètre [chaineConnexionOLEDB] qui est la chaîne de connexion désignant la bse de données à exploiter. A partir de celle-ci, on prépare les quatre commandes d'exploitation et de mise à jour de la table ainsi que l'adaptateur. Ce n'est qu'une préparation et aucune connexion n'est faite.

```
Public Sub New(ByVal chaineConnexionOLEDB As String)
  ' on prépare la connexion
  connexion = New OleDbConnection(chaineConnexionOLEDB)
  ' on prépare les commandes de requêtes
  Dim commandes() As OleDbCommand = {selectCommand, insertCommand, updateCommand, deleteCommand}
  Dim textes() As String = {selectText, insertText, updateText, deleteText}
  For i As Integer = 0 To commandes.Length - 1
    With commandes(i)
      .CommandText = textes(i)
      .Connection = connexion
    End With
  Next
  ' on prépare l'adaptateur d'accès aux données
  adaptateur.SelectCommand = selectCommand
End Sub
```

La méthode getProduits

Cette méthode permet d'avoir le contenu de la table [Liste] dans un objet [DataTable]. Son code est le suivant :

```
Public Function getProduits() As DataTable
  ' on met la table [liste] dans un [dataset]
  Dim contenu As New DataSet
  ' on crée un objet DataAdapter pour lire les données de la source OLEDB
  Try
    With adaptateur
      .FillSchema(contenu, SchemaType.Source)
      .Fill(contenu)
    End With
  Catch e As Exception
    ' pb
    Dim erreursCommande As New ArrayList
    erreursCommande.Add(String.Format("Erreur d'accès à la base de données : {0}", e.Message))
    Throw New ExceptionProduits(erreursCommande)
  End Try
  ' on rend le résultat
  Return contenu.Tables(0)
End Function
```

Le travail est fait par les deux instructions suivantes :

```
    With adaptateur
      .FillSchema(contenu, SchemaType.Source)
      .Fill(contenu)
    End With
```

La méthode [FillSchema] fixe la structure (colonnes, contraintes, relations) du [DataSet] *contenu* à partir de la structure de la base référencée par [adaptateur.Connexion]. Cela nous permet de récupérer la structure de la table [liste] et notamment sa clé primaire. L'opération [Fill] qui suit remplit le [Dataset] *contenu* avec les lignes de la table [liste]. Avec cette seule opération, on aurait bien eu les données et la structure mais pas la clé primaire. Or celle-ci nous sera utile pour mettre à jour la table [liste] en mémoire. Ici, comme dans les autres méthodes, nous gérons les éventuelles erreurs à l'aide de la classe [ExceptionProduits] afin d'obtenir une liste (ArrayList) d'erreurs plutôt qu'une seule erreur. La méthode [getProduits] rend la table [liste] sous la forme d'un objet [DataTable].

La méthode ajouterProduits

Cette méthode permet d'ajouter une ligne (id, nom, prix) à la table [liste]. Ces informations lui sont données sous la forme d'une structure [sProduit] de champs [id, nom, prix]. Le code de la méthode est le suivant :

```
  Public Sub ajouterProduit(ByVal produit As sProduit)
    ' on ajoute un produit [nom,prix]
    ' on prépare les paramètres de l'ajout
    With insertCommand.Parameters
      .Clear()
      .Add(New OleDbParameter("nom", produit.nom))
      .Add(New OleDbParameter("prix", produit.prix))
    End With
    ' on fait l'ajout
    Try
      ' ouverture connexion
      connexion.Open()
      ' exécution commande
      insertCommand.ExecuteNonQuery()
    Catch ex As Exception
      ' pb
      Dim erreursCommande As New ArrayList
      erreursCommande.Add(String.Format("Erreur lors de l'ajout : {0}", ex.Message))
      Throw New ExceptionProduits(erreursCommande)
    Finally
      ' fermeture connexion
      connexion.Close()
    End Try
  End Sub
```

Les champs de la structure [produit] sont injectés dans les paramètres de la commande [insertCommand]. Rappelons la configuration actuelle de cette commande (cf constructeur) :

```
Private Const insertText As String = "insert into liste(nom,prix) values(?,?)"
insertCommand.Connexion=connexion
```

Le texte de la commande SQL [insert] comporte des paramètres formels ? qu'il faut remplacer par des paramètres effectifs. Cela se fait avec la collection [Parameters] de la classe [OleDbCommand]. Celle-ci contient des éléments de type [OleDbParameter] qui définissent les paramètres effectifs qui doivent remplacer les paramètres formels ?. Comme ces derniers ne sont pas nommés, c'est l'index des paramètres effectifs qui est utilisé pour savoir à quel paramètre formel correspond tel paramètre effectif. Ici, le paramètre effectif n° i dans la collection [Parameters] remplacera le paramètre formel ? n° i. Pour créer un paramètre effectif de type [OleDbParameter] on utilise ici le constructeur [OleDbParameter (Byval nom as String, Byval valeur as Object)] qui définit le nom et la valeur du paramètre effectif. Le nom peut être quelconque. De plus ici, il ne sera pas utilisé. Les deux paramètres de l'instruction SQL [insert] reçoivent pour valeurs celles des champs [nom, prix] de la structure [produit]. Ceci fait, l'insertion est faite par l'instruction [insertCommand.ExecuteNonQuery].

La méthode modifierProduits

Cette méthode permet de modifier la ligne de la table [liste]. Les informations qui lui sont nécessaires lui sont données sous la structure [sProduit] de champs [id, nom, prix].

```
  Public Sub modifierProduit(ByVal produit As sProduit)
    ' on modifie un produit [id,nom,prix]
    ' on prépare les paramètres de la mise à jour
    With updateCommand.Parameters
      .Clear()
      .Add(New OleDbParameter("nom", produit.nom))
      .Add(New OleDbParameter("prix", produit.prix))
      .Add(New OleDbParameter("id", produit.id))
    End With
    ' on fait la modification
    Try
      ' ouverture connexion
      connexion.Open()
      ' exécution commande
      Dim nbLignes As Integer = updateCommand.ExecuteNonQuery()
      If nbLignes = 0 Then Throw New Exception(String.Format("Le produit de clé [{0}] n'existe pas dans
la table des données", produit.id))
    Catch ex As Exception
      ' pb
      Dim erreursCommande As New ArrayList
      erreursCommande.Add(String.Format("Erreur lors de la modification : {0}", ex.Message))
      Throw New ExceptionProduits(erreursCommande)
    Finally
      ' fermeture connexion
      connexion.Close()
    End Try
```

```
  End Sub
```

Le code est quasi identique à celui de la méthode [ajouterProduits] si ce n'est que la commande [OleDbCommand] concernée est [updateCommand] au lieu de [insertCommand].

La méthode supprimerProduits

Cette méthode permet de suprimer la ligne de la table [liste] de clé [id] passée en paramètre. Le code est le suivant :

```
  Public Sub supprimerProduit(ByVal id As Integer)
    ' supprime le produit de clé [id]
    ' on prépare les paramètres de la suppression
    With deleteCommand.Parameters
      .Clear()
      .Add(New OleDbParameter("id", id))
    End With
    ' on fait la suppression
    Try
      ' ouverture connexion
      connexion.Open()
      ' exécution commande
      Dim nbLignes As Integer = deleteCommand.ExecuteNonQuery()
      If nbLignes = 0 Then Throw New Exception(String.Format("Le produit de clé [{0}] n'existe pas dans
la table des données", id))
    Catch ex As Exception
      ' pb
      Dim erreursCommande As New ArrayList
      erreursCommande.Add(String.Format("Erreur lors de la suppression : {0}", ex.Message))
      Throw New ExceptionProduits(erreursCommande)
    Finally
      ' fermeture connexion
      connexion.Close()
    End Try
  End Sub
```

On retrouve la même démarche que pour les méthodes précédentes.

3.5.4 Tests de la classe [produits]

Un programme de tests [testproduits.vb] de type console, pourrait être le suivant :

```
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.VisualBasic
Imports System.Collections

Namespace st.istia.univangers.fr

 ' pg de test
 Module testproduits
   Dim contenu As DataTable

   Sub Main(ByVal arguments() As String)
     ' affiche le contenu d'une table de produits
     ' la table est dans une base ACCESS dont le pg reçoit le nom de fichier
     Const syntaxe1 As String = "pg bdACCESS"

     ' vérification des paramètres du programme
     If arguments.Length <> 1 Then
       ' msg d'erreur
       Console.Error.WriteLine(syntaxe1)
       ' fin
       Environment.Exit(1)
     End If
     ' on prépare la chaîne de connexion
     Dim chaineConnexion As String = "Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data Source="
+ arguments(0)
     ' création d'un objet produits
     Dim objProduits As produits = New produits(chaineConnexion)
     ' on affiche tous les produits
     afficheProduits(objProduits)
     ' on insère un produit
     Dim produit As New sProduit
     With produit
```

```
      .nom = "xxx"
      .prix = 1
    End With
    Try
      objProduits.ajouterProduit(produit)
    Catch ex As ExceptionProduits
      afficheErreurs(ex.erreurs)
    End Try
    ' on affiche tous les produits
    afficheProduits(objProduits)
    ' on récupère l'id du produit ahjouté
    produit.id = CType(contenu.Rows(contenu.Rows.Count - 1)("id"), Integer)
    ' on modifie le produit ajouté
    produit.prix = 200
    Try
      objProduits.modifierProduit(produit)
    Catch ex As ExceptionProduits
      afficheErreurs(ex.erreurs)
    End Try
    ' on affiche tous les produits
    afficheProduits(objProduits)
    ' on supprime le produit ajouté
    Try
      objProduits.supprimerProduit(produit.id)
    Catch ex As ExceptionProduits
      afficheErreurs(ex.erreurs)
    End Try
    ' on affiche tous les produits
    afficheProduits(objProduits)
  End Sub

  Sub afficheProduits(ByRef objProduits As produits)
    ' on récupère la table des produits dans un datatable
    Try
      contenu = objProduits.getProduits()
    Catch ex As ExceptionProduits
      afficheErreurs(ex.erreurs)
      Environment.Exit(2)
    End Try
    Dim lignes As DataRowCollection = contenu.Rows
    For i As Integer = 0 To lignes.Count - 1
      ' ligne i de la table
      Console.Out.WriteLine(lignes(i).Item("id").ToString + "," + lignes(i).Item("nom").ToString + _
      "," + lignes(i).Item("prix").ToString)
    Next
    ' stoppe le flux console
    Console.WriteLine("...")
    Console.ReadLine()
  End Sub

  Sub afficheErreurs(ByRef erreurs As ArrayList)
    ' affiche les erreurs sur la console
    If erreurs.Count <> 0 Then
      Console.WriteLine("Les erreurs suivantes se sont produites :")
      For i As Integer = 0 To erreurs.Count - 1
        Console.WriteLine(String.Format("-- {0}", CType(erreurs(i), String)))
      Next
    End If
    ' stoppe le flux console
    Console.WriteLine("...")
    Console.ReadLine()
  End Sub
 End Module
End Namespace
```

On compile les deux fichiers source :

```
dos>vbc /t:library /r:system.dll /r:system.data.dll /r:system.xml.dll produits.vb

dos >vbc /r:produits.dll /r:system.dll /r:system.data.dll /r:system.xml.dll testproduits.vb

dos>dir
07/04/2004  08:40             7 168 produits.dll
04/04/2004  16:38           118 784 produits.mdb
07/04/2004  08:31             6 209 produits.vb
07/04/2004  08:40             5 120 testproduits.exe
03/04/2004  19:02             3 312 testproduits.vb
```

puis on teste :

```
dos>testproduits produits.mdb
1,produit1,10
2,produit2,20
3,produit3,30
...

1,produit1,10
2,produit2,20
3,produit3,30
8,xxx,1
...

1,produit1,10
2,produit2,20
3,produit3,30
8,xxx,200
...

1,produit1,10
2,produit2,20
3,produit3,30
...
```

Le lecteur est invité à rapprocher la sortie écran ci-dessus du code du programme de test.

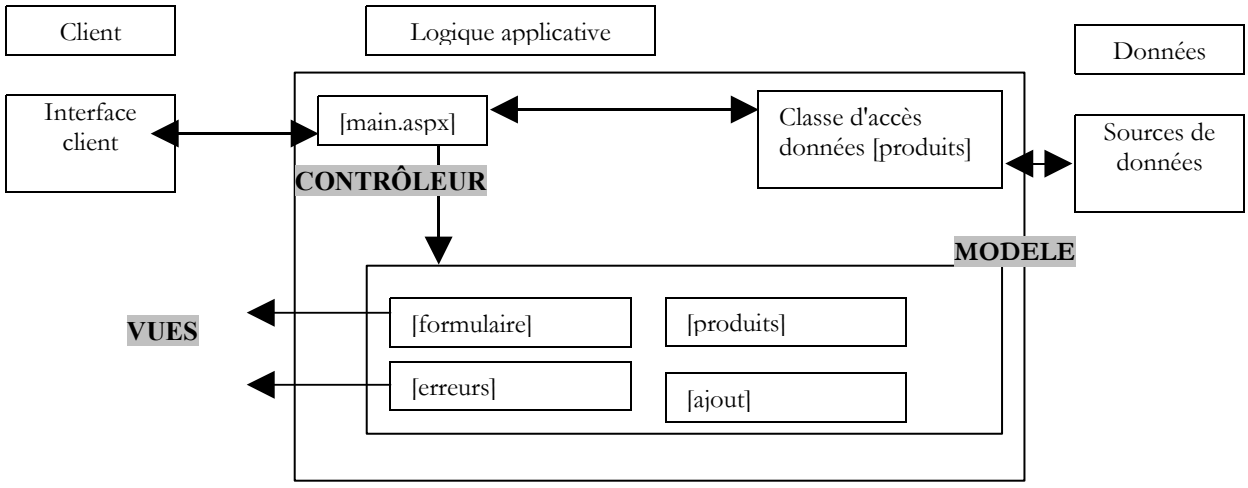
3.6 Application web de mise à jour de la table des produits en cache

3.6.1 Introduction

Nous écrivons maintenant une application web de mise à jour de la table des produits (ajout, suppression, modification). La table mise à jour restera en mémoire dans un objet [DataTable] et sera partagée par tous les utilisateurs. Nous voulons mettre en lumière deux points :

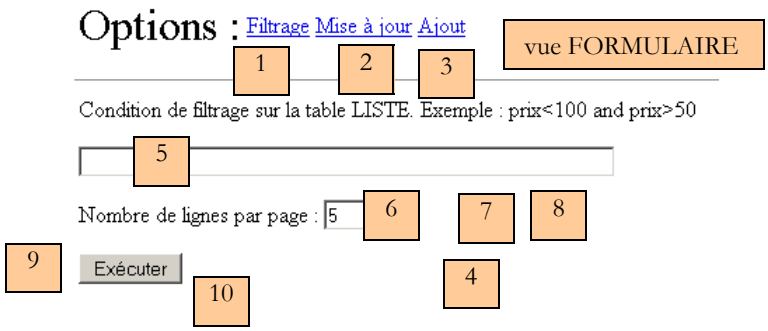
- la gestion d'un objet [DataTable]
- les problèmes de mise à jour simultanée d'une table par plusieurs utilisateurs.

L'architecture MVC de l'application sera la suivante :



3.6.2 Fonctionnement et vues

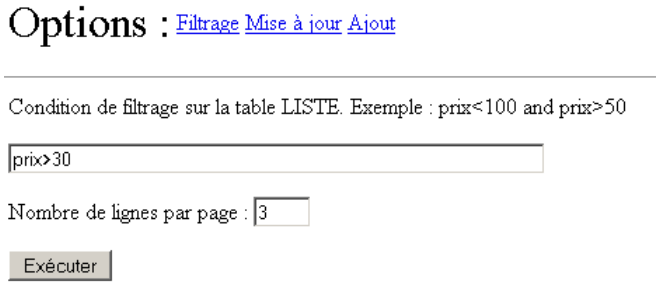
La vue d'accueil de l'application est la suivante :



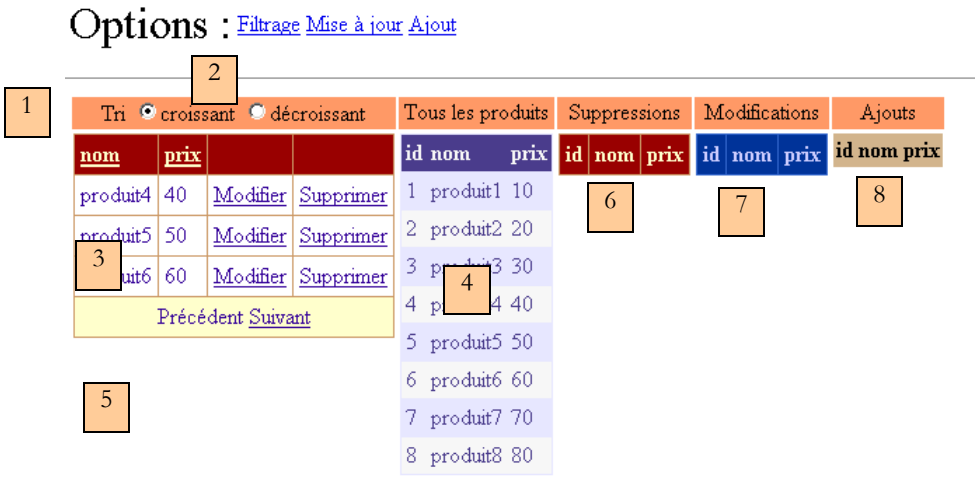
Cette vue appelée [formulaire] permet à l'utilisateur de poser une condition de filtrage sur les produits et de fixer le nombre de produits par page qu'il désire avoir.

n°	nom	type	rôle
1	lnkFiltre	LinkButton	affiche la vue [Formulaire] qui sert à fixer la condition de filtrage
2	lnkMisaJour	LinkButton	affiche la vue [Produits] qui sert à visualiser et à mettre à jour la table des produits (modification et suppression)
3	lnkAjout	LinkButton	affiche la vue [Ajout] qui sert à ajouter un produit
4	panel	vueFormulaire	la vue [Formulaire]
5	txtFiltre	TextBox	la condition de filtrage
6	txtPages	TextBox	nombre de produits par page
7	rfvLignes	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtPages]
8	rvLignes	RangeValidator	vérifie que txtPages est dans l'intervalle [3,10]
9	btnExécuter		bouton [submit] qui fait afficher la vue [produits] filtrée par la condition (5)
10	lblInfo1	Label	Texte d'information en cas d'erreurs

Par exemple, si la vue [formulaire] est remplie de la façon suivante :



on obtient le résultat suivant :



n°	nom	type	rôle
1	vueProduits	panel	
2	rdCroissant rdDécroissant	RadioButton	permet àl'utilisateur de fixer l'ordre du tri désiré lorsqu'il clique sur le titre d'une des colonnes [nom], [prix]. Les deux boutons font partie du groupe [rdTri].
3	DataGrid1	DataGrid	grille d'affichage d'une vue filtrée de la table des produits. Le filtre est celui fixé par la vue [formulaire]. On a également .AllowPaging=true, .AllowSorting=true
4	DataGrid2	DataGrid	affiche la totalité de la table des produits - permet de suivre les mises à jour
5	LblInfo2	Label	texte d'information notamment en cas d'erreurs
6	DataGrid3	DataGrid	affichera les produits supprimés dans la table des produits
7	DataGrid4	DataGrid	affichera les produits modifiés dans la table des produits
8	DataGrid5	DataGrid	affichera les produits ajoutés dans la table des produits

Il y a cinq conteneurs de données dans cette page. Ils affichent tous la même table [dtProduits] au travers d'une vue [DataView] différente. Une vue représente un sous-ensemble des lignes de la table source de la vue. Ce sous-ensemble est créé à partir des propriétés [RowFilter] et [RowStateFilter] de la classe [DataView] :

- [RowFilter] permet de fixer un filtre sur les lignes, comme par exemple ci-dessus [prix>30]. Ce type de filtrage sera utilisé par [DataGrid1].
- [RowStateFilter] permet de fixer un filtre selon l'état de la ligne de la table. Celui-ci indique l'état de la ligne par rapport à l'état originel qu'elle avait lorsque la vue sur la table a été créée. Ici la table [dtProduits] provient d'une base de données. Initialement toutes ses lignes auront un état égal à [Original] pour indiquer qu'on a affaire aux lignes originelles de la table. Cet état peut ensuite évoluer et prendre différentes valeurs dont voici quelques-unes :
  - [Added] : la ligne a été ajoutée - elle ne faisait pas partie de la table d'origine
  - [Deleted] : la ligne a été supprimée - elle est encore dans la table mais "marquée" comme "à supprimer"
  - [Modified] : la ligne a été modifiée

  [RowStateFilter] permet d'afficher les lignes de la table ayant un certain état :
  - [DataViewRowState.Added] : seules les lignes ajoutées sont affichées. Elles le sont avec leurs valeurs actuelles.
  - [DataViewRowState.ModifiedOriginal] : seules les lignes modifiées sont affichées. Elles le sont avec leurs valeurs d'origine.
  - [DataViewRowState.ModifiedCurrent] : seules les lignes modifiées sont affichées. Elles le sont avec leurs valeurs actuelles.
  - [DataViewRowState.Deleted] : seules les lignes supprimées sont affichées. Elles le sont avec leurs valeurs d'origine.
  - [DataViewRowState.CurrentRows] : les lignes non supprimées sont affichées. Elles le sont avec leurs valeurs actuelles.

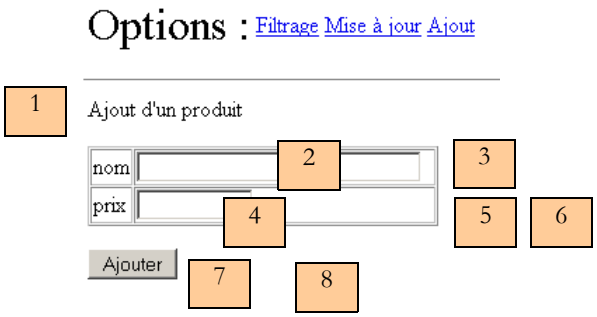
Ainsi le filtre [RowStateFilter] aura les valeurs suivantes :

DataGrid1	pas de filtre sur l'état des lignes	
DataGrid2	DataViewRowState.CurrentRows	affiche l'état actuel de la table des produits

DataGrid3	DataViewRowState.Deleted	affiche les lignes supprimées de la table des produits
DataGrid4	DataViewRowState.ModifiedOriginal	affiche les lignes modifiées de la table des produits avec leurs valeurs initiales
DataGrid5	DataViewRowState.Added	affiche les lignes ajoutées à la table des produits initiale

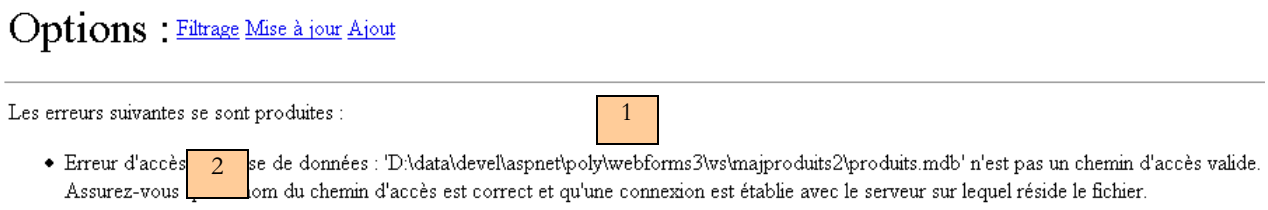
Les conteneurs [DataGrid1-5] vont nous permettre de suivre la mise à jour de la table [dtProduits]. Le composant [DataGrid1] permet la modification et la suppression d'un produit. Nous verrons comment la configuration du composant permet cette mise à jour. Il est également paginé et trié. Ces deux points ont déjà été étudiés dans un précédent exemple.

Le lien [Ajout] donne accès à un formulaire d'ajout de produit :



n°	nom	type	rôle
1	vueAjout	panel	
2	txtNom	TextBox	nom du produit
3	rfvNom	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtNom]
4	txtPrix	TextBox	prix du produit
5	rfvPrix	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtPrix]
6	cvPrix	CompareValidator	vérifie que prix>=0
7	btnAjouter	Button	bouton [submit] d'ajout du produit
8	lblInfo3	Label	texte d'information sur le résultat de l'opération d'Ajout

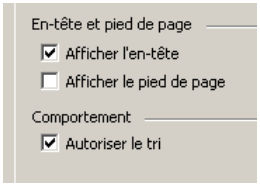
La base de données [produits] peut être indisponible au démarrage de l'application. Dans ce cas, la vue [erreurs] est présentée à l'utilisateur :



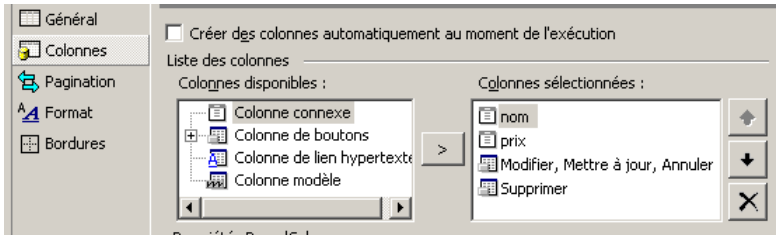
n°	nom	type	rôle
1	vueErreurs	panel	
2	rptErreurs	Repeater	liste des erreurs

3.6.3 Configuration des conteneurs de données

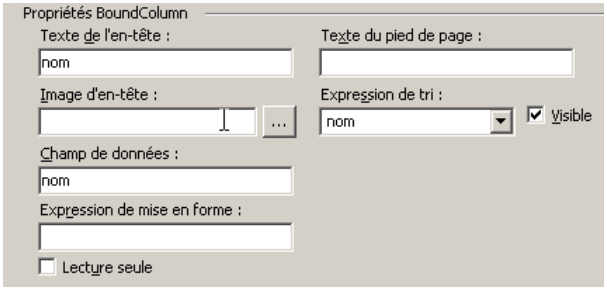
Les cinq conteneurs [DataGrid] ont été configurés sous [WebMatrix]. Ils ont été mis en forme (couleurs et bordures) avec le lien [Configuration automatique] de leur panneau de propriétés. Le conteneur [DataGrid1] a été configuré à l'aide du lien [Générateur de propriétés] de ce même panneau. On a autorisé le tri (onglet [Général]) :



Les colonnes ne sont pas générées automatiquement contrairement aux quatre autres conteneurs. Elles ont été définies à la main à l'aide de l'assistant :

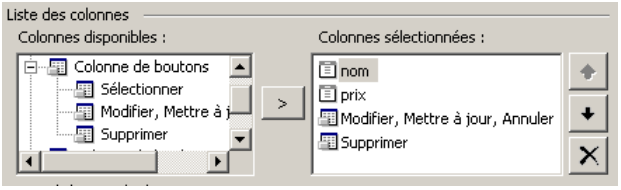


On a tout d'abord créé deux colonnes de type [Colonne connexe] appelées [nom] et [prix]. Elles ont été associées respectivement aux champs [nom] et [prix] de la source de données que les conteneurs afficheront. Voici par exemple, la configuration de la colonne [nom] :

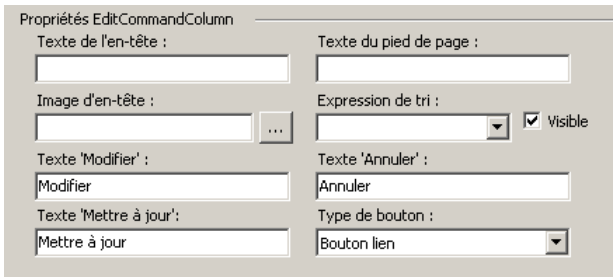


L'expression de tri est l'expression qui devra être mise derrière la clause [order by] de l'instruction SQL [select] qui sera exécutée lorsque l'utilisateur cliquera sur le nom de la colonne d'entête [nom] associée au champ [nom] du [DataGrid]. Ici nous avons mis [nom] de sorte que la clause de tri sera [order by nom]. Nous verrons que nous modifierons celle-ci afin qu'elle soit [order by nom asc] ou [order by nom desc] selon l'ordre de tri choisi par l'utilisateur.

Nous avons créé par ailleurs, deux colonnes de boutons :

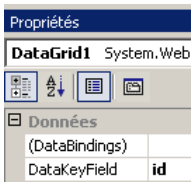


La colonne [Modifier, Mettre à jour, Annuler] va nous permettre de modifier un produit et la colonne [Supprimer] de le supprimer. Chacune de ces colonnes peut être configurée. La colonne [Modifier, Mettre à jour, Annuler] offre la configuration suivante :



On voit qu'on peut modifier les textes des boutons. En fait de boutons, on a le choix entre liens et boutons (liste déroulante ci-dessus). Ce sont les liens qui ont été choisis ici. La configuration de la colonne [Supprimer] est analogue. Outre cet assistant de configuration, nous avons utilisé directement la fenêtre de propriétés du [DataGrid] pour renseigner la propriété [DataKeyField] qui

indique avec quel champ de la source de donnéees, sont indexées les lignes du [DataGrid]. Ici c'est la clé primaire de la table de produits qui est utilisée :



Au final, cette configuration génère le code de présentation suivant :

```
<asp:DataGrid id="DataGrid1" runat="server" AllowSorting="True" PageSize="4" AllowPaging="True"
AutoGenerateColumns="False" DataKeyField="id">
  <SelectedItemStyle ...></SelectedItemStyle>
  <HeaderStyle ...></HeaderStyle>
  <FooterStyle ...></FooterStyle>
  <Columns>
       <asp:BoundColumn DataField="nom" SortExpression="nom" HeaderText="nom"></asp:BoundColumn>
     <asp:BoundColumn DataField="prix" SortExpression="prix" HeaderText="prix"></asp:BoundColumn>
       <asp:EditCommandColumn ButtonType="LinkButton" UpdateText="Mettre &#224; jour"
CancelText="Annuler" EditText="Modifier"></asp:EditCommandColumn>
     <asp:ButtonColumn Text="Supprimer" CommandName="Delete"></asp:ButtonColumn>
  </Columns>
<PagerStyle NextPageText="Suivant" PrevPageText="Pr&#233;c&#233;dent" ...></PagerStyle>
</asp:DataGrid>
```

Comme toujours, une fois une certaine expérience acquise, on peut écrire tout ou partie du code ci-dessus directement.

Les autres conteneurs [DataGrid] ont la configuration par défaut obtenue par la génération automatique des colonnes du [DataGrid] à partir de celles de la source de données à laquelle il est associé.

Le conteneur [Repeater] sert à afficher une liste d'erreurs. Sa configuration est faite directement dans le code de présentation :

```
            <asp:Repeater id="rptErreurs" runat="server" EnableViewState="False">
              <HeaderTemplate>
                Les erreurs suivantes se sont produites :
                <ul>
              </HeaderTemplate>
              <ItemTemplate>
                <li>
                  <%# Container.DataItem %>
                </li>
              </ItemTemplate>
              <FooterTemplate>
                </ul>
              </FooterTemplate>
            </asp:Repeater>
```

Chaque ligne du composant affiche la valeur [Container.DataItem], c.a.d. la valeur correspondante de la liste de données. Celle-ci sera de type [ArrayList] et représentera une liste d'erreurs.

3.6.4 Le code de présentation de l'application

Celui-ci est placé dans le fichier [main.aspx] :

```
<%@ Page src="main.aspx.vb" inherits="main" autoeventwireup="false" Language="vb" %>
<HTML>
 <HEAD>
 </HEAD>
 <body>
   <form id="Form1" runat="server">
     <P>
       <table>
         <tr>
           <td><FONT size="6">Options :</FONT></td>
           <td>
             <asp:linkbutton id="lnkFiltre" runat="server" CausesValidation="False">
                 Filtrage
             </asp:linkbutton>
           </td>
           <td>
             <asp:linkbutton id="lnkMisajour" runat="server" CausesValidation="False">
               Mise à jour
```

```
            </asp:linkbutton>
          </td>
          <td>
            <asp:linkbutton id="lnkAjout" runat="server" CausesValidation="False">
               Ajout
            </asp:linkbutton>
          </td>
        </tr>
      </table>
    </P>
    <HR width="100%" SIZE="1">
    <table>
      <tr>
        <td>
```

```
          <asp:panel id="vueFormulaire" runat="server">
            <P>Condition de filtrage sur la table LISTE. Exemple : prix&lt;100 and
              prix&gt;50</P>
            <P>
              <asp:TextBox id="txtFiltre" runat="server" Columns="60"></asp:TextBox></P>
            <P>Nombre de lignes par page :
              <asp:TextBox id="txtPages" runat="server" Columns="3">5</asp:TextBox>
              <asp:RequiredFieldValidator id="rfvLignes" runat="server" Display="Dynamic"
                 ControlToValidate="txtPages" ErrorMessage="Indiquez le nombre de lignes par page"
                EnableClientScript="False">
              </asp:RequiredFieldValidator></P>
            <P>
              <asp:RangeValidator id="rvLignes" runat="server" Display="Dynamic"
                ControlToValidate="txtPages" ErrorMessage="Vous devez indiquer un nombre entre 3 et 10"
                EnableClientScript="False" MaximumValue="10" MinimumValue="3" Type="Integer"
                  EnableViewState="False">
                </asp:RangeValidator></P>
            <P>
              <asp:Label id="lblinfo1" runat="server"></asp:Label></P>
            <P>
              <asp:Button id="btnExécuter" runat="server" CausesValidation="False"
                EnableViewState="False" Text="Exécuter">
                </asp:Button></P>
          </asp:panel>
```

```
          <asp:panel id="vueProduits" runat="server">
            <TABLE>
              <TR>
                <TD align="center" bgColor="#ff9966">
                  <P>Tri
                    <asp:RadioButton id="rdCroissant" runat="server" Text="croissant"
                      GroupName="rdTri" Checked="True">
                    </asp:RadioButton>
                    <asp:RadioButton id="rdDécroissant" runat="server" Text="décroissant"
                       GroupName="rdTri">
                      </asp:RadioButton></P>
                </TD>
                <TD align="center" bgColor="#ff9966">Tous les produits
                </TD>
                <TD align="center" bgColor="#ff9966">Suppressions
                </TD>
                <TD align="center" bgColor="#ff9966">Modifications
                </TD>
                <TD align="center" bgColor="#ff9966">Ajouts
                </TD>
              <TR>
                <TD vAlign="top">
                  <P>
                    <asp:DataGrid id="DataGrid1" runat="server" AllowSorting="True" PageSize="4"
                       AllowPaging="True" ....  AutoGenerateColumns="False" DataKeyField="id">
                      <ItemStyle ...></ItemStyle>
                      <HeaderStyle ...></HeaderStyle>
                      <FooterStyle ...></FooterStyle>
                      <Columns>
                        <asp:BoundColumn DataField="nom" SortExpression="nom" HeaderText="nom">
                          </asp:BoundColumn>
                        <asp:BoundColumn DataField="prix" SortExpression="prix" HeaderText="prix">
                          </asp:BoundColumn>
                        <asp:EditCommandColumn ButtonType="LinkButton" UpdateText="Mettre &#224; jour"
                           CancelText="Annuler" EditText="Modifier">
                        </asp:EditCommandColumn>
                        <asp:ButtonColumn Text="Supprimer" CommandName="Delete">
                        </asp:ButtonColumn>
                      </Columns>
                      <PagerStyle NextPageText="Suivant" PrevPageText="Pr&#233;c&#233;dent" ....>
                        </PagerStyle>
```

```
                    </asp:DataGrid>
                    </P>
                  </TD>
                  <TD vAlign="top">
                    <asp:DataGrid id="DataGrid2" runat="server" ...>
                      <AlternatingItemStyle ...></AlternatingItemStyle>
                      <ItemStyle ...></ItemStyle>
                      <HeaderStyle ....></HeaderStyle>
                      <FooterStyle ...></FooterStyle>
                      <PagerStyle ... Mode="NumericPages"></PagerStyle>
                    </asp:DataGrid>
                    </TD>
                  <TD vAlign="top">
                    <asp:DataGrid id="DataGrid3" runat="server" ...>
                      <ItemStyle ...></ItemStyle>
                      <HeaderStyle ...></HeaderStyle>
                      <FooterStyle ...></FooterStyle>
                      <PagerStyle ...></PagerStyle>
                    </asp:DataGrid>
                    </TD>
                  <TD vAlign="top">
                    <asp:DataGrid id="DataGrid4" runat="server" ...>
                      <ItemStyle ....></ItemStyle>
                      <HeaderStyle ....></HeaderStyle>
                      <FooterStyle ...></FooterStyle>
                      <PagerStyle .... Mode="NumericPages"></PagerStyle>
                    </asp:DataGrid>
                  </TD>
                  <TD vAlign="top">
                    <asp:DataGrid id="DataGrid5" runat="server....>
                      <AlternatingItemStyle ...></AlternatingItemStyle>
                      <HeaderStyle ..></HeaderStyle>
                      <FooterStyle ...></FooterStyle>
                      <PagerStyle ....></PagerStyle>
                    </asp:DataGrid>
                  </TD>
                </TR>
              </TABLE>
              <P></P>
              <P>
                <asp:Label id="lblInfo2" runat="server"></asp:Label></P>
            </asp:panel>
```

```
            <asp:panel id="vueErreurs" runat="server">
              <asp:Repeater id="rptErreurs" runat="server" EnableViewState="False">
                <HeaderTemplate>
                  Les erreurs suivantes se sont produites :
                  <ul>
                </HeaderTemplate>
                <ItemTemplate>
                  <li>
                    <%# Container.DataItem %>
                  </li>
                </ItemTemplate>
                <FooterTemplate>
                  </ul>
                </FooterTemplate>
              </asp:Repeater>
            </asp:panel>
```

```
            <asp:panel id="vueAjout" EnableViewState="False" Runat="server">
              <P>Ajout d'un produit</P>
              <P>
                <TABLE ... border="1">
                  <TR>
                    <TD>nom</TD>
                    <TD>
                      <asp:TextBox id="txtNom" runat="server" Columns="30"></asp:TextBox>
                      <asp:RequiredFieldValidator id="rfvNom" runat="server" ControlToValidate="txtNom"
                           ErrorMessage="Vous devez indiquer un nom">
                      </asp:RequiredFieldValidator>
                    </TD>
                  </TR>
                  <TR>
                    <TD>prix</TD>
                    <TD>
                      <asp:TextBox id="txtPrix" runat="server" Columns="10"></asp:TextBox>
                      <asp:RequiredFieldValidator id="rfvPrix" runat="server" ControlToValidate="txtPrix"
                          ErrorMessage="Vous devez indiquer un prix">
                        </asp:RequiredFieldValidator>
                      <asp:CompareValidator id="cvPrix" runat="server" ControlToValidate="txtPrix"
```

```
                        ErrorMessage="CompareValidator" Type="Double" Operator="GreaterThanEqual"
                        ValueToCompare="0">
                    </asp:CompareValidator>
                    </TD>
                  </TR>
                </TABLE>
              </P>
              <P>
                <asp:Button id="btnAjouter" runat="server" CausesValidation="False"
                    Text="Ajouter">
                  </asp:Button></P>
              <P>
                <asp:Label id="lblInfo3" runat="server"></asp:Label></P>
            </asp:panel>
```

```
        </td>
      </tr>
    </table>
  </form>
 </body>
</HTML>
```

Notons les quelques points suivants :

- la page est composée de quatre conteneurs (panel) [vueFormulaire, vueProduits, vueAjout, vueErreurs] qui vont former les quatre vues de l'application.
- les boutons ou liens de type [submit] ont la propriété [CausesValidation=false]. La propriété [causesValidation=true] provoque l'exécution de tous les contrôles de validation de la page. Or ici, tous les contrôles de validité ne sont pas à faire en même temps. Lorsqu'on fait un ajout par exemple, on ne veut pas que les contrôles concernant le nombre de lignes par page soient exécutés. On précisera donc nous-mêmes quels contrôles de validité doivent être faits.

3.6.5 Le code de contrôle [global.asax]

Le contrôleur [global.asax] est le suivant :

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

Le code associé [global.asax.vb] :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration
Imports System.Data
Imports Microsoft.VisualBasic
Imports System.Collections

Public Class Global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on récupère les informations de configuration
   Dim chaînedeConnexion As String = ConfigurationSettings.AppSettings("OLEDBStringConnection")
   Dim defaultProduitsPage As String = ConfigurationSettings.AppSettings("defaultProduitsPage")
   Dim erreurs As New ArrayList
   If IsNothing(chaînedeConnexion) Then erreurs.Add("Le paramètre [OLEDBStringConnection] n'a pas été
initialisé")
   If IsNothing(defaultProduitsPage) Then
     erreurs.Add("Le paramètre [defaultProduitsPage] n'a pas été initialisé")
   Else
     Try
       Dim defProduitsPage As Integer = CType(defaultProduitsPage, Integer)
       If defProduitsPage <= 0 Then Throw New Exception
     Catch ex As Exception
       erreurs.Add("Le paramètre [defaultProduitsPage] a une valeur incorrecte")
     End Try
   End If
   ' des erreurs de configuration ?
   If erreurs.Count <> 0 Then
     ' on note les erreurs
     Application("erreurs") = erreurs
     ' on quitte
     Exit Sub
   End If
   ' ici pas d'erreurs de configuration
   ' on crée un objet produits
```

```
    Dim dtProduits As DataTable
    Try
      dtProduits = New produits(chaînedeConnexion).getProduits
    Catch ex As ExceptionProduits
      'il y a eu erreur d'accès aux produits, on le note dans l'application
      Application("erreurs") = ex.erreurs
      Exit Sub
    Catch ex As Exception
      ' erreur non gérée
      erreurs.Add(ex.Message)
      Application("erreurs") = erreurs
      ' exit sub
    End Try
    ' ici pas d'erreurs d'initialisation
    ' on mémorise le nbre de produits par page
    Application("defaultProduitsPage") = defaultProduitsPage
    ' on mémorise la table des produits
    Application("dtProduits") = dtProduits
  End Sub

  Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' init des variables de session
    If IsNothing(Application("erreurs")) Then
      ' vue sur la table des produits
      Session("dvProduits") = CType(Application("dtProduits"), DataTable).DefaultView
      ' nbre de produits par page
      Session("nbProduitsPage") = Application("defaultProduitsPage")
      ' page courante affichée
      Session("pageCourante") = 0
    End If
  End Sub
End Class
```

Dans [Application_Start], on commence par récupérer deux informations du fichier de configuration [web.config] de l'application :

- OLEDBStringConnection : la chaîne OLEDB de connexion à la base des produits
- defaultProduitsPage : le nombre par défaut de produits par page affichée

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="OLEDBStringConnection" value="Provider=Microsoft.Jet.OLEDB.4.0; Ole DB Services=-4; Data
Source=D:\data\devel\aspnet\poly\webforms3\vs\majproduits1\produits.mdb" />
    <add key="defaultProduitsPage" value="5" />
  </appSettings>
</configuration>
```

Si l'une de ces deux informations est absente, une liste d'erreurs est alimentée et mise dans l'application. Il en est de même si le paramètre [defaultProduitsPage] existe mais est incorrect. Si les deux paramètres attendus sont présents et corrects, une table [dtProduits] est construite et mise dans l'application. C'est cette table qui sera exploitée et mise à jour par les différents clients. La base de données elle, restera inchangée. On s'intéressera à la mise à jour de celle-ci dans une prochaine application. Cette table est construite à partir d'une instance de la classe [produits] étudiée précédemment et de la méthode [getProduits] de celle-ci. L'obtention de la table [dtProduits] peut échouer. Dans ce cas, on sait que la classe [produits] lance une exception de type [ExceptionProduits]. Elle est ici interceptée et la liste d'erreurs associée est mise dans l'application associée à la clé [erreurs]. La présence de cette clé dans les informations enregistrées dans l'application sera testée à chaque traitement de requête. Si elle est trouvée, la vue [erreurs] sera envoyée au client.

Si la table [dtProduits] est partagée par tous les clients web, chacun d'eux aura néanmoins sa propre vue [dvProduits] sur celle-ci. En effet, chaque client web a la possibilité de fixer un filtre sur la table [dtProduits] ainsi qu'un ordre de tri. Ces informations propres à chaque client web sont stockées dans la vue du client. Aussi celle-ci est-elle créée dans [Session_Start] afin d'être placée dans la session propre à chaque client. On utilise l'attribut [DefaultView] de la table [dtProduits] pour avoir une vue par défaut sur la table. Au départ, il n'y a ni filtre ni ordre de tri. Par ailleurs, deux informations sont également mises dans la session :

- le nombre de produits par page de clé [nbProduitsPage]. Au démarrage de la session, ce nombre est égal au nombre par défaut défini dans le fichier de configuration.
- le numéro de la page courante des produits. Au départ, c'est la première page.

3.6.6 Le code de contrôle [main.aspx.vb]

Le squelette du contrôleur est le suivant :

```
Imports System.Collections
Imports Microsoft.VisualBasic
Imports System.Data
```

```
Imports st.istia.univangers.fr
Imports System
Imports System.Xml
Imports System.Web.UI.WebControls

Public Class main
 Inherits System.Web.UI.Page

 ' composants page
 Protected WithEvents txtPages As System.Web.UI.WebControls.TextBox
 Protected WithEvents btnExécuter As System.Web.UI.WebControls.Button
 Protected WithEvents vueFormulaire As System.Web.UI.WebControls.Panel
 Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents lnkErreurs As System.Web.UI.WebControls.LinkButton
 Protected WithEvents vueErreurs As System.Web.UI.WebControls.Panel
 Protected WithEvents rdCroissant As System.Web.UI.WebControls.RadioButton
 Protected WithEvents rdDécroissant As System.Web.UI.WebControls.RadioButton
 Protected WithEvents txtFiltre As System.Web.UI.WebControls.TextBox
 Protected WithEvents lblInfo2 As System.Web.UI.WebControls.Label
 Protected WithEvents lblinfo1 As System.Web.UI.WebControls.Label
 Protected WithEvents lblErreurs As System.Web.UI.WebControls.Label
 Protected WithEvents DataGrid2 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents vueProduits As System.Web.UI.WebControls.Panel
 Protected WithEvents vueAjout As System.Web.UI.WebControls.Panel
 Protected WithEvents lnkMisajour As System.Web.UI.WebControls.LinkButton
 Protected WithEvents lnkFiltre As System.Web.UI.WebControls.LinkButton
 Protected WithEvents txtNom As System.Web.UI.WebControls.TextBox
 Protected WithEvents txtPrix As System.Web.UI.WebControls.TextBox
 Protected WithEvents btnAjouter As System.Web.UI.WebControls.Button
 Protected WithEvents lblInfo3 As System.Web.UI.WebControls.Label
 Protected WithEvents rfvLignes As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents rvLignes As System.Web.UI.WebControls.RangeValidator
 Protected WithEvents rfvNom As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents rfvPrix As System.Web.UI.WebControls.RequiredFieldValidator
 Protected WithEvents cvPrix As System.Web.UI.WebControls.CompareValidator
 Protected WithEvents lnkAjout As System.Web.UI.WebControls.LinkButton
 Protected WithEvents DataGrid3 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents DataGrid4 As System.Web.UI.WebControls.DataGrid
 Protected WithEvents DataGrid5 As System.Web.UI.WebControls.DataGrid

 ' données page
 Protected dtProduits As DataTable
 Protected dvProduits As DataView
 Protected defaultProduitsPage As Integer
 Protected erreur As Boolean = False

 ' chargement page
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
...
 End Sub

 Private Sub afficheErreurs()
...
 End Sub

 Private Sub afficheFormulaire()
...
 End Sub

 Private Sub btnExécuter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnExécuter.Click
....
 End Sub

 Private Sub afficheProduits(ByVal page As Integer, ByVal taillePage As Integer)
...
 End Sub

 Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles DataGrid1.PageIndexChanged
...
 End Sub

 Private Sub DataGrid1_SortCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridSortCommandEventArgs) Handles DataGrid1.SortCommand
....
 End Sub

 Private Sub DataGrid1_DeleteCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.DeleteCommand
....
```

```
 End Sub

 Private Sub DataGrid1_EditCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.EditCommand
...
 End Sub

 Private Sub DataGrid1_CancelCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.CancelCommand
...
 End Sub

 Private Sub DataGrid1_UpdateCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.UpdateCommand
...
 End Sub

 Private Sub supprimerProduit(ByVal idProduit As Integer)
...
 End Sub

 Private Sub modifierProduit(ByVal idProduit As Integer, ByVal item As DataGridItem)
....
 End Sub

 Private Sub lnkMisajour_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkMisajour.Click
...
 End Sub

 Private Sub lnkAjout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkAjout.Click
...
 End Sub

 Private Sub afficheAjout()
...
 End Sub

 Private Sub lnkFiltre_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkFiltre.Click
....
 End Sub

 Private Sub btnAjouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAjouter.Click
....
 End Sub
End Class
```

3.6.7 Les données d'instance

La classe [main] utilise les données d'instance suivantes :

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

 ' composants page
 Protected WithEvents txtPages As System.Web.UI.WebControls.TextBox
...

 ' données page
 Protected dtProduits As DataTable
 Protected dvProduits As DataView
 Protected defaultProduitsPage As Integer
```

`dtProduits`	la table [DataTable] des produits - commune à tous les clients
`dvProduits`	la vue [DataView] des produits - propre à chaque client
`defaultProduitsPage`	nbre de produits par page proposé par défaut

3.6.8 La procédure [Page_Load] de chargement de la page

Le code de la procédure [page_Load] est le suivant :

```
' chargement page
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
  ' on regarde si l'application est en erreur
  If Not IsNothing(Application("erreurs")) Then
    ' l'application ne s'est pas initialisée correctement
    afficheErreurs(CType(Application("erreurs"), ArrayList))
    Exit Sub
  End If
  ' on récupère une référence sur la table des produits
  dtProduits = CType(Application("dtProduits"), DataTable)
  ' on récupère la vue sur les produits
  dvProduits = CType(Session("dvProduits"), DataView)
  ' on récupère le nbre de produits par page
  defaultProduitsPage = CType(Application("defaultProduitsPage"), Integer)
  ' on annule une éventuelle mise à jour en cours
  DataGrid1.EditItemIndex = -1
  '1ère requête
  If Not IsPostBack Then
    ' on affiche le formulaire initial
    txtPages.Text = defaultProduitsPage.ToString
    afficheFormulaire()
  End If
End Sub
```

- on commence tout d'abord par vérifier si la table des produits a pu être chargée au démarrage de l'application. Si ce n'est pas le cas, on affiche la vue [erreurs] avec les messages d'erreurs appropriés puis on quitte la procédure après avoir positionné le booléen [erreur] à vrai. Cet indicateur sera testé par certaines procédures.
- on récupère la table des produits [dtProduits] dans l'application et on la met dans la variable d'instance [dtProduits] afin qu'elle soit partagée par toutes les méthodes de la page.
- on fait de même avec la vue [dvProduits] récupérée elle dans la session et le nombre par défaut de produits par page récupéré lui dans l'application.
- si c'est la 1ère requête du client, on lui présente le formulaire de définition des conditions de filtrage et de pagination avec une pagination par défaut de [defaultProduitsPage] produits par page.
- on annule le mode édition du composant [dataGrid1]. Ce composant a un attribut [EditItemIndex] qui est l'index de l'élément en cours de modification. Si [EditItemIndex]=-1, alors il n'y a aucun élément en cours de modification. Si [EditItemIndex]=i, alors l'élément n° i du composant [DataGrid] est en cours d'édition. Il est alors présenté différemment des autres éléments du [dataGrid] :

nom	prix		
produit6	60	Modifier	Supprimer
produit7	70	Modifier	Supprimer
produit8	80	Mettre à jour Annuler	Supprimer
xx	11	Modifier	Supprimer
Précédent Suivant			

Ci-dessus, l'élément [Produit8, 80] est en mode [édition]. On voit ci-dessus que l'utilisateur peut valider sa mise à jour par le lien [Mettre à jour], l'annuler par le lien [Annuler]. Il peut aussi utiliser d'autres liens n'ayant rien à voir avec l'élément en cours d'édition. En mettant [DataGrid1.EditItemIndex=-1] à chaque chargement de la page, on s'assure d'annuler systématiquement le mode édition de [DataGrid1]. On ne lui affectera une valeur différente que si un lien [Modifier] a été cliqué. On le fera dans la procédure traitant cet événement. On aura les situations suivantes :

1. le lien [Modifier] de l'élément n° 8 de [DataGrid1] a été cliqué. [Page_Load] met tout d'abord -1 dans [DataGrid1.EditItemIndex]. Puis la procédure traitant l'événement [Modifier] mettra 8 dans [DataGrid1.EditItemIndex]. Au final, lorsque la page sera renvoyée au client, l'élément n° 8 sera bien en mode [édition] et celui-ci apparaîtra comme montré ci-dessus.
2. l'utilisateur modifie le produit qui est en mode [édition] et valide sa modification par le lien [Mettre à jour]. [Page_Load] met -1 dans [DataGrid1.EditItemIndex]. Puis la procédure traitant l'événement [Mettre à jour] s'exécutera et l'élément n° 8 de [dtProduits] sera mis à jour. Lorsque la page sera renvoyée au client, aucun élément de [DataGrid1] ne sera en mode édition (DataGrid.EditItemIndex=-1).
3. Si un utilisateur est entré en mise à jour d'un produit, il serait cohérent qu'il abandonne cette mise à jour par [Annuler]. Cependant rien ne l'empêche à cliquer sur un autre lien. Il nous faut donc prévoir ce cas. Dans ce cas comme dans les précédents, [Page_Load] commence par mettre -1 dans [DataGrid1.EditItemIndex], puis la procédure traitant l'événement qui s'est produit s'exécutera. Elle ne modifiera pas la propriété [DataGrid1.EditItemIndex] qui restera donc avec la valeur -1. Lorsque la page sera renvoyée au client, aucun élément de [DataGrid1] ne sera en mode édition.

On voit qu'en mettant [EditItemIndex] à -1 au chargement de la page, on évite d'avoir à se préoccuper si l'utilisateur était ou non en mise à jour lorsqu'il a cliqué sur un lien.

3.6.9 Affichage des vues [erreurs], [formulaire] et [ajout]

L'affichage de la vue [erreurs] est demandée au chargement de la page [Page_Load] si on découvre que l'application s'est mal initialisée. Cet affichage se fait en associant le composant de données [rptErreurs] à la liste d'erreurs passée en paramètres et en rendant visible le conteneur approprié. Enfin, les trois liens [Filtre, Mise à jour, Ajout] sont cachés car en cas d'erreur l'application ne peut être utilisée.

```
Private Sub afficheErreurs(ByVal erreurs As ArrayList)
  ' on associe la liste d'erreurs au répéteur rptErreurs
  With rptErreurs
    .DataSource = erreurs
    .DataBind()
  End With
  'on inhibe les options
  lnkAjout.Visible = False
  lnkMisajour.Visible = False
  lnkFiltre.Visible = False
  ' on affiche la vue [erreurs]
  vueErreurs.Visible = True
  vueFormulaire.Visible = False
  vueProduits.Visible = False
  vueAjout.Visible = False
End Sub
```

Les autres vues sont demandées à partir des options proposées à l'utilisateur :

Options : Filtrage Mise à jour Ajout

L'affichage de la vue [Ajout] est demandé lorsque l'utilisateur clique sur le lien [Ajout] de la page :

```
Private Sub lnkAjout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkAjout.Click
  ' on affiche la vue [ajout]
  afficheAjout()
End Sub
```

```
Private Sub afficheAjout()
  ' affiche la vue ajout
  vueAjout.Visible = True
  vueFormulaire.Visible = False
  vueProduits.Visible = False
  vueErreurs.Visible = False
End Sub
```

L'affichage de la vue [Formulaire] est demandé lorsque l'utilisateur clique sur le lien [Filtrage] de la page. On doit alors présenter la vue permettant de définir

- le filtre sur la table des produits
- le nombre de produits présentés dans chaque page web

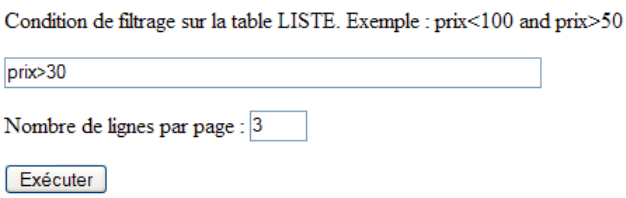
On présente un formulaire reprenant les valeurs stockées dans la session :

```
Private Sub lnkFiltre_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkFiltre.Click
  ' définir les conditions de filtrage
  txtFiltre.Text = dvProduits.RowFilter
  ' définir la pagination
  txtPages.Text = CType(Session("nbProduitsPage"), String)
  ' on affiche le formulaire de filtrage
  afficheFormulaire()
End Sub
```

La condition de filtrage d'une vue est définie dans son attribut [RowFilter]. Rappelons que la vue filtré et paginée s'appelle [dvProduits] et a été récupérée dans la session au chargement de la page [Page_Load]. La condition de filtrage est donc récupérée dans [dvProduits.RowFilter]. Le nombre de produits par page est également récupéré dans la session. Si l'utilisateur n'a jamais défini cette information, ce nombre est égal au nombre par défaut de produits par page. Ceci fait, on affiche le formulaire permettant de définir ces deux informations :

```
Private Sub afficheFormulaire()
  ' on affiche la vue [formulaire]
  vueFormulaire.Visible = True
```

```
    vueErreurs.Visible = False
    vueProduits.Visible = False
    vueAjout.Visible = False
End Sub
```



3.6.10 Validation de la vue [formulaire]

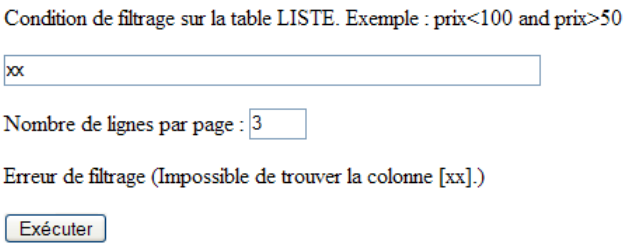
Le clic sur le bouton [Exécuter] ci-dessus est traité par la procédure suivante :

```
 Private Sub btnExécuter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnExécuter.Click
    ' page valide ?
    rfvLignes.Validate()
    rvLignes.Validate()
    If Not rfvLignes.IsValid Or Not rvLignes.IsValid Then
      afficheFormulaire()
      Exit Sub
    End If
    ' attachement des données filtrées à la grille
    Try
      dvProduits.RowFilter = txtFiltre.Text.Trim
    Catch ex As Exception
      lblinfo1.Text = "Erreur de filtrage (" + ex.Message + ")"
      afficheFormulaire()
      Exit Sub
    End Try
    ' on mémorise le nbre de produits par page
    Session("nbProduitsPage") = txtPages.Text
    'et la page courante
    Session("pageCourante") = 0
    ' tout va bien - on affiche les données
    afficheProduits(0, CType(txtPages.Text, Integer))
 End Sub
```

Rappelons tout d'abord que la page a deux composants de validation :

n°	nom	type	rôle
7	rfvLignes	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtPages]
8	rvLignes	RangeValidator	vérifie que txtPages est dans l'intervalle [3,10]

La procédure commence par faire exécuter le code de contrôle des deux composants ci-dessus à l'aide de leur méthode [Validate] puis vérifie la valeur de leur attribut [IsValid]. Celui-ci est à [true] uniquement si la donnée vérifiée a été trouvée valide. Si les tests de validité d'un des deux composants a échoué, on réaffiche la vue [formulaire] afin que l'utlisateur corrige son ou ses erreurs. La condition de filtrage est appliquée à l'attribut [dvProduits.RowFilter]. Ici une exception peut se produire si l'utilisateur a mis un critère de filtrage incorrect comme il est montré ci-dessous :



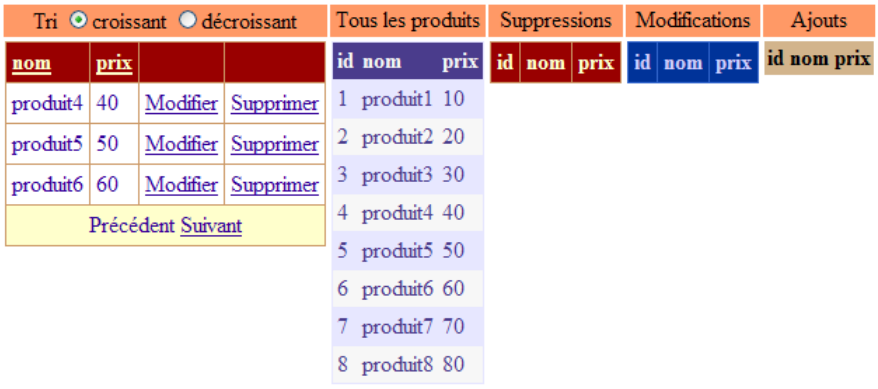
Dans ce cas, la vue [formulaire] est réaffichée. Si les deux informations saisies sont correctes, alors deux données sont placées dans la session :

- le nombre de produits par page choisi par l'utilisateur
- le n° de la page qui va être affichée - au départ la page 0

Ces deux informations sont utilisées à chaque fois que la vue [produits].

3.6.11 Affichage de la vues [produits]

La vue [produits] est la suivante :



Nous avons 5 composants [DataGrid] exposant chacun une vue particulière de la table des produits [dtProduits]. En partant de la gauche :

nom	rôle
DataGrid1	grille d'affichage d'une vue filtrée de la table des produits. Le filtre est celui fixé par la vue [formulaire]. On a également .AllowPaging=true, .AllowSorting=true
DataGrid2	affiche la totalité de la table des produits - permet de suivre les mises à jour
DataGrid3	affichera les produits supprimés dans la table des produits
DataGrid4	affichera les produits modifiés dans la table des produits
DataGrid5	affichera les produits ajoutés dans la table des produits

La procédure [afficheProduits] est chargée d'afficher la vue précédente :

```
Private Sub afficheProduits(ByVal page As Integer, ByVal taillePage As Integer)
  ' on attache les données aux deux composants [DataGrid]
  Application.Lock()
  With DataGrid1
    .DataSource = dvProduits
    .PageSize = taillePage
    .CurrentPageIndex = page
    .DataBind()
  End With
  Dim dvCurrent As New DataView(dtProduits)
  dvCurrent.RowStateFilter = DataViewRowState.CurrentRows
  With DataGrid2
    .DataSource = dvCurrent
    .DataBind()
  End With
  Dim dvSupp As DataView = New DataView(dtProduits)
  dvSupp.RowStateFilter = DataViewRowState.Deleted
  With DataGrid3
    .DataSource = dvSupp
    .DataBind()
  End With
  Dim dvModif As DataView = New DataView(dtProduits)
  dvModif.RowStateFilter = DataViewRowState.ModifiedOriginal
  With DataGrid4
    .DataSource = dvModif
    .DataBind()
  End With
  Dim dvAjout As DataView = New DataView(dtProduits)
  dvAjout.RowStateFilter = DataViewRowState.Added
  With DataGrid5
    .DataSource = dvAjout
    .DataBind()
  End With
  Application.UnLock()
```

```
    ' on affiche la vue [produits]
    vueProduits.Visible = True
    vueFormulaire.Visible = False
    vueErreurs.Visible = False
    vueAjout.Visible = False
    ' on mémorise la page courante
    Session("pageCourante") = page
End Sub
```

La procédure admet deux paramètres :

- le n° de page [page] à afficher dans [DataGrid1]
- le nombre de produits [taillePage] par page

La liaison des [DataGrid] à la table des produits se fait au travers de 5 vues différentes.

- [DataGrid1] est liée à la vue paginée et triée [dvProduits].

```
With DataGrid1
  .DataSource = dvProduits
  .PageSize = taillePage
  .CurrentPageIndex = page
  .DataBind()
End With
```

C'est lors de cette liaison de [DataGrid1] à sa source de données qu'on a besoin des deux informations passées en paramètres à la procédure.

- [DataGrid2] est lié à une vue affichant tous les éléments actuels de la table [dtProduits]. Elle présente comme [DataGrid1] une vue actualisée de la table [dtProduits] mais sans pagination ni tri.

```
Dim dvCurrent As New DataView(dtProduits)
dvCurrent.RowStateFilter = DataViewRowState.CurrentRows
With DataGrid2
  .DataSource = dvCurrent
  .DataBind()
End With
```

Nous utilisons ici l'attribut [RowStateFilter] de la classe [DataView]. Une ligne d'une table ou ici d'une vue possède un attribut [RowState] indiquant l'état de la ligne. En voici quelques-uns :

- [Added] : la ligne a été ajoutée
- [Modified] : la ligne a été modifiée
- [Deleted] : la ligne a été détruite
- [Unchanged] : la ligne n'a pas changé

Lorsque la table [dtProduits] a été construite initialement dans [Application_Start], toutes ses lignes étaient dans l'état [Unchanged]. Cet état va évoluer au fil des mises à jour des clients web :

- lorsqu'un client crée un nouveau produit, une ligne est ajoutée à la table [dtProduits]. Elle sera dans l'état [Added].
- lorsqu'un produit est modifié, son état passe de [Unchanged] à [Modified].
- lorsqu'un produit est supprimé, la ligne n'est pas supprimée physiquement. Elle est plutôt marquée comme étant à supprimer et son état passe à [Deleted]. Il est possible d'annuler cette suppression.

L'attribut [RowStateFilter] de la classe [DataView] permet de filtrer une vue selon l'état [RowState] de ses lignes. Ses valeurs possibles sont celles de l'énumération [DataRowViewState] :

- DataRowViewState.CurrentRows : les lignes non supprimées sont affichées avec leurs valeurs actuelles
- DataRowViewState.Added : les lignes ajoutées sont affichées avec leurs valeurs actuelles
- DataRowViewState.Deleted : les lignes supprimées sont affichées avec leurs valeurs d'origine
- DataRowViewState.ModifiedOriginal : les lignes modifiées sont affichées avec leurs valeurs d'origine
- DataRowViewState.ModifiedCurrent : les lignes modifiées sont affichées avec leurs valeurs actuelles

Une ligne modifiée a deux valeurs : la valeur d'origine qui est celle que la ligne avait avant la première modification et la valeur actuelle celle obtenue après une ou plusieurs modifications. Ces deux valeurs sont conservées simultanément.

Les DataGrid 2 à 5 affichent une vue de la table [dtProduits] filtrée par l'attribut [RowStateFilter]

DataGrid	Filtre
DataGrid2	RowStateFilter= DataRowViewState.CurrentRows
DataGrid3	RowStateFilter= DataRowViewState.Deleted
DataGrid4	RowStateFilter= DataRowViewState.ModifiedOriginal

DataGrid	Filtre
DataGrid5	RowStateFilter= DataRowViewState.Added

La table [dtPoduits] est mise à jour simultanément par différents clients web, ce qui peut amener à des conflits d'accès à la table. Lorsqu'un client affiche ses vues de la table [dtProduits], on veut éviter qu'il le fasse alors que la table est dans un état instable parce qu'en cours de modification par un autre client web. Aussi à chaque fois qu'un client a besoin de la table [dtProduits] en lecture comme ci-dessus ou en écriture lorsqu'il va ajouter, modifier et supprimer des produits, il se synchronisera avec les autres clients au moyen de la séquence suivante :

```
Application.Lock
... section critique 1
Application.Unlock
```

Il peut exister plusieurs sections critiques dans le code de l'application :

```
Application.Lock
... section critique 2
Application.Unlock
```

La mécanique est la suivante :

1. un ou plusieurs clients arrivent à l'instruction [Application.Lock] de la section critique 1. C'est un distributeur de jeton d'entrée unique. Un unique client btient ce jeton. Appelons-le C1.
2. tant que le client C1 n'a pas rendu le jeton par [Application.Unlock], aucun autre client n'est autorisé à entrer dans une section critique contrôlée par [Application.Lock]. Dans l'exemple ci-dessus, aucun client ne peut donc entrer dans les sections critiques 1 et 2.
3. le client C1 exécute [Application.Unlock] et rend donc le jeton d'entrée. Celui-ci peut alors être donné à un autre client. Les étapes 1 à 3 sont rejouées.

Avec ce mécanisme, nous assurerons qu'un seul client a accès à la table [dtProduits] que ce soit en lecture ou écriture.

Le lien [Mise à jour] affiche également la vue [Produits] :

La procédure associée est la suivante :

```
 Private Sub lnkMisajour_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkMisajour.Click
   ' affichage vue produits
   afficheProduits(CType(Session("pageCourante"), Integer), CType(Session("nbProduitsPage"), Integer))
 End Sub
```

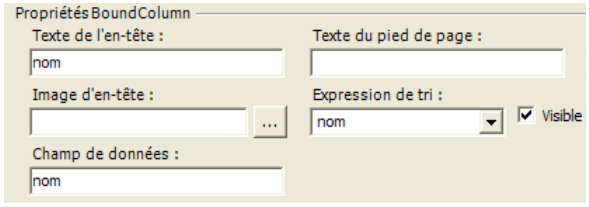
On doit afficher la vue [produits]. Ceci est fait à l'aide de la procédure [afficheProduits] qui admet deux paramètres : le n° de la page courante à afficher et le nombre de produits par page à afficher. Ces deux informations sont dans la session, éventuellement avec leurs valeurs initiales si l'utilisateur ne les a jamais définies lui-même.

3.6.12 Pagination et tri de [DataGrid1]

Nous avons déjà rencontré les procédures gérant la pagination et le tri d'un [DataGrid] dans un autre exemple. Dans le cas d'un changement de page, on affiche la vue [produits] en passant le n° de la nouvelle page en paramètre à la procédure [afficheProduits].

```
 Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridPageChangedEventArgs) Handles DataGrid1.PageIndexChanged
   ' chgt de page
   afficheProduits(e.NewPageIndex, DataGrid1.PageSize)
 End Sub
```

La procédure [DataGrid1_SortCommand] est exécutée lorsque l'utilisateur clique sur l'entête de l'une des colonnes de [DataGrid1]. Il faut alors affecter à l'attribut [Sort] de la vue [dvProduits] affichée par [DataGrid1] l'expression de tri. Celle-ci est syntaxiquement équivalente à l'expression de tri placée derrière la clause [order by] de l'instruction SQL SELECT. L'argument [e] de la procédure a un attribut [SortExpression] qui nous donne l'expression de tri associée à la colonne dont l'entête a été cliqué. Lorsque le composannt [DataGrid1] a été construit, cette expression de tri avait été définie. Ci-dessous, celle définie pour la colonne [nom] de [DataGrid1] :



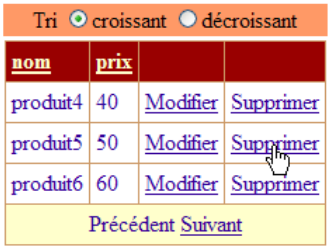
Si l'expression de tri n'a pas été définie lors de la conception du [DataGrid], c'est le nom du champ de données associé à la colonne du [DataGrid] qui est utilisée. Le sens du tri (croissant, décroissant) est ici fixé par l'utilisateur à l'aide des boutons radio [rdCroissant, rd Décroissant] :

La procédure de tri est la suivante :

```
 Private Sub DataGrid1_SortCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridSortCommandEventArgs) Handles DataGrid1.SortCommand
   ' on trie le dataview
   With dvProduits
     .Sort = e.SortExpression + " " + CType(IIf(rdCroissant.Checked, "asc", "desc"), String)
   End With
   ' on l'affiche
   afficheProduits(0, DataGrid1.PageSize)
 End Sub
```

3.6.13 Suppression d'un produit

Pour supprimer un produit, l'utilisateur utilise le lien [Supprimer] de la ligne du produit :



La procédure [DataGrid1_DeleteCommand] est alors exécutée :

```
 Private Sub DataGrid1_DeleteCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.DeleteCommand
   ' suppression d'un produit
   ' clé du produit à supprimer
   Dim idProduit As Integer = CType(DataGrid1.DataKeys(e.Item.ItemIndex), Integer)
   ' suppression du produit
   Dim erreur As Boolean = False
   Try
     supprimerProduit(idProduit)
   Catch ex As Exception
     ' pb
     lblInfo2.Text = ex.Message
     erreur = True
   End Try
   ' chgt de page ?
   Dim page As Integer = DataGrid1.CurrentPageIndex
   If Not erreur AndAlso DataGrid1.Items.Count = 1 Then
     page = DataGrid1.CurrentPageIndex - 1
     If page < 0 Then page = 0
   End If
   ' affichage produits
   afficheProduits(page, DataGrid1.PageSize)
 End Sub
```

La mise à jour des produits se fera via la clé [id] du produit. En effet la colonne [id] de la table [dtProduits] est clé primaire et grâce à elle, nous pouvons retrouver dans la table [dtProduits] la ligne du produit qui est mis à jour dans le composant [DataGrid1]. La procédure commence donc par récupérer la clé du produit dont le lien [Supprimer] a été cliqué :

```
    ' clé du produit à supprimer
    Dim idProduit As Integer = CType(DataGrid1.DataKeys(e.Item.ItemIndex), Integer)
```

Nous savons que [e.Item] est l'élément de [DataGrid1] à l'origine de l'événement. Grosso-modo cet élément est une ligne du [DataGrid]. [e.Item.ItemIndex] est le n° de la ligne à l'origine de l'événement. Cet index est relatif à la page courante affichée. Ainsi la première ligne de la page affichée à la propriété [ItemIndex=0] même si elle a le numéro 17 dans la table des produits. [DataGrid1.DataKeys] est la liste des clés du [DataGrid]. Parce que nous avons, à la conception, écrit [DataGrid1.DataKey=id], les clés de [DataGrid1] sont constituées des valeurs de la colonne [id] de la table [dtProduits] qui est en même temps clé primaire. Ainsi [DataGrid1.DataKeys(e.Item.ItemIndex)] est la clé [id] du produit à supprimer. Ceci obtenu, on demande sa suppression à l'aide de la procédure [supprimerProduit] :

```
    ' suppression du produit
    Dim erreur As Boolean = False
    Try
      supprimerProduit(idProduit)
    Catch ex As Exception
      ' pb
      lblInfo2.Text = ex.Message
      erreur = True
    End Try
```

Cette suppression peut échouer dans certains cas. Nous verrons pourquoi. Une exception est alors lancée. Elle est ici gérée. S'il y a erreur, le [DataGrid1] n'a pas à changer. Seul un message d'erreur est ajouté à la vue [produits]. S'il n'y a pas d'erreur et si l'utilisateur vient de supprimer l'unique produit de la page courante alors on affiche la page précédente.

```
    ' chgt de page ?
    Dim page As Integer = DataGrid1.CurrentPageIndex
    If Not erreur AndAlso DataGrid1.Items.Count = 1 Then
      page = DataGrid1.CurrentPageIndex - 1
      If page < 0 Then page = 0
    End If
```

Dans tous les cas, la vue [produits] est réaffichée avec la procédure [afficheProduits] :

```
    ' affichage produits
    afficheProduits(page, DataGrid1.PageSize)
```

La procédure [supprimerProduit] est la suivante :

```
  Private Sub supprimerProduit(ByVal idProduit As Integer)
    Dim erreur As String
    Try
      ' synchronisation
      Application.Lock()
      ' recherche de la ligne à supprimer
      Dim ligne As DataRow = dtProduits.Rows.Find(idProduit)
      If ligne Is Nothing Then
        erreur = String.Format("Produit [{0}] inexistant", idProduit)
      Else
        ' suppression de la ligne
        ligne.Delete()
      End If
    Catch ex As Exception
      erreur = String.Format("Erreur de suppression : {0}", ex.Message)
    Finally
      ' fin de synchronisation
      Application.UnLock()
    End Try
    ' on lance une exception si erreur
    If erreur <> String.Empty Then Throw New Exception(erreur)
  End Sub
```

La procédure reçoit en paramètre la clé du produit à supprimer. S'il n'y avait qu'un client à faire les mises à jour, cette suppression pourrait se faire comme suit :

```
      ' suppression de la ligne [idProduit]
      dtProduits.Rows.Find(idProduit).Delete()
```

Avec plusieurs clients faisant les mises à jour en même temps, c'est plus compliqué. En effet, considérons la séquence dans le temps suivante :

Temps	Action
T1	le client A lit la table [dtProduits] - il y a un produit de clé 20
T2	le client B lit la table [dtProduits] - il y a un produit de clé 20
T3	le client A supprime le produit de clé 20
T4	le client B supprime le produit de clé 20

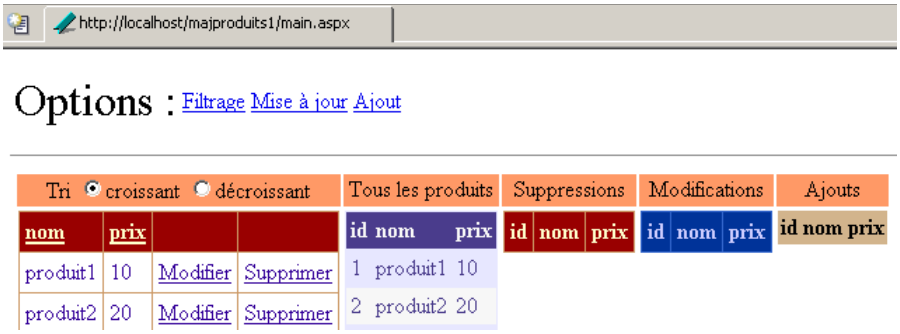
Lorsque les clients A et B lisent la table des produits et qu'ils en affichent le contenu dans une page web, la table contient le produit de clé 20. Ils peuvent donc vouloir faire une action sur ce produit, par exemple le détruire. Il y en forcément un des deux qui le fera le premier. Celui qui passera après voudra alors détruire un produit qui n'existe plus. Ce cas est pris en compte par la procédure [supprimerProduits] de plusieurs façons :

- il y a tout d'abord synchronisation avec [Application.Lock]. Cela signifie que lorsqu'un client passe cette barrière, il n'y en a pas d'autre à modifier ou lire la table des produits. En effet ces opérations sont toutes synchronisées de cette façon. Nous l'avons déjà vu pour la lecture.
- on vérifie ensuite si le produit qu'on veut supprimer existe. Si oui, il est supprimé, sinon un message d'erreur est préparé.

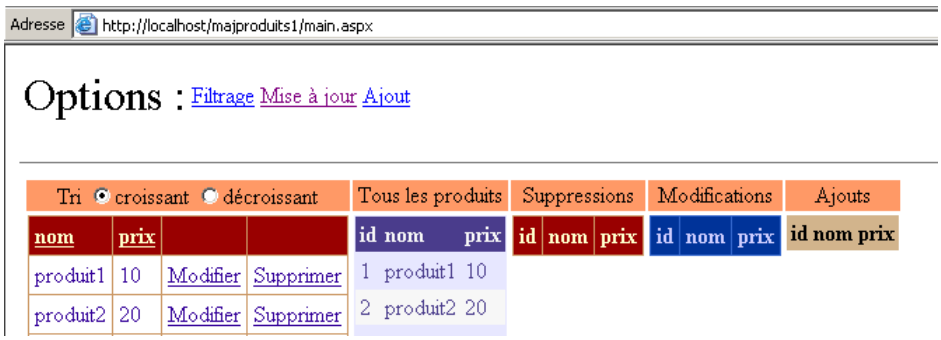
```
' recherche de la ligne à supprimer
Dim ligne As DataRow = dtProduits.Rows.Find(idProduit)
If ligne Is Nothing Then
  erreur = String.Format("Produit [{0}] inexistant", idProduit)
Else
  ' suppression de la ligne
  ligne.Delete()
End If
```

- on sort de la séquence critique par [Application.Unlock] afin de permettre aux autres clients de faire leurs mises à jour.
- si la ligne n'a pas pu être supprimée, la procédure lance une exception associée à un message d'erreur.

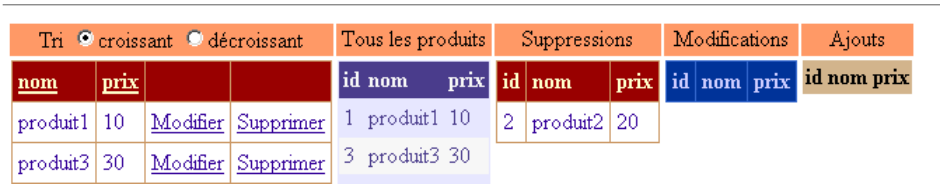
Voyons un exemple. Nous lançons un client [Mozilla] et nous prenons immédiatement l'option [Mise à jour] (vue partielle) :



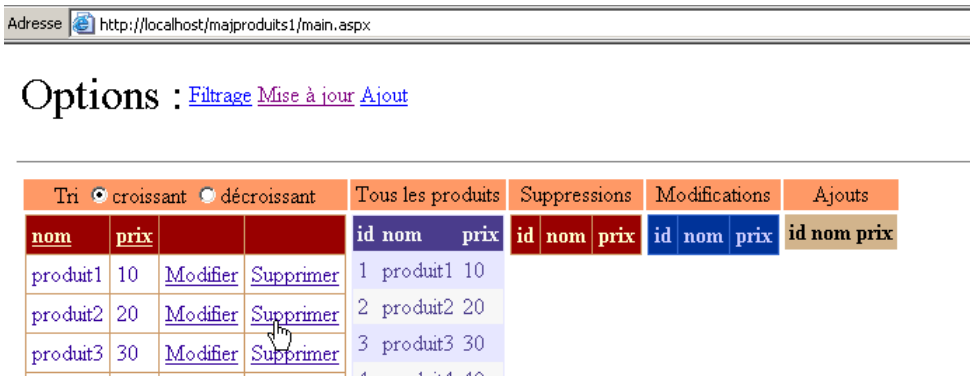
Nous faisons de même avec un client [Internet Explorer] (vue partielle) :



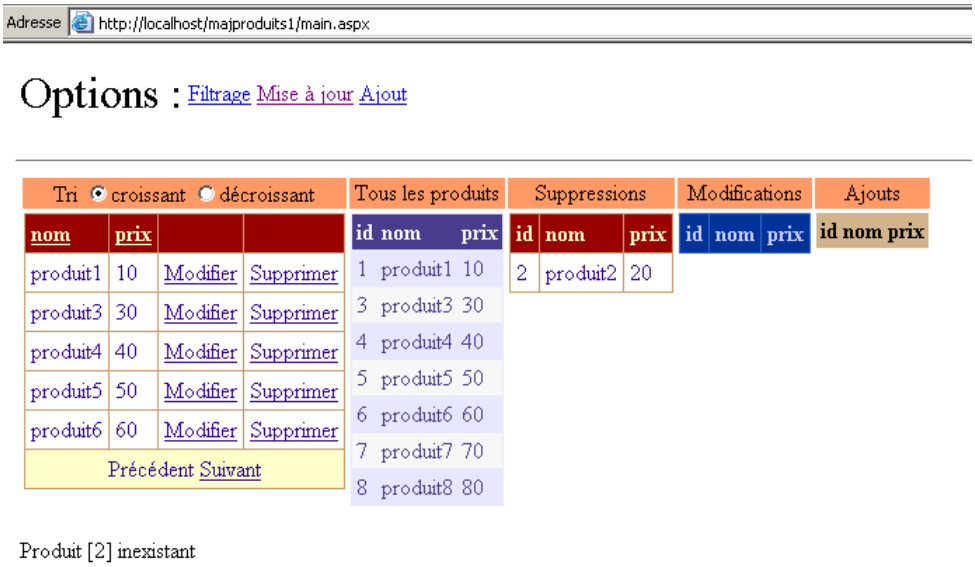
Avec le client [Mozilla] nous supprimons le produit [produit2]. Nous obtenons la nouvelle page suivante :



L'opération de suppression s'est bien déroulée. Le produit supprimé apparaît bien le [DataGrid] des produits supprimés et n'apprait plus dans la liste des produits des composants [DataGrid1] et [DataGrid2] qui exposent les produits actuellement présents dans la table. Faisons de même avec [Interbet Explorer]. On supprime l'élément [produit2] :



La réponse obtenue est la suivante :



Un message d'erreur signale à l'utilisateur que le produit de clé [2] n'existe pas. La réponse renvoie au client une nouvelle vue relétant l'état actuel de la table. On voit qu'effectivement le produit de clé [2] est dans la liste des produits supprimés. La vue [produits] reflète donc les mises à jour de tous les clients et non d'un seul.

3.6.14 Ajout d'un produit

Pour ajouter un produit, l'utilisateur utilise le lien [Ajout] des options. Celui-ci se contente d'afficher la vue [Ajout] :

Les deux procédures impliquées dans cette action sont les suivantes :

```
 Private Sub lnkAjout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkAjout.Click
   ' on affiche la vue [ajout]
   afficheAjout()
 End Sub

 Private Sub afficheAjout()
   ' affiche la vue ajout
   vueAjout.Visible = True
   vueFormulaire.Visible = False
   vueProduits.Visible = False
   vueErreurs.Visible = False
 End Sub
```

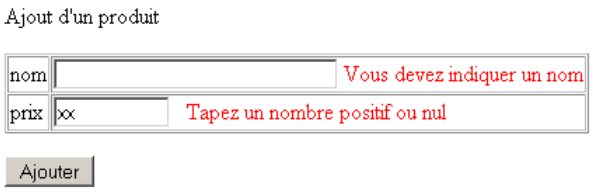
Rappelons que la vue [Ajout] a des composants de validation :

nom	type	rôle
rfvNom	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtNom]
rfvPrix	RequiredFieldValidator	vérifie la présence d'une valeur dans [txtPrix]
cvPrix	CompareValidator	vérifie que prix>=0

La procédure de gestion du clic sur le bouton [Ajouter] est la suivante :

```
 Private Sub btnAjouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAjouter.Click
   ' ajout d'un nouvel élément à la table des produits
   ' il faut tout d'abord que les données soient valides
   rfvNom.Validate()
   rfvPrix.Validate()
   cvPrix.Validate()
   If Not rfvNom.IsValid Or Not rfvPrix.IsValid Or Not cvPrix.IsValid Then
     ' de nouveau le formulaire de saisie
     afficheAjout()
     Exit Sub
   End If
   ' on crée une ligne
   Dim produit As DataRow = dtProduits.NewRow
   produit("nom") = txtNom.Text.Trim
   produit("prix") = txtPrix.Text.Trim
   ' on ajoute la ligne à la table
   Application.Lock()
   Try
     dtProduits.Rows.Add(produit)
     lblInfo3.Text = "Ajout réussi"
     ' nettoyage
     txtNom.Text = ""
     txtPrix.Text = ""
   Catch ex As Exception
     lblInfo3.Text = String.Format("Erreur : {0}", ex.Message)
   End Try
   Application.UnLock()
 End Sub
```

Tout d'abord, on exécute les contrôles de validité des trois composants [rfvNom, rfvPrix, cvPrix]. Si l'un des contrôles échoue, la vue [Ajout] est réaffichée avec les messages d'erreurs des composants de validation. Voici un exemple :



Si les données sont valides, on prépare la ligne à insérer dans la table [dtProduits].

```
' on crée une ligne
Dim produit As DataRow = dtProduits.NewRow
produit("nom") = txtNom.Text.Trim
produit("prix") = txtPrix.Text.Trim
```

On crée tout d'abord une nouvelle ligne de la table [dtProduits] avec la méthode [DataTable.NewRow]. Cette ligne aura les trois colonnes [id, nom, prix] de la table [dtProduits]. Les colonnes [nom, prix] sont renseignées avec les valeurs saisies dans le formulaire d'ajout. La colonne [id] elle n'est pas renseignée. Cette colonne est de type [AutoIncrement], c.a.d. que le SGBD donnera comme clé à un nouveau produit, la clé maximale existante augmentée de 1. La ligne [produit] créée ici est détachée de la table [dtProduits]. Il nous faut maintenant l'insérer dans cette table. Parce qu'on va mettre à jour la table [dtProduits] on active la synchronisation inter-clients :

```
Application.Lock()
```

Ceci fait, on essaie d'ajouter la ligne à la table [dtProduits]. Si on réussit, on fait afficher un message de réussite sinon un message d'erreur.

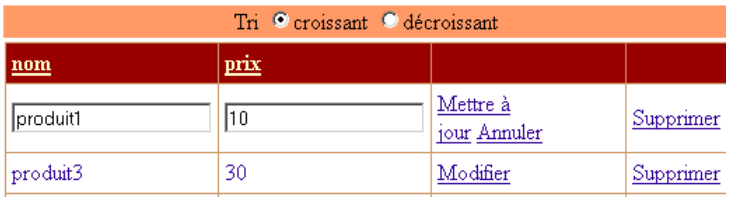
```
Try
  dtProduits.Rows.Add(produit)
  lblInfo3.Text = "Ajout réussi"
  ' nettoyage
  txtNom.Text = ""
  txtPrix.Text = ""
Catch ex As Exception
  lblInfo3.Text = String.Format("Erreur : {0}", ex.Message)
End Try
```

Il n'y a normalement aucune exception possible. C'est par précaution qu'on a mis néanmoins en place une gestion d'exception. L'ajout fait, on quitte la section critique par [Application.Unlock].

3.6.15 Modification d'un produit

Pour modifier un produit, l'utilisateur utilise le lien [Modifier] de la ligne du produit :

On passe alors en mode édition :



Grâce au composant [DataGrid] ce résultat est obtenu avec très peu de code :

```
 Private Sub DataGrid1_EditCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.EditCommand
   ' on met l'élément courant en mode édition
   DataGrid1.EditItemIndex = e.Item.ItemIndex
   ' on réaffiche les produits
   afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
```

```
End Sub
```

Le clic sur le lien [Modifier] provoque l'exécution, côté serveur, de la procédure [DataGrid1_EditCommand]. Le composant [DataGrid1] a un champ [EditItemIndex]. La ligne ayant pour index la valeur de [EditItemIndex] est mise en mode édition. Chaque valeur de la ligne ainsi mise à jour peut être modifiée dans une boîte de saisie comme le montre la copie d'écran ci-dessus. Nous récupérons donc l'index du produit sur lequel a eu lieu le clic sur le lien [Modifier] et l'affectons à l'attribut [EditItemIndex] du composant [DataGrid1]. Il ne nous reste plus qu'à réafficher la vue [produits]. Elle appraîtra identique à ce qu'elle était mais avec une ligne en mode édition.

Le lien [Annuler] du produit en cours d'édition permet à l'utilisateur d'abandonner sa mise à jour. La procédure liée à ce lien est la suivante :

```
Private Sub DataGrid1_CancelCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.CancelCommand
  ' on réaffiche les produits
  afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
End Sub
```

Elle se contente de réafficher la vue [produits]. On pourrait être tenté de mettre -1 dans [DataGrid1.EditItemIndex] pour annuler me mode de mise à jour. En fait, nous savons que la procédure [Page_Load] le fait systématiquement. Il n'y donc pas lieu de le refaire.
La modification est validée par le lien [Mettre à jour] de la ligne en cours d'édition. La procédure suivante est alors exécutée :

```
Private Sub DataGrid1_UpdateCommand(ByVal source As Object, ByVal e As
System.Web.UI.WebControls.DataGridCommandEventArgs) Handles DataGrid1.UpdateCommand
  ' clé du produit à modifier
  Dim idProduit As Integer = CType(DataGrid1.DataKeys(e.Item.ItemIndex), Integer)
  ' éléments modifiés
  Dim nom As String = CType(e.Item.Cells(0).Controls(0), TextBox).Text.Trim
  Dim prix As String = CType(e.Item.Cells(1).Controls(0), TextBox).Text.Trim
  ' modifications valides ?
  lblInfo2.Text = ""
  If nom = String.Empty Then lblInfo2.Text += "[Indiquez un nom]"
  If prix = String.Empty Then
    lblInfo2.Text += "[Indiquez un prix]"
  Else
    Dim nouveauPrix As Double
    Try
      nouveauPrix = CType(prix, Double)
      If nouveauPrix < 0 Then Throw New Exception
    Catch ex As Exception
      lblInfo2.Text += "[prix invalide]"
    End Try
  End If
  '  si erreur, on réaffiche la page de mise à jour
  If lblInfo2.Text <> String.Empty Then
    ' on remet la ligne en mode mise à jour
    DataGrid1.EditItemIndex = e.Item.ItemIndex
    ' affichage produits
    afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
    ' fin
    Exit Sub
  End If
  ' cas où pas d'erreur - on modifie la table
  Try
    modifierProduit(idProduit, e.Item)
  Catch ex As Exception
    ' pb
    lblInfo2.Text = ex.Message
  End Try
  ' affichage produits
  afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
End Sub
```

Comme pour la suppression, il nous faut récupérer la clé du produit à modifier afin de retrouver sa ligne dans la table des produits [dtProduits] :

```
  ' clé du produit à modifier
  Dim idProduit As Integer = CType(DataGrid1.DataKeys(e.Item.ItemIndex), Integer)
```

Ensuite il nous faut récupérer les nouvelles valeurs à affecter à la ligne. Celles-ci sont dans le composant [DataGrid1]. L'argument [e] de la procédure est là pour nous aider. [e.Item] reprsésente la ligne de [DataGrid1] source de l'événement. C'est donc la ligne en cours de mise à jour puisque le lien [Mettre à jour] n'existe que sur cette ligne. Cette ligne contient des colonnes désignées par la collection [Cells] de la ligne. Ainsi [e.Item.Cells(0)] reprséente la colonne 0 de la ligne mise à jour. Nous savons que les nouvelles

valeurs sont dans des boîtes de saisie. La collection [e.Item.Cells(i).Controls] représente la collection des contrôles de la colonne i de la ligne [e.Item]. Les deux instructions suivantes récupèrent les valeurs des boîtes de saisie de la ligne mise à jour dans [DataGrid1] :

```
' éléments modifiés
Dim nom As String = CType(e.Item.Cells(0).Controls(0), TextBox).Text.Trim
Dim prix As String = CType(e.Item.Cells(1).Controls(0), TextBox).Text.Trim
```

Nous avons donc les nouvelles valeurs de la ligne modifiée sous la forme de chaînes de caractères. Nous vérifions ensuite si ces données sont valides. Le nom doit être non vide et le prix doit être un nombre positif ou nul :

```
' modifications valides ?
lblInfo2.Text = ""
If nom = String.Empty Then lblInfo2.Text += "[Indiquez un nom]"
If prix = String.Empty Then
  lblInfo2.Text += "[Indiquez un prix]"
Else
  Dim nouveauPrix As Double
  Try
    nouveauPrix = CType(prix, Double)
    If nouveauPrix < 0 Then Throw New Exception
  Catch ex As Exception
    lblInfo2.Text += "[prix invalide]"
  End Try
End If
```

En cas d'erreur, le label [lblInfo2] affichera un message d'erreur et on se contente de réafficher la même page :

```
'  si erreur, on réaffiche la page de mise à jour
If lblInfo2.Text <> String.Empty Then
  ' on remet la ligne en mode mise à jour
  DataGrid1.EditItemIndex = e.Item.ItemIndex
  ' affichage produits
  afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
  ' fin
  Exit Sub
End If
```

Ce que ne montre pas le code ci-dessus, c'est que les valeurs saisies sont perdues. En effet, [DataGrid1] est lié aux données de la table [dtProduits] qui contient les valeurs d'origine de la ligne modifiée. Voici un exemple.

nom	prix	
produit1xx	10xx	Mettre à jour
produit2	20	Modifier

La réponse obtenue est la suivante :

nom	prix	
produit1	10	Mettre à jour Annuler
produit2	20	Modifier
produit3	30	Modifier
produit4	40	Modifier
produit5	50	Modifier
Précédent Suivant		

[prix invalide]

On voit qu'on a perdu les valeurs initialement saisies. Dans une application professionnelle, ce ne serait probablement pas acceptable. On atteint ici certaines limites du mode standard de mise à jour du composant [DataGrid]. Il serait préférable d'avoir une vue [Modification] analogue à la vue [Ajout].

Si les données sont valides, elles sont utilisées pour faire la mise à jour de la table [dtProduits] :

```
' cas où pas d'erreur - on modifie la table
Try
```

```
    modifierProduit(idProduit, e.Item)
  Catch ex As Exception
    ' pb
    lblInfo2.Text = ex.Message
  End Try
  ' affichage produits
  afficheProduits(DataGrid1.CurrentPageIndex, DataGrid1.PageSize)
```

Pour la même raison évoquée lors de la suppression d'un produit, la modification peut échouer. En effet, entre le moment où le client a lu la table [dtProduits] et celui où il va modifier l'un de ses produits, celui-ci a pu être supprimé par un autre client. La procédure [modifierProduit] gère ce cas en lançant une exception. Celle-ci est ici gérée. Après réussite ou échec de la mise à jour, l'application renvoie la vue [produits] au client. Il nous reste à voir comment la procédure [modifierProduit] réalise la mise à jour :

```
Private Sub modifierProduit(ByVal idProduit As Integer, ByVal item As DataGridItem)
  Dim erreur As String
  Try
    ' synchronisation
    Application.Lock()
    ' recherche de la ligne à modifier
    Dim ligne As DataRow = dtProduits.Rows.Find(idProduit)
    If ligne Is Nothing Then
      erreur = String.Format("Produit [{0}] inexistant", idProduit)
    Else
      ' on modifie la ligne
      With ligne
        .Item("nom") = CType(item.Cells(0).Controls(0), TextBox).Text.Trim
        .Item("prix") = CType(item.Cells(1).Controls(0), TextBox).Text.Trim
      End With
    End If
  Catch ex As Exception
    erreur = String.Format("Erreur lors de la modification : {0}", ex.Message)
  Finally
    ' fin de synchronisation
    Application.UnLock()
  End Try
  ' on lance une exception si erreur
  If erreur <> String.Empty Then Throw New Exception(erreur)
End Sub
```

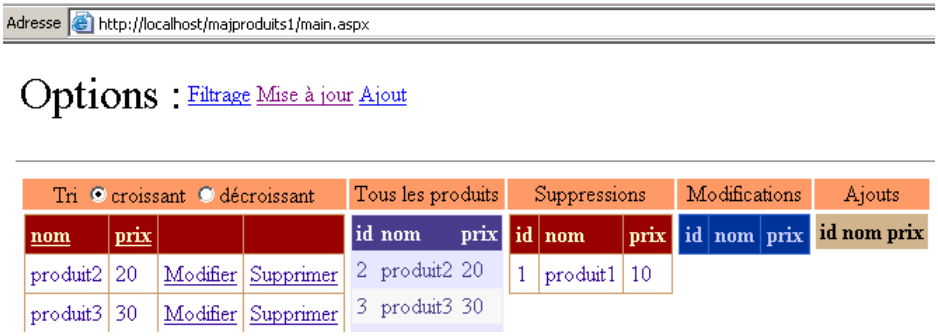
Nous ne rentrerons pas dans les détails de cette procédure dont le code est analogue à celui de la procédure [supprimerProduit] qui a été expliqué longuement. Voyons simplement deux exemples. Tout d'abord nous modifions le prix du produit [produit1] :

L'utilisation du lien [Mettre à jour] ci-dessus amène la réponse suivante :

La modification est bien présente dans les composants [DataGrid] 1 et 2 qui reflètent l'état actuel de la table [dtProduits]. Elle est également présente dans le composant [DataGrid4] qui est une vue sur les lignes modifiées, celles-ci étant affichées avec leurs valeurs d'origine. Maintenant voyons un cas de conflit d'accès. Comme pour l'exemple de suppression, nous allons utiliser deux clients web différents. Un client [Mozilla] lit la table [dtProduits] et se met en modification du produit [produit1] :

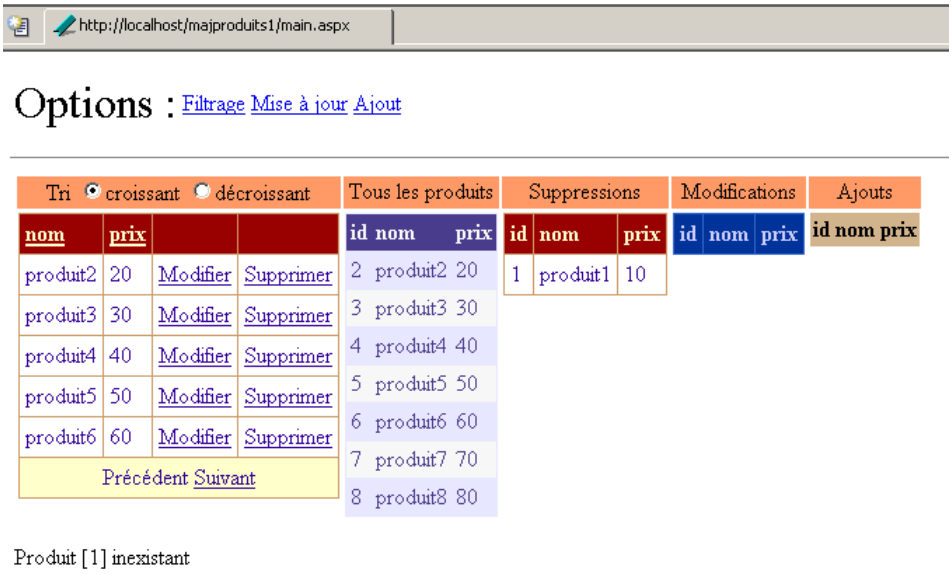
Un client [Internet Explorer] s'apprête lui à supprimer le produit [produit1] :

Le client [Internet Explorer] supprime [produit1] :



On remarquera que [produit1] n'est plus dans le tableau des lignes modifiées mais dans celui des lignes supprimées. Le client [Mozilla] lui valide sa mise à jour :

Le client [Mozilla] reçoit la réponse suivante à sa mise à jour :



Il peut constater que [produit1] a été supprimé puisqu'il est dans la liste des produits supprimés.

3.7 Application web de mise à jour de la table physique des produits

3.7.1 Solutions proposées

L'application précédente était davantage un cas d'école destiné à montrer la gestion d'un objet [DataTable] en cache qu'un cas courant. Il faut bien en effet, à un moment ou à un autre, mettre à jour la source réelle des données. On peut opter pour deux stratégies différentes :

1. on utilise le cache [dtProduits] en mémoire pour mettre à jour la source des données. On peut prévoir une page située dans l'arborescence web de l'application précédente afin d'avoir accès au cache [dtProduits] de celle-ci. Cette page offrirait à un administrateur la possibilité de répercuter sur la source physique des données les modification en cache dans [dtProduits]. On pourrait pour ce faire, ajouter une nouvelle méthode à la classe d'accès [produits] qui admettrait pour paramètre le cache [dtProduits] et qui avec ce cache mettrait à jour la source physique des données.
2. on met à jour la source de données physique en même temps que le cache.

La stratégie n° 1 permet de n'ouvrir qu'une connexion avec la source de données physique. La stratégie n° 2 nécessite une connexion à chaque mise à jour. Selon la disponibilité des connexions on pourra préférer l'une ou l'autre des stratégies. Parce que nous avons les outils pour la mettre en oeuvre (la classe [produits]), nous choisissons la stratégie n° 2.

3.7.2 Mise en oeuvre de la solution 1

Nous choisissons par souci de continuité avec l'application précédente la statégie suivante :

- la source physique est mise à jour en même temps que le cache
- le cache n'est construit qu'une fois à l'initialisation de l'application dans [global.asax.vb]. Cela signifie que si la source de données physique est mise à jour par d'autres clients que les clients web, ceux-ci ne voient pas ces modifications. Ils ne voient que celles qu'eux-mêmes apportent à la table en cache.

Pour faire la mise à jour de la source physique des données, nous avons besoin d'une instance de la classe d'accès aux produits. Chaque client pourrait avoir la sienne. On peut aussi partager une unique instance qui serait créée par l'application à son démarrage. C'est la solution que nous choisissons ici. Le code de contrôle [global.asax.vb] est modifié comme suit :

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

Public Class Global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on récupère les informations de configuration
   Dim chaînedeConnexion As String = ConfigurationSettings.AppSettings("OLEDBStringConnection")
   Dim defaultProduitsPage As String = ConfigurationSettings.AppSettings("defaultProduitsPage")
   Dim erreurs As New ArrayList
...
   ' ici pas d'erreurs de configuration
   ' on crée un objet produits
```

```
    Dim objProduits As New produits(chaînedeConnexion)
    Dim dtProduits As DataTable
    Try
      dtProduits = objProduits.getProduits
    Catch ex As ExceptionProduits
      'il y a eu erreur d'accès aux produits, on le note dans l'application
      Application("erreurs") = ex.erreurs
      Exit Sub
    Catch ex As Exception
      ' erreur non gérée
      erreurs.Add(ex.Message)
      Application("erreurs") = erreurs
      ' exit sub
    End Try
    ' ici pas d'erreurs d'initialisation
...
    ' on mémorise l'instance d'accès aux données
    Application("objProduits") = objProduits
  End Sub

  Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
...
  End Sub
End Class
```

Une instance de la classe d'accès aux données a été mémorisée dans l'application, associée à la clé [objProduits]. Chaque client utilisera cette instance pour accéder à la source physique des données. Elle sera récupérée dans la procédure [Page_Load] de [main.aspx.vb] :

```
  Protected objProduits As produits
....
  Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
...
    ' on récupère l'instance d'accès aux produits
    objProduits = CType(Application("objProduits"), produits)
...
  End Sub
```

L'instance d'accès aux données [objProduits] est accessible à toutes les méthodes de la page. Elle sera utilisée pour les trois opérations de mise à jour : l'ajout, la suppression, la modification.

La procédure d'ajout est modifiée comme suit :

```
  Private Sub btnAjouter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAjouter.Click
...
    ' on ajoute la ligne
    Application.Lock()
    Try
      ' on ajoute la ligne à la table en cache
      dtProduits.Rows.Add(produit)
      ' on ajoute la ligne à la source physique
      Dim nouveauProduit As sProduit
      With nouveauProduit
        .nom = CType(produit("nom"), String)
        .prix = CType(produit("prix"), Double)
      End With
      objProduits.ajouterProduit(nouveauProduit)
      ' suivi
      lblInfo3.Text = "Ajout réussi"
      ' nettoyage
      txtNom.Text = ""
      txtPrix.Text = ""
    Catch ex As Exception
      ' erreur
      lblInfo3.Text = String.Format("Erreur : {0}", ex.Message)
    End Try
    Application.UnLock()
  End Sub
```

La procédure de modification est modifiée comme suit :

```
  Private Sub modifierProduit(ByVal idProduit As Integer, ByVal item As DataGridItem)
    Dim erreur As String
    Try
      ' synchronisation
      Application.Lock()
      ' recherche de la ligne à modifier
```

```
    Dim ligne As DataRow = dtProduits.Rows.Find(idProduit)
    If ligne Is Nothing Then
      erreur = String.Format("Produit [{0}] inexistant", idProduit)
    Else
      ' on modifie la ligne dans le cache
      With ligne
        .Item("nom") = CType(item.Cells(0).Controls(0), TextBox).Text.Trim
        .Item("prix") = CType(item.Cells(1).Controls(0), TextBox).Text.Trim
      End With
      ' on modifie la ligne dans la source physique
      Dim nouveauProduit As sProduit
      With nouveauProduit
        .id = idProduit
        .nom = CType(ligne.Item("nom"), String)
        .prix = CType(ligne.Item("prix"), Double)
      End With
      objProduits.modifierProduit(nouveauProduit)
    End If
  Catch ex As Exception
    erreur = String.Format("Erreur lors de la modification : {0}", ex.Message)
  Finally
    ' fin de synchronisation
    Application.UnLock()
  End Try
  ' on lance une exception si erreur
  If erreur <> String.Empty Then Throw New Exception(erreur)
End Sub
```

La procédure de suppression est modifiée comme suit :

```
Private Sub supprimerProduit(ByVal idProduit As Integer)
  Dim erreur As String
  Try
    ' synchronisation
    Application.Lock()
    ' recherche de la ligne à supprimer
    Dim ligne As DataRow = dtProduits.Rows.Find(idProduit)
    If ligne Is Nothing Then
      erreur = String.Format("Produit [{0}] inexistant", idProduit)
    Else
      ' suppression de la ligne dans le cache
      ligne.Delete()
      ' suppression de la ligne dans la source physique
      objProduits.supprimerProduit(idProduit)
    End If
  Catch ex As Exception
    erreur = String.Format("Erreur de suppression : {0}", ex.Message)
  Finally
    ' fin de synchronisation
    Application.UnLock()
  End Try
  ' on lance une exception si erreur
  If erreur <> String.Empty Then Throw New Exception(erreur)
End Sub
```

3.7.3 Tests

Nous partons de la table de données suivante dans un fichier ACCESS :

liste : Table

id	nom	prix
1	produit1	10
2	produit2	20
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80

Un client web est lancé :

Tri croissant décroissant

nom	prix		
produit1	10	Modifier	Supprimer
produit2	20	Modifier	Supprimer
produit3	30	Modifier	Supprimer
produit4	40	Modifier	Supprimer
produit5	50	Modifier	Supprimer

Précédent Suivant

Tous les produits

id	nom	prix
1	produit1	10
2	produit2	20
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80

Nous ajoutons un produit :

Tri croissant décroissant

nom	prix		
produit1	10	Modifier	Supprimer
produit2	20	Modifier	Supprimer
produit3	30	Modifier	Supprimer
produit4	40	Modifier	Supprimer
produit5	50	Modifier	Supprimer

Précédent Suivant

Tous les produits

id	nom	prix
1	produit1	10
2	produit2	20
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80
9	nouveau produit	1000

Suppressions

id	nom	prix

Modifications

id	nom	prix

Ajouts

id	nom	prix
9	nouveau produit	1000

Nous supprimons [produit1] :

Tri croissant décroissant

nom	prix		
produit2	20	Modifier	Supprimer
produit3	30	Modifier	Supprimer
produit4	40	Modifier	Supprimer
produit5	50	Modifier	Supprimer
produit6	60	Modifier	Supprimer

Précédent Suivant

Tous les produits

id	nom	prix
2	produit2	20
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80
9	nouveau produit	1000

Suppressions

id	nom	prix
1	produit1	10

Modifications

id	nom	prix

Ajouts

id	nom	prix
9	nouveau produit	1000

Nous modifions le prix de [produit2] :

Tri croissant décroissant

nom	prix		
produit2	22	Modifier	Supprimer
produit3	30	Modifier	Supprimer
produit4	40	Modifier	Supprimer
produit5	50	Modifier	Supprimer
produit6	60	Modifier	Supprimer

Précédent Suivant

Tous les produits

id	nom	prix
2	produit2	22
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80
9	nouveau produit	1000

Suppressions

id	nom	prix
1	produit1	10

Modifications

id	nom	prix
2	produit2	20

Ajouts

id	nom	prix
9	nouveau produit	1000

Après ces modifications, nous regardons le contenu de la table [liste] dans la base ACCESS :

liste : Table

	id	nom	prix
	2	produit2	22
	3	produit3	30
	4	produit4	40
	5	produit5	50
	6	produit6	60
	7	produit7	70
	8	produit8	80
	9	nouveau produit	1000

Les trois modification ont été correctement prises en compte dans la table physique. Examinons maintenant un cas de conflit. Nous supprimons, directement sous ACCESS, la ligne de [produit2] :

liste : Table

	id	nom	prix
	3	produit3	30
	4	produit4	40
	5	produit5	50
	6	produit6	60
	7	produit7	70
	8	produit8	80
	9	nouveau produit	1000

Nous revenons à notre client web. Celui-ci ne voit pas la suppression qui a été faite et veut supprimer à son tour [produit2] :

Tri ◉ croissant ○ décroissant

nom	prix		
produit2	22	Modifier	Supprimer
produit3	30	Modifier	Supprimer

Il obtient la réponse suivante :

Tri ◉ croissant ○ décroissant

nom	prix		
produit3	30	Modifier	Supprimer
produit4	40	Modifier	Supprimer
produit5	50	Modifier	Supprimer
produit6	60	Modifier	Supprimer
produit7	70	Modifier	Supprimer

Précédent Suivant

Tous les produits

id	nom	prix
3	produit3	30
4	produit4	40
5	produit5	50
6	produit6	60
7	produit7	70
8	produit8	80
9	nouveau produit	1000

Suppressions

id	nom	prix
1	produit1	10
2	produit2	20

Modifications

id	nom	prix

Ajouts

id	nom	prix
9	nouveau produit	1000

Erreur lors de la mise à jour de la source de données physique : Erreur lors de la suppression : Le produit de clé [2] n'existe pas dans la table des données

La ligne [produit2] a bien été enlevée du cache comme l'indique la liste des produits supprimés. La suppression de [produit2] dans la source physique a elle échoué comme l'indique le message d'erreur.

3.7.4 Mise en oeuvre de la solution 2

Dans la solution précédente, les clients web mettent simultanément à jour la source de données physique mais il ne voit pas les modifications apportées par les autres clients. Ils ne voient que les leurs. Nous voudrions maintenant qu'un client puisse voir la source de données physique telle qu'elle est actuellement et non pas telle qu'elle était au lancement de l'application. Pour cela, on va offrir au client une nouvelle option :

Options : Filtrage Mise à jour Ajout Rafraîchir

Avec l'option [Rafraîchir], le client force une relecture de la source de données physique. Pour que ceci n'affecte pas les autres clients, il faut que la table issue de cette lecture appartienne au client qui fait le rafraîchissement et ne soit pas partagée avec les autres clients. C'est la première différence avec l'application précédente. Le cache [dtProduits] de la source de données sera construit par chaque client et non pas par l'application elle-même. La modification est faite dans [global.asax.vb] :

```
Imports System
Imports System.Web
Imports System.Web.SessionState
Imports st.istia.univangers.fr
Imports System.Configuration
Imports System.Data
Imports Microsoft.VisualBasic
Imports System.Collections

Public Class Global
 Inherits System.Web.HttpApplication

 Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' on récupère les informations de configuration
...
   ' ici pas d'erreurs de configuration
   ' on crée un objet produits
   Dim objProduits As New produits(chaînedeConnexion)
   ' on mémorise le nbre de produits par page
   Application("defaultProduitsPage") = defaultProduitsPage
   ' on mémorise l'instance d'accès aux données
   Application("objProduits") = objProduits
 End Sub

 Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
   ' init des variables de session
   If IsNothing(Application("erreurs")) Then
     ' on met la source de données en cache
     Dim dtProduits As DataTable
     Try
       Application.Lock()
       dtProduits = CType(Application("objProduits"), produits).getProduits
     Catch ex As ExceptionProduits
       'il y a eu erreur d'accès aux produits, on le note dans la session
       Session("erreurs") = ex.erreurs
       Exit Sub
     Finally
       Application.UnLock()
     End Try
     ' on met le cache [dtProduits] dans la session
     Session("dtProduits") = dtProduits
     ' vue sur la table des produits
     Session("dvProduits") = dtProduits.DefaultView
     ' nbre de produits par page
     Session("nbProduitsPage") = Application("defaultProduitsPage")
     ' page courante affichée
     Session("pageCourante") = 0
   End If
 End Sub

End Class
```

Les informations mises en session sont récupérées à chaque requête dans la procédure [Page_Load] :

```
 ' données page
 Protected dtProduits As DataTable
 Protected dvProduits As DataView
 Protected objProduits As produits
 Protected nbProduitsPage As Integer
 Protected pageCourante As Integer

 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
   ' on regarde si l'application est en erreur
   If Not IsNothing(Application("erreurs")) Then
     ' l'application ne s'est pas initialisée correctement
     afficheErreurs(CType(Application("erreurs"), ArrayList))
     Exit Sub
   End If
   ' on regarde si la session est en erreur
   If Not IsNothing(Session("erreurs")) Then
     ' la session ne s'est pas initialisée correctement
     afficheErreurs(CType(Session("erreurs"), ArrayList))
     Exit Sub
   End If
```

```
    ' on récupère une référence sur la table des produits
    dtProduits = CType(Session("dtProduits"), DataTable)
    ' on récupère la vue sur les produits
    dvProduits = CType(Session("dvProduits"), DataView)
    ' on récupère le nombre de produits par page
    nbProduitsPage = CType(Session("nbProduitsPage"), Integer)
    ' on récupère la page courante
    pageCourante = CType(Session("pageCourante"), Integer)
    ' on récupère l'instance d'accès aux produits
    objProduits = CType(Application("objProduits"), produits)
    ' on annule une éventuelle mise à jour en cours
    DataGrid1.EditItemIndex = -1
    '1ère requête
    If Not IsPostBack Then
      ' on affiche le formulaire initial
      txtPages.Text = nbProduitsPage.ToString
      afficheFormulaire()
    End If
  End Sub
```

Les informations récupérées dans la session seront sauvegardées dans cette même fin de session à chaque fin de requête :

```
  Private Sub Page_PreRender(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.PreRender
    ' on mémorise certaines informations dans la session
    Session("dtProduits") = dtProduits
    Session("dvProduits") = dvProduits
    Session("nbProduitsPage") = nbProduitsPage
    Session("pageCourante") = pageCourante
  End Sub
```

L'événement [PreRender] est celui qui signale que la réponse va être envoyée au client. On en profite pour sauvegarder dans la session toutes les données à conserver. C'est excessif dans la mesure où assez souvent seules certaines d'entre-elles ont changé de valeur. Cette sauvegarde systématique a l'avantage de nous décharger de la gestion de session dans les autres méthodes de la page.

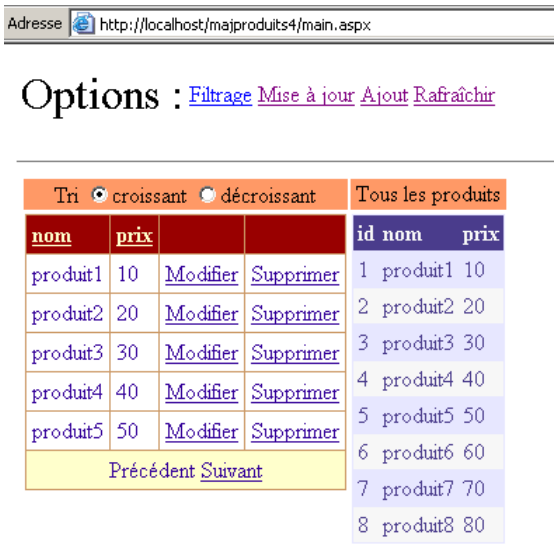
L'opération de rafraîchissement du cache est assurée par la procédure suivante :

```
  Private Sub lnkRefresh_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
lnkRefresh.Click
    ' on doit rafraîchir le cache [dtProduits] avec la source de données physique
    ' début synchro
    Application.Lock()
    Try
      ' la table des produits
      dtProduits = CType(Application("objProduits"), produits).getProduits
      ' on mémorise le filtre actuel
      Dim filtre As String = dvProduits.RowFilter
      ' on crée la nouvelle vue filtrée
      dvProduits = New DataView(dtProduits)
      ' on remet le filtre
      dvProduits.RowFilter = filtre
    Catch ex As ExceptionProduits
      'il y a eu erreur d'accès aux produits, on le note dans la session
      Session("erreurs") = ex.erreurs
      ' on fait afficher la vue [erreurs]
      afficheErreurs(ex.erreurs)
      ' fini
      Exit Sub
    Finally
      ' fin synchro
      Application.UnLock()
    End Try
    ' ça s'est bien passé - on affiche les produits à partir de la 1ère page
    afficheProduits(0, nbProduitsPage)
  End Sub
```

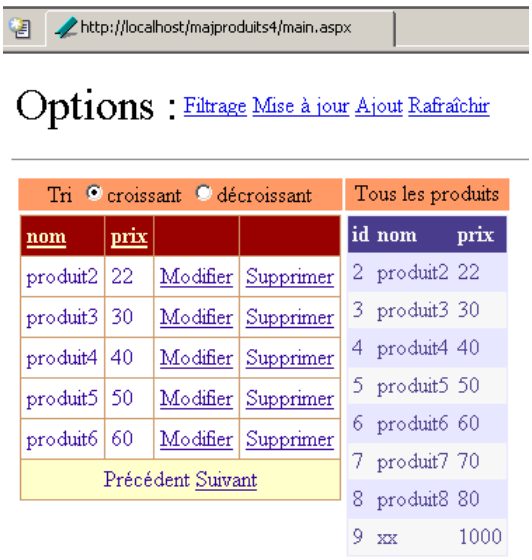
La procédure régénère de nouvelles valeurs pour le cache [dtProduits] et la vue [dvProduits]. Celles-ci seront placées dans la session par la procédure [Page_PreRender] décrite plus haut. Une fois le cache [dtProduits] reconstruit, on affiche les produits à partir de la première page.

Voici un exemple d'exécution. Un client [mozilla] est lancé et affiche les produits :

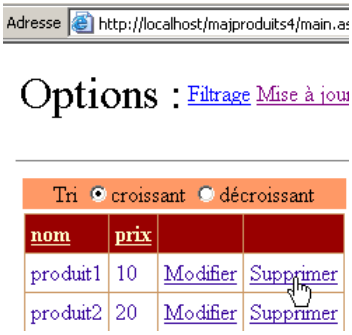
Un client [Internet Explorer] fait de même :



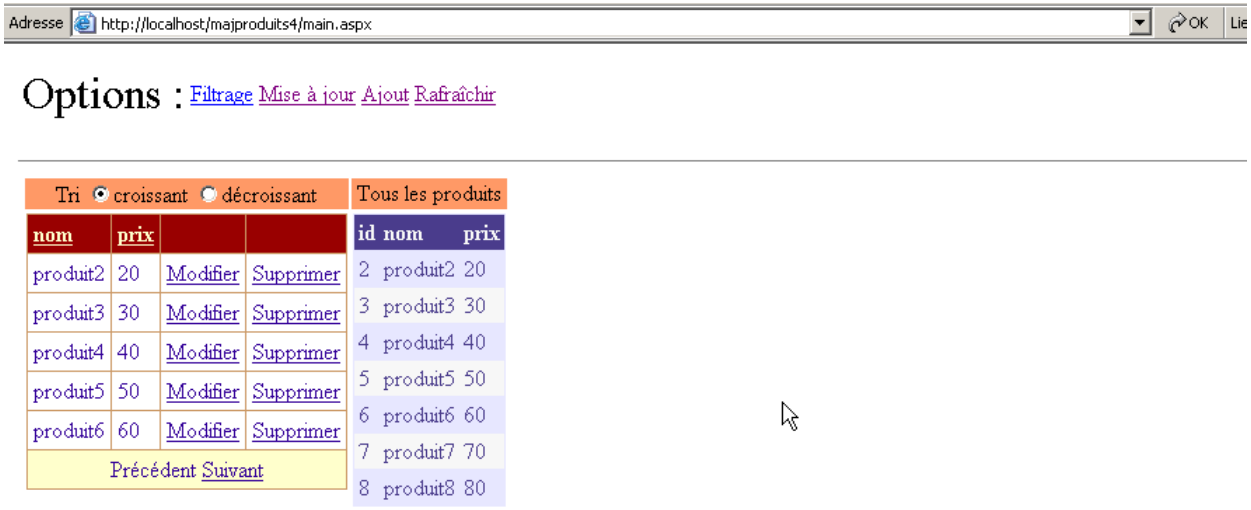
Le client [Mozilla] supprime [produit1] modifie [produit2] et ajoute un nouveau produit. Il obtient la nouvelle page suivante :



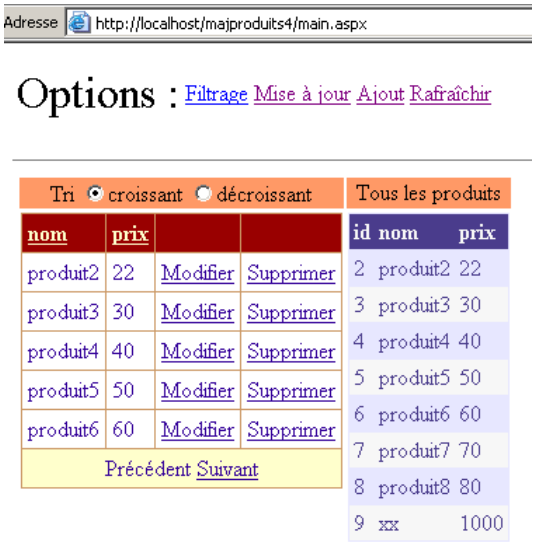
Le client [Internet Explorer] veut supprimer [produit1].



Il obtient la réponse suivante :



On lui a signalé que [produit1] n'existait plus. L'utilisateur décide alors de rafraîchir son cache avec le lien [Rafraîchir] ci-dessus. Il obtient la réponse suivante :



Il a maintenant la même source de données que le client [Mozilla].

3.8 Conclusion

Dans ce chapitre, nous avons passé un long moment sur les conteneurs de données et leur liaisons avec des sources de données. Nous avons terminé en montrant comment on pouvait mettre à jour une source de données à l'aide d'un composant [DataGrid].

Nous avons pris comme source de données une table de base de données. Même si le composant [DataGrid] facilite un peu les choses pour la représentation des données, la véritable difficulté ne réside pas dans la partie présentation mais bien dans la gestion des mises à jour de la source de données faites par les différents clients. Des conflits d'accès peuvent surgir et doivent être gérés. Ici, nous les avons gérés dans le contrôleur par l'utilisation de [Application.Lock]. Il serait probablement plus avisé de synchroniser les accès à la source de données dans la classe d'accès à celles-ci afin que le contrôleur n'ait pas à se préoccuper de tels détails qui ne sont pas de son ressort.
Dans la pratique, les tables d'une base de données sont liées entre-elles par des relations et leur mise à jour doit tenir compte de celles-ci. Cela a des conséquences tout d'abord sur la classe d'accès aux données qui devient plus complexe que celle nécessaire pour une table indépendante. Cela a en général également des conséquences sur la partie présentation de l'application web car il faut souvent afficher non pas une unique table mais les tables liées entre-elles par des relations.
Ce chapitre a permis, par ailleurs, de présenter différents structures de données telles que [DataTable, DataView].