Les contrôles serveurs spécialisés

Sommaire

Pourquoi dit-on de ces contrôles serveur qu'ils sont spécialisés ? Tout simplement parce qu'ils ont une tâche bien précise. Au contraire d'un *Label* qui affiche du texte ou d'une *TextBox* qui est une zone de saisie, ceux là permettent des choses plus poussées comme l'affichage et la gestion d'un calendrier, permettre l'envoi de fichiers sur le serveur etc …

Quasiment tous ces contrôles serveur héritent de la classe WebControl. Nous vous le dirons lorsque ce ne sera pas le cas.

1 Des contrôles serveurs spécialisés

Rappel : On peut ajouter un contrôle serveur grâce à un drag & drop depuis la *ToolBox*.

1.1 Literal

Tout comme le *Label*, le contrôle serveur *Literal* va afficher du texte. A la différence du *Label*, qui lui va rajouter des balises dans le code source, le contrôle serveur *Literal* ne va rajouter aucun code HTML. Ce qui veut dire qu'il est impossible de lui appliquer un style. On peut en revanche lui appliquer un ID et donc faire appel à lui depuis le code behind pour changer ses propriétés (comme Text, Mode, …).

Propriété	Valeur(s)	Définition
Text	Chaîne de caractère	Permet de changer le texte qui sera affiché.
Mode	-	Permet de changer le mode d'affichage du texte.
	PassThrough	Va afficher le texte comme si c'était du HTML (les balises HTML se trouvant dans le code sont donc prises en compte).
	Transform	Permet de convertir le code pour qu'il s'affiche comme s'il était dans le code source (donc les balises HTML seront prises en compte). Mais plus que le HTML on peut aussi y mettre du XHTML (eXtensible Hypertexte Markup Langage), WML Wireless Markup Langage, cHTML (Compact Hypertexte Markup Langage) qui seront affichés / transformés en html.
	Encode	Va afficher le texte brut, c'est-à-dire sans prendre en compte les balises HTML ou autres. Un texte contenant des balises HTML sera affiché avec les balises visibles sur la page.

Ainsi ce code :

Page.ASPX

```
<p>PassThrough</p>
      <asp:Literal ID="Literal1" runat="server" Text="Salut
&lt;strong&gt;tous&lt;/strong&gt; le <sub>monde</sub> !<script
type='text/javascript'>alert('Hello World !');</script>" Mode="PassThrough" /><br />

<p>Transform</p>
      <asp:Literal ID="Literal2" runat="server" Text="Salut
&lt;strong&gt;tous&lt;/strong&gt; le <sub>monde</sub> !<script
type='text/javascript'>alert('Hello World !');</script>" Mode="Transform" /><br />

<p>Encode</p>
      <asp:Literal ID="Literal3" runat="server" Text="Salut
&lt;strong&gt;tous&lt;/strong&gt; le <sub>monde</sub> !<script
type='text/javascript'>alert('Hello World !');</script>" Mode="Encode" />
```

Donnera ceci :

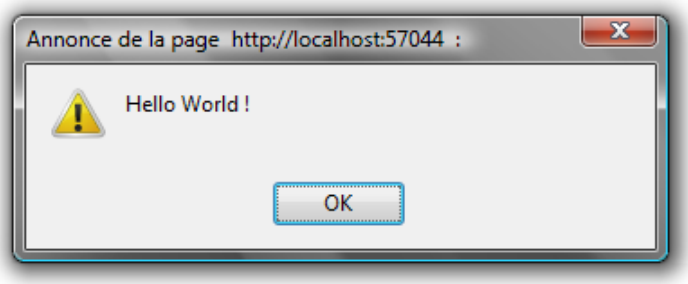
PassThrough

Salut **tous** le monde !

Transform

Salut **tous** le monde !

Encode



Salut tous le _{monde} !<script type='text/javascript'>alert('Hello World !');</script>

Les différents mode d'affichage du contrôle serveur Literal

Avec la fenêtre "Hello World !" qui s'ouvre pour *PassThrough* et *Transform*.

1.2 Table, TableRow, TableCell

Un tableau est un bon moyen pour mettre en forme/présenter des informations à l'utilisateur. Pour cela le contrôle serveur *Table* crée un tableau. *TableRow* correspond à une nouvelle ligne du tableau et *TableCell* correspond à une nouvelle cellule (plus exactement il fait une colonne comme en HTML).

Donc dans le *Table* on a une balise racine *Table*, puis des *TableRow* qui contiennent des *TableCell*. Il existe aussi des balises comme *TableHeaderRow* et *TableFooterRow*. *TableHeaderRow* est le Header du tableau. Il se situe donc en haut de celui-ci (c'est le titre de chaque colonne) : il correspond aux *th* en HTML. Le *TableFooterRow* fonctionne lui aussi comme en HTML, c'est-à-dire qu'il contient des cellules normales (*TableCell*).

Tout comme les autres contrôles serveur Web on peut le modifier ou même le créer depuis le code behind. Pour ajouter des *Rows* ou des *Cells* depuis le code behind il faut se servir de la méthode

Rows.Add qui attend un objet de type TableRow du tableau et de *Cells.Add* du *TableRow*. Plus précisément on se sert de la méthode *Add*() des collections *Cells* et *Rows*.

1.3 Image

On l'utilise pour afficher une image sur la page. Ce contrôle génère une balise HTML <img />.

Propriété	Valeur(s)	Définition
ImageUrl	Lien vers l'image (relatif ou absolu)	Indique le chemin pour accéder à l'image (correspond à l'attribut href de la balise img).
AlternateText	Chaîne de caractère	Permet de définir un texte alternatif (pour les cas où l'image ne pourrait pas être chargée). Correspond à l'attribut Alt de la balise img.
ImageAlign	NoSet, Left, Right, BaseLine, Top, Middle, Bottom, AbsBottom, AbsMiddle, TextTop	Définit l'alignement de l'image par rapport au reste de la page.
DescriptionUrl	Chaîne de caractère	Permet de donner une description de l'image (correspond à l'attribut longdesc de la balise img).
GenerateEmptyAlternateText	True, false	Permet de définir un Alt vide. Si l'image ne peut pas être chargé ou que le navigateur ne les supporte pas, rien se sera affiché (utile par exemple pour une image blanche ou de séparation).

Important : Il n'existe pas d'évènement *onclick* sur un contrôle *Image*. Pour l'avoir, il faut utiliser les contrôles *ImageButton* ou *ImageMap* que nous verrons dans les deux prochaines parties. Ces contrôles permettent de récupérer les coordonnées du *click*.

1.4 ImageButton

Il hérite de la classe *Image*. Il possède donc tout d'une *image* (*ImageUrl*, *AlternateText*, ...) mais sur laquelle on peut cliquer. Il va donc effectuer un *PostBack* de la page. Ce contrôle serveur va générer une balise de type <input type="image" />.

En revanche il possède un évènement *click* et *command*. Nous avons déjà parlé de l'évènement *click* dans les modules précédents. L'évènement *command* fait référence à l'activation d'un contrôle. Par exemple, on active un *Button* ou un *RadioButton* en cliquant dessus, un *DropDownList* en changeant la sélection …

Grâce à l'évènement *click*, *l'ImageButton* peut récupérer les coordonnées de la souris lors du clic.

1.5 ImageMap

ImageMap hérite lui aussi de la classe *Image*, il possède donc la plupart des propriétés *d'Image*. Ce contrôle serveur sert à définir une zone cliquable sur une image. Ce contrôle va rajouter l'attribut *usemap* sur la balise HTML *img* : (avec comme valeur le name de la map) et une balise <map name="NomMap"> qui contiendra des balises *area* dans le code source. Mais vous n'avez pas à vous en soucier car c'est du code généré automatiquement.

Voici ses propriétés :

Propriété	Valeur(s)	Définition
AccessKey	Un caractère	Permet de définir un raccourci (il faudra faire Alt + le caractère pour accéder à la zone).
HotSpotMode	NotSet, Inactive, Navigate, PostBack	Permet de définir le comportement du HotSpot lorsqu'il est cliqué.
NavigateUrl	Une URL	L'Url pour la navigation lorsque le HotSpot est cliqué.
TabIndex	Un numéro	Défini / change l'index du HotSpot.

Si le *HotSpotMode* est défini à *Navigate*, la propriété *NavigateUrl* sera prise en compte. S'il est à *PostBack* alors un *PostBack* sera effectué. S'il est à inactive rien ne sera fait.

La zone peut être de différentes formes : cela peut être un cercle (*CircleHotSpot*), un rectangle (*RectangleHotSpot)* ou un polygone (*PolygonHotSpot*). En anglais ces zones s'appellent des *area* ou *HotSpot* (zone cliquable pour faire une action).

Voici un exemple de code ASPX avec un *CircleHotSpot* :

ASPX

```
<asp:ImageMap ID="Image" ImageUrl="~/image.png" runat="server">

      <asp:CircleHotSpot AlternateText="Mon image" HotSpotMode="PostBack"
          PostBackValue="valeur" Radius="40" X="100" Y="100" />
      <asp:CircleHotSpot HotSpotMode="Inactive" Radius="60" X="100" Y="100" />

</asp:ImageMap>
```

1.6 Calendar

Ce contrôle affiche un calendrier du mois courant ou choisit (défini dans la page ASPX ou grâce au code behind, ce qui permet de demander à l'utilisateur). Il permet de choisir une date (on peut donc se déplacer d'un mois à l'autre, de même pour les années) et/ou d'afficher des informations sur les jours. C'est un contrôle complexe qui génère un tableau (un <table>).

Propriété	Valeur(s)	Définition
Caption	Chaine de caractères	Défini le texte qui sera affiché au dessus du calendrier en titre.
CaptionAlign	Top, Bottom, Right, Left, NotSet	Défini l'alignement du Caption (titre).
CellPadding	Taille	Défini l'espace entre le contenu d'une cellule et sa bordure (soit la marge intérieure).
CellSpacing	Taille	Défini la distance entre chaque cellule.
SelectionMode	Day	Spécifie que l'on peut sélectionner un jour.
	DayWeek	Un jour ou un weekend
	DayWeekMonth	Un jour, un weekend ou un mois entier
	None	On ne peut pas sélectionner de jour.

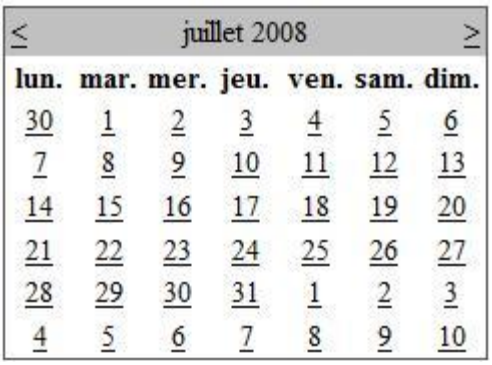
L'évènement *SelectionChanged* est programmé pour que dès qu'il y a un changement de la date (donc que l'utilisateur choisit une date), un *PostBack* s'effectue. Une fois que l'utilisateur a choisit une date et que le *PostBack* a été effectué, on peut accéder à la date choisi via la propriété *SelectedDates*.

Exemple d'un calendrier de base :

ASPX

```
<asp:Calendar ID="Calendar1" runat="server" />
```

Et voici ce que sa donne :



Calendrier basique

Le calendrier peut être utilisé pour afficher un emploi du temps.

1.7 FileUpload

FileUpload est un contrôle qui va afficher une *TextBox* avec un bouton « Parcourir » qui va permettre de sélectionner un fichier de notre disque dur et de l'envoyer (il ajoute un <input type="file" dans le code source). Une fois le fichier téléchargé il faudra vérifier que le fichier ne présente aucun risque pour le serveur et qu'il correspond à ce que l'on attend (extension, type MIME ect …). Avant cela voyons son fonctionnement un peu plus en détails.

Il va générer automatiquement le code pour créer la *TextBox* et le bouton "Parcourir". Pour le bouton "Envoyer" il faut le rajouter soit même. Ce n'est qu'un contrôle de type *Button* auquel on met le texte "Envoyer" et qui va effectuer le *PostBack*. L'utilisateur va parcourir ses fichiers, sélectionner le bon et cliquer sur « Envoyer ». C'est lors du *PostBack* que le fichier sera envoyé au serveur. Remarquez que le chargement (le *PostBack*) et l'affichage de la page suivante ne se termine qu'après la fin de l'envoi du fichier. Une fois le fichier reçu on a accès à des informations comme : son nom, sa taille, son type MIME (Multipurpose Internet Mail Extension, c'est un système qui définit le type de fichier). Jusque là le fichier est dans la mémoire du serveur. Pour le sauvegarder il existe une méthode *SaveAs* que l'on peut utiliser sur le contrôle *FileUpload* ou sur l'objet *HttpPostedFile*. Il faut bien entendu avoir les droits d'écriture dans le dossier où vous voulez l'enregistrer. Par défaut, l'attribut *requireRootedSaveAsPath* de la configuration *HttpRuntime* qui se trouve dans le *Web.config* est à *true*. Ce qui veut dire qu'il faut mettre un lien absolu pour sauver le fichier.

La valeur de *MaxRequestLength* (qui se trouve dans le *Web.config*) va déterminer la taille maximale du fichier qui peut être envoyé.

Voici le contrôle *FileUpload* par défaut :

Image du contrôle FileUpload par défaut

Bien évidement on peut lui appliquer un style etc …

1.8 Panel

Un *Panel* est un contrôle serveur utilisé comme un conteneur. C'est à dire que l'on va le placer à un endroit et qu'il contiendra d'autres contrôles (cela peut servir par exemple à afficher, cacher un groupe de contrôle). Nous verrons plus tard une autre utilisation lors des master pages. Il va générer dans le code source des balises . La propriété *BackImageUrl* permet de définir l'image de fond du *Panel* et *HorizontalAlignement* définit l'alignement horizontal du *Panel*.

1.9 MultiView et View

Un contrôle serveur View est un conteneur au même titre que Panel à la différence près qu'il possède un index. Seul un View peut être affiché. Par conséquence, il est pratique d'afficher un View à la suite de l'autre. Un View ne peut se situer que dans un MultiView. A la différence de Panel, View ne crée aucune balise dans le code source. Il permet surtout de gérer un groupe de vue qui seront affichées les unes après les autres, système qui devra être géré par le code behind. Le plus important à retenir est qu'il ne peut y avoir plusieurs View visible en même temps. Les View sont numérotées en commençant par 0. Une fois le(s) View dans le MultiView il faut que la propriété ActiveViewIndex du MultiView soit initialisée sinon aucun des View ne sera affiché. Elle prend comme valeur le numéro d'index d'un des View.

Ce sera plus compréhensible avec un exemple :

ASPX

```
<asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="0"> <%-- On remarque
qu'il faut la propriété ActiveViewIndex. Sinon aucun View ne serait affiché --%>

    <asp:View ID="View1" runat="server"> <%-- Ce View aura l'index 0 --%>
          <asp:Label ID="Label1" runat="server" Text="View Numéro 0" />
          <asp:Button ID="Button1" runat="server" Text="Button1" OnClick="But1" />
    </asp:View>

    <asp:View ID="View2" runat="server"> <%-- Ce View aura l'index 1 --%>
          <asp:Label ID="Label2" runat="server" Text="View Numéro 1" />
          <asp:Button ID="Button2" runat="server" Text="Button2" OnClick="But2" />
    </asp:View>

    <asp:View ID="View3" runat="server"> <%-- Ce View aura l'index 2 --%>
          <asp:Label ID="Label3" runat="server" Text="View Numéro 2" />
          <asp:Button ID="Button3" runat="server" Text="Button3" OnClick="But3" />
    </asp:View>

</asp:MultiView>
```

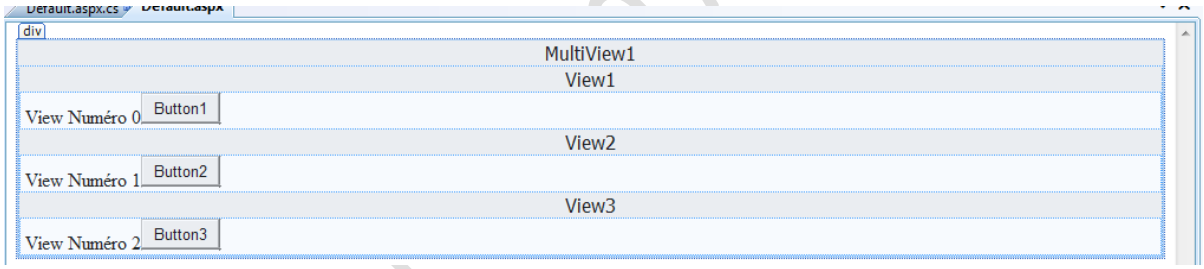
C#

```
protected void But1(object sender, EventArgs e)
{ MultiView1.ActiveViewIndex = 1; }

protected void But2(object sender, EventArgs e)
{ MultiView1.ActiveViewIndex = 2; }

protected void But3(object sender, EventArgs e)
{ MultiView1.ActiveViewIndex = 0; }
```

VB.NET

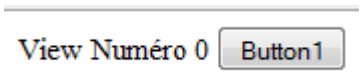
```
Protected Sub But1(ByVal sender As Object, ByVal e As System.EventArgs)
    MultiView1.ActiveViewIndex = 1
End Sub

Protected Sub But2(ByVal sender As Object, ByVal e As System.EventArgs)
    MultiView1.ActiveViewIndex = 2
End Sub

Protected Sub But3(ByVal sender As Object, ByVal e As System.EventArgs)
    MultiView1.ActiveViewIndex = 0
End Sub
```

Ce qui donne ceci en mode Design :



Affichage du code en mode Design

Et dans le navigateur :



Affichage du code dans le navigateur

Il n'affiche que le premier *View* (qui a un index de 0). Si on clique sur le bouton on va activer l'affichage du second *View* (avec l'index 1) et donc le serveur va cacher le premier *View*. Et ainsi de suite.

1.10 Wizard

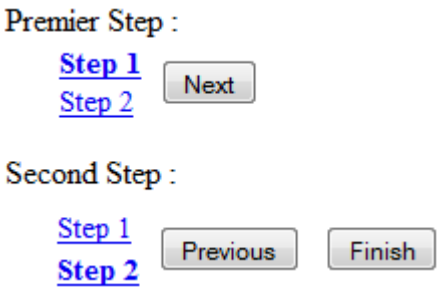
Il nous arrive d'avoir besoin de demander beaucoup de données à l'utilisateur par exemple lors d'un enregistrement sur un site, pour remplir un formulaire particulièrement long et même pour les sites de ventes (la succession de : vue du panier, confirmation, données sur où livrer et autres, paiement, facture). Pour cela le contrôle serveur Wizard est pratique puisqu'il permet de faire un formulaire pas à pas. Il va permettre tout comme avec des View d'afficher des vues ou écran pas à pas (l'une après l'autre). Ainsi on pourra demander beaucoup d'informations à l'utilisateur tout en restant dans quelque chose qui est organisé. On aurait pu faire de même sur plusieurs pages ou en utilisant les View. Mais là nous récupérons toutes les données sur la même page en utilisant un contrôle qui regroupe déjà toutes les fonctionnalités dont on a besoin pour cela. Ce contrôle hérite directement de la classe Control.

En effet, à la différence d'un View ou MultiView, ce contrôle gère déjà le passage d'une vue à l'autre (pas besoin de rajouter de code) et prend aussi en charge le système de titre pour chaque vue tout en affichant en même temps une liste de toute les vues dans un menu. Il gère aussi la navigation entre chacun d'eux (Suivant, Précédent, Finish à la fin). Par défaut les boutons sont en anglais mais tout est paramétrable.

Lors de l'ajout d'un contrôle Wizard dans le code source ou en mode Design, il crée automatiquement deux Step (Step qui veut dire "pas" en anglais).

ASPX

```
<asp:Wizard ID="Wizard1" runat="server">
    <WizardSteps>
        <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
        </asp:WizardStep>

        <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
        </asp:WizardStep>
    </WizardSteps>
 </asp:Wizard>
```



Wizard de base

Remarque : Les WizardStep sont inclus dans un WizardSteps.

Chaque WizardStep possède une méthode AllowReturn qui prend un booléen en valeur et qui définit si on autorise ou non l'utilisateur à revenir sur cette vue.

1.11 XML

Il est utilisé pour afficher le contenu d'un document XML ou le résultat de l'exécution d'une transformation Extensible Stylesheet Langage (XSL). Un fichier XSL est un peu comme du CSS : il va permettre de définir la mise en page du XML. Toutefois il permet aussi de modifier totalement la structure du XML, ce qui permet depuis le XML de générer d'autres types de document comme du HTML, PostScript, Tex, RTF .....).

Propriété	Valeur(s)	Définition
DocumentSource	String	Spécifie l'emplacement du fichier XML qui va être utilisé.
DocumentContent	String	Accepte un string qui est du XML. Il sera utilisé comme si c'était le fichier. Si DocumentSource est aussi défini, ne sera utilisé que le dernier à être défini dans le cycle de vie de la page.
TransformSource	String	Spécifie l'emplacement du fichier XSL qui permettra de mettre en forme le XML.

Si on passe à ce contrôle du XML pur sans XSL il n'y aura aucun affichage.

Voici le contenu du fichier XML pour notre exemple :

```xml
<?xml version="1.0" encoding="utf-8" ?>
<personneListe>
  <Personne nom="Dupond" prenom="Dupont" age="35" />
  <Personne nom="Machin" prenom="Truc" age="27" />
  <Personne nom="Nom" prenom="Prenom" age="42" />
</personneListe>
```

Contenu du fichier XSL

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">

<xsl:template match="/">
	<html>
		<head>
			<title>Affichage du XML Personne.xml</title>
		</head>
		<body>
			<h1>Affichage du XML Personne.xml</h1>
			<xsl:call-template name="CreerArborescencePage" />
		</body>
	</html>
</xsl:template>

<xsl:template name="CreerArborescencePage">
	<table border="1">
		<tr>
			<th>Nom</th>
			<th>Prenom</th>
			<th>Age</th>
		</tr>
		<xsl:call-template name="RemplirTableau" />
	</table>
</xsl:template>

<xsl:template name="RemplirTableau">
	<xsl:for-each select="/personneListe/Personne/">
		<tr>
			<td>
				<xsl:value-of select="@nom" />
			</td>
			<td>
				<xsl:value-of select="@prenom" />
			</td>
			<td>
				<xsl:value-of select="@age" />
			</td>
	</xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

On rempli les propriétés DocumentSource avec "~/Personne.xml" et TransformSource avec "~/Personne.xsl". Qu'on le fasse depuis le code behind ou la page ASPX on obtient le résultat montré par l'image de droite.

Affichage du XML Personne.xml

Nom	Prenom	Age
Dupond	Dupont	35
Machin	Truc	27
Nom	Prenom	42

Pour en savoir plus sur le XSL visiter ce lien : W3C (http://www.w3.org/TR/xslt)

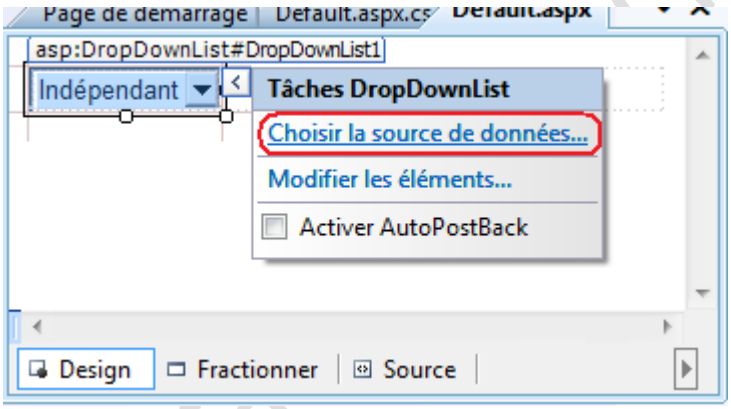
2 Le bind, qu'est-ce ?

Le bind c'est lier des données avec un contrôle. Au moment de l'exécution, le Framework va chercher les données qui se situent dans une base de données (quelconque que nous aurons défini) pour remplir les champs du contrôle qui seront créés par rapport au nombre d'éléments à afficher. Par exemple on peut binder une DropDownList.

En exemple nous allons faire un bind sur une DropDownList depuis une base de données locale.

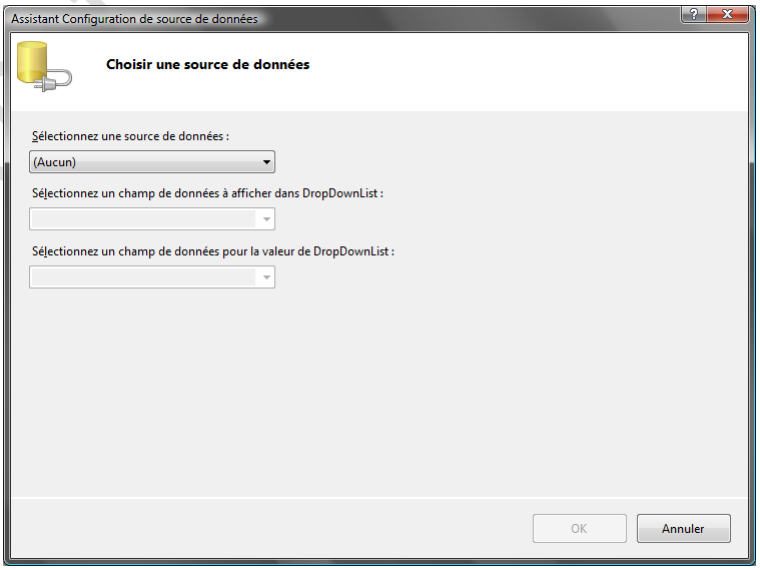
Mode Graphique :

Ajoutons, par exemple, un DropDownList (ou tout autre contrôle serveur sur lequel on peut faire un bind). Ensuite il faut aller en mode Design et passer la souris sur le contrôle en question. Une petite flèche apparaît en haut à droite : cliquez dessus. Une nouvelle fenêtre apparait qui doit ressembler à ceci.



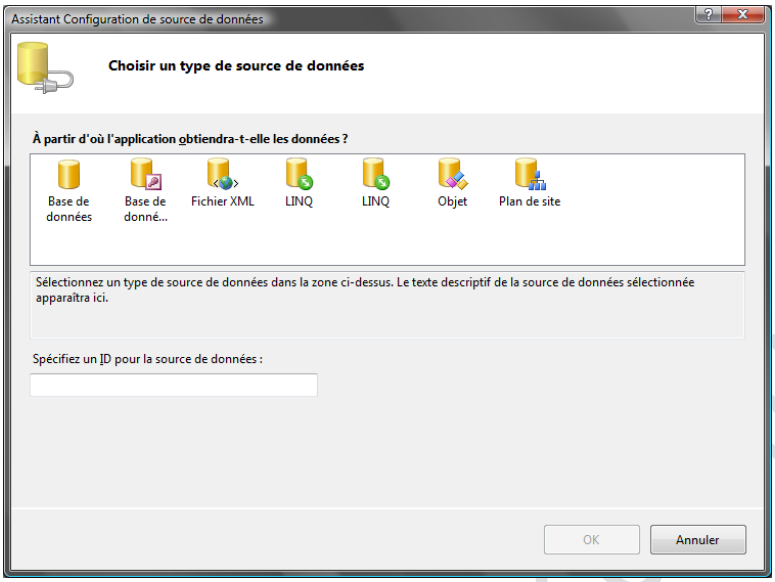
Bind en mode graphique

Ce qui nous intéresse c'est "Choisir la source de données…" : cliquez dessus.



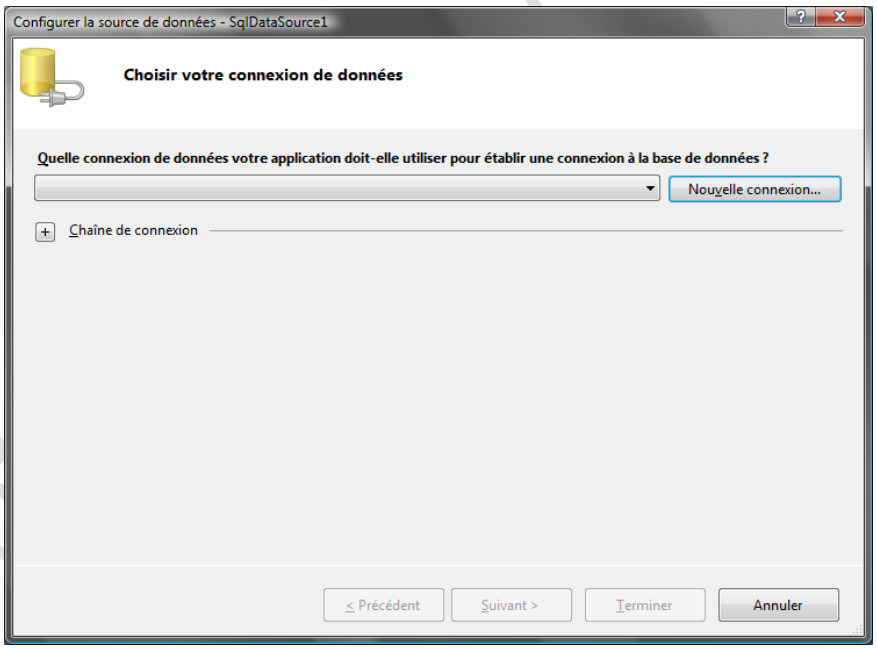
Choix de la source de données

Dans le menu déroulant du choix de la source de données choisissez "<Nouvelle connexion>" à moins que vous ayez déjà celle que vous voulez. Vous arrivez sur cette fenêtre qui vous propose plusieurs types de source de données possible.



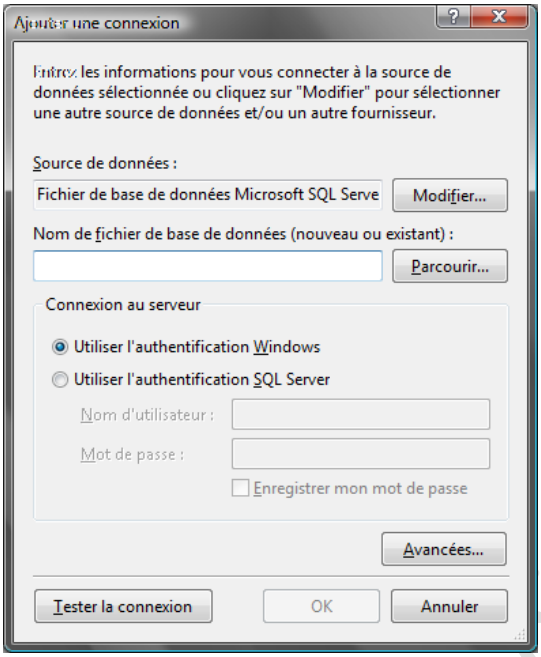
Choix du type de source de données

Choisissez celle qui vous intéresse (dans notre cas ce sera Base de données). On arrive sur ceci :



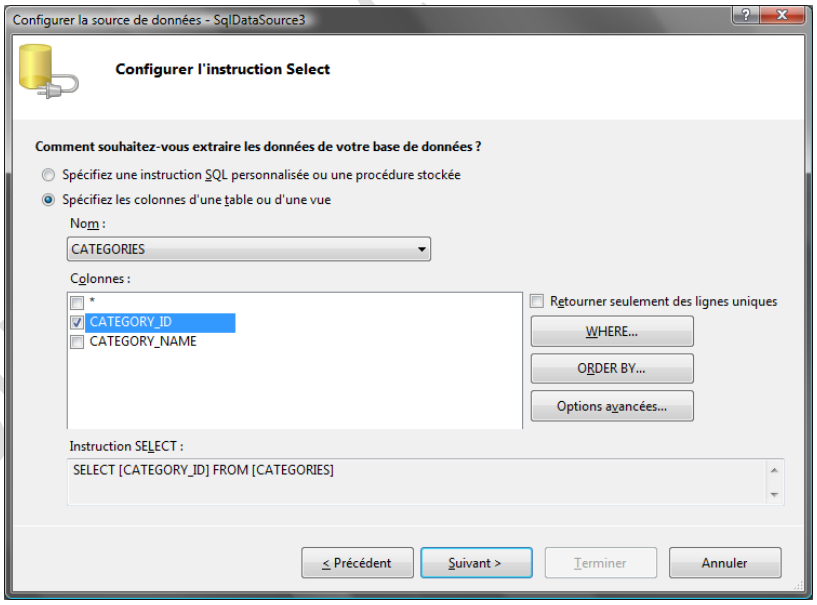
Nouvelle connexion SQL

Choisissez Nouvelle connexion et vous arriverez sur un écran qui va vous demander des informations.



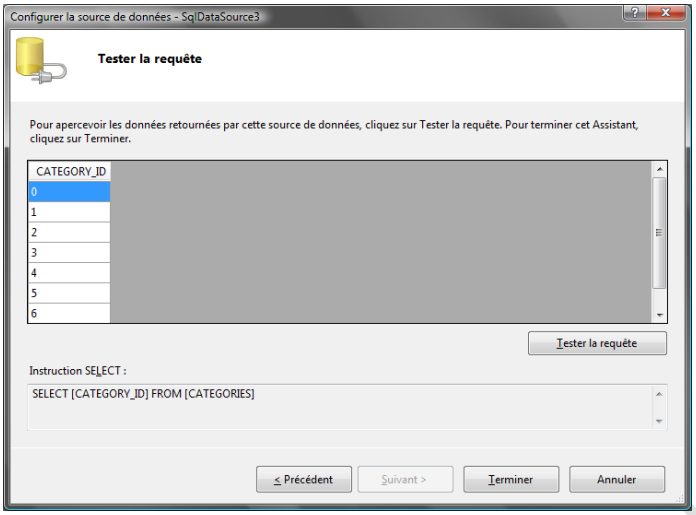
Information sur la nouvelle connexion

Faites Parcourir pour lui donner le fichier .mdf qu'il demande, entrez les informations sur l'authentification, si besoin est, puis cliquer sur « Ok », puis sur suivant. On arrive alors sur une autre fenêtre qui va vous proposer de créer la requête pour extraire les données. On va faire simple dans notre cas : nous ne prendrons que l'ID. A savoir que l'on peut choisir entre les différentes tables et vues etc …



Création de la requête permettant d'extraire les données

A l'écran suivant on peut tester notre requête pour voir les données retournées. Si on est satisfait du test, on termine sinon on fait « Précédent » pour corriger la requête.



Tester sa requête avant de terminer

Une fois qu'on a cliqué sur Terminer, on revient sur la toute première fenêtre qu'on a eu. On choisit alors (dans le cas d'une DropDownList) une donnée à afficher et une donnée pour la valeur. Plus qu'a lancer le projet pour vérifier que cela a bien fonctionné.

Ceci fait vous remarquerez dans la source qu'une balise a été rajoutée :

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$
ConnectionStrings:MyNewConnectionString %>" SelectCommand="SELECT [CATEGORY_ID]
FROM [CATEGORIES]" />.
```

`SqlDataSource` est un contrôle qui permet la liaison de données avec une base de données SQL, elle est obligatoire pour établir la connexion. `SelectCommand` est la requête qui va permettre la récupération des données et `ConnectionString` est la chaîne de connection à la base de données. Le contrôle sur lequel vous appliquez le bind a maintenant une propriété DataSourceId qui contient l'ID du SqlDataSource contenant votre connexion à la base de données. Ensuite suivant le type de contrôle d'autres choses seront rajoutées.

Le `ConnectionString` fait référence à une ligne du Web.config :

```
<connectionStrings>
<add name="supinShopConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\supinShop.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True"
providerName="System.Data.SqlClient" />
 </connectionStrings>
```

La balise connectionString contient toutes les connexions que l'on a créé. La balise add est à chaque fois une nouvelle connexion. Le name de celle-ci correspond à ce qui est écrit dans la propriété ConnectionString du SqlDataSource. Suis ensuite le providerName. Concrètement c'est le namespace du fournisseur d'accés à utiliser.

La suite de ce module va nous permettre de voir plus en détail comment on bind des contrôles. Comme nous venons de voir une méthode graphique, on spécifiera pour chaque contrôle quels sont les balises qu'il doit y avoir dans le code source.

3 Les contrôles serveur Data-Bound

3.1 Introduction

Les contrôles Data-Bound, sont des contrôles qui ont nécessairement besoin d'un bind pour pouvoir fonctionner correctement. Ils sont classés en trois catégories : simple, composé et hiérarchique. Les simples sont ceux qui héritent de la classe ListControl. Les composés héritent de la classe CompositeDataBoundControl comme les contrôles GridView, DetailsView et FormsView. Les hiérarchiques sont les contrôles Menu et TreeView. Lorsque l'on bind un contrôle, tous les contrôles fils de ce contrôle le seront aussi par récursivité.

3.2 Utilisation d'une collection pour le bind

Lorsque vous allez faire votre bind, quoiqu'il arrive une collection sera récupérée pour remplir les contrôles. Dans les exemples que nous ferons, nous utiliserons une collection de personnes dont voici la classe :

Classe Personne C#

```
using System.Collections.Generic; //Ne pas oublier de rajouter cette assembly

namespace WebApplication1
{
    public class Personne //Début de la classe
    {
        //Déclaration des attributs en private
        private string p_Nom;
        private string p_Prenom;
        private int p_Age;

        public Personne() //Constructeur vide
        { }

        public Personne(string nom, string prenom, int age) //Surcharge du
constructeur avec les attributs nom, prenom et age
        {
            p_Nom = nom;
            p_Prenom = prenom;
            p_Age = age;
        }

        //Création d'accesseur / mutateur
        public string Nom
        {
            get { return p_Nom; }
            set { p_Nom = value; }
        }

        public string Prenom
        {
            get { return p_Prenom; }
            set { p_Prenom = value; }
        }

        public int Age
        {
            get { return p_Age; }
            set { p_Age = value; }
        }

        //Création de la collection d'objet personne
        public static List<Personne> GetList()
        {
              List<Personne> personneListe = new List<Personne>();
              personneListe.Add(new Personne("Dupond", "Dupont", 35));
            personneListe.Add(new Personne("Machin", "Truc", 27));
            personneListe.Add(new Personne("Nom", "Prenom", 42));
            return personneListe;
        }
    }
}
```

Classe Personne VB.NET

```
Imports System.Collections.Generic 'Ne pas oublier de rajouter cette assembly pour
la création de la collection

Public Class Personne 'Début de la classe Personne

    'Déclaration des attributs
    Private p_Prenom As String
    Private p_Nom As String
    Private p_Age As Integer

    Public Sub New() 'Constructeur vide

    End Sub

    Public Sub New(ByVal prenom As String, ByVal nom As String, ByVal age As
Integer) 'Surcharge du constructeur
        p_Prenom = prenom
        p_Nom = nom
        p_Age = age
    End Sub

    'Accesseur / Mutateur
    Public Property Prenom() As String
        Get
            Return p_Prenom
        End Get
        Set(ByVal value As String)
            p_Prenom = value
        End Set
    End Property

    Public Property Nom() As String
        Get
            Return p_Nom
        End Get
        Set(ByVal value As String)
            p_Nom = value
        End Set
    End Property

    Public Property Age() As Integer
        Get
            Return p_Age
        End Get
        Set(ByVal value As Integer)
            p_Age = value
        End Set
    End Property

    'Création de la collection d'objet Personne
    Public Shared Function GetList() As List(Of Personne)
        Dim personneListe As New List(Of Personne)

        personneListe.Add(New Personne("Dupond", "Dupont", 35))
        personneListe.Add(New Personne("Machin", "Truc", 27))
        personneListe.Add(New Personne("Nom", "Prenom", 42))

        Return personneListe
    End Function

End Class
```

Pour effectuer un bind sur une collection on doit passer par le code behind. Aussi ce que nous vous présenterons dans la suite sera expliqué à partir du code behind.

3.3 La classe ListControl

La classe ListControl est une classe abstraite dont héritent les contrôles qui suivent dans cette partie. Ils font partie de la catégorie simple. Cette classe contient une collection d'Items qui sont des objets de type ListItem. ListItem contient une propriété Text qui sera affiché à l'utilisateur et une propriété valeur qui sera envoyée au serveur. Les objets ListItem peuvent être rempli en ajoutant de nouveaux objets ListItem dans le code ou en paramétrant les propriétés DataSource et DataMember. Si on paramètre ces deux propriétés on peut choisir respectivement la source de données et les champs qui contiennent les valeurs pour le bind. Comme on l'a vu lors du bind on crée une requête qui va donc retourner plusieurs valeurs et peut être plusieurs champs. On peut choisir parmi eux lesquels seront affichées pour le bind des propriétés ListItem.Text et ListItem.Value. Pour cela, on va remplir les propriétés DataTextField et DataValueField (respectivement) qui vont spécifier quel champ des valeurs retournées va correspondre au champ de texte ou de valeur. On peut aussi formater le texte qui sera affiché grâce à la propriété DataTextFormatString.

La propriété SelectedIndex permet de définir quel sera l'objet sélectionné par défaut. Avec SelectedItem on peut accéder aux propriétés de l'Item sélectionné. Si vous avez seulement besoin d'accéder à la valeur du ListItem sélectionné, utilisez plutôt la propriété SelectedValue. Elle possède aussi un évènement SelectedIndexChanged qui se produit lorsque le ListItem sélectionné change. La propriété AppendDataBoundItems définit à true permet de garder les ListItem déjà existant. Sinon ils sont enlevés avant de lier les données.

Les contrôles suivant héritent de cette classe et possèdent donc toutes ces propriétés.

3.3.1 DropDownList

Nous avons déjà vu dans le module précédent que c'est une liste d'item qui permet à l'utilisateur de pouvoir sélectionner qu'un seul item sous forme de liste déroulante. Nous allons voir comment le binder.

Pour cela il faut remplir un minimum d'informations. Comme la méthode graphique a déjà été décrite (rappelons qu'elle fonctionne pour tous contrôles que l'on peut lier à des données), nous allons décrire pour ce premier contrôle les deux autres manières de le faire. La première étant de remplir les propriétés dans le code source de la page ASPX et la seconde de le faire depuis le code behind.

Pour ce premier contrôle nous allons faire quelque chose de plus complexe que pour les autres. On va montrer comment récupérer la valeur pour l'afficher dans un TextBox à l'aide d'un bouton.

ASPX

```
<asp:DropDownList ID="DropDownList1" runat="server" />
<asp:Button ID="Button1" runat="server" Text="Button" />
<asp:Label ID="Label1" runat="server" Text="Rien pour l'instant" />
```

C#

```
using System.Collections.Generic; //Ajout de l'assembly

namespace WebApplication1
{
    public partial class _Default : System.Web.UI.Page
    {
        private List<Personne> personneListe = Personne.GetList(); //On récupère
notre collection

        protected void Page_Load(object sender, EventArgs e) //Lors du chargement de
la page
        {

            if (!IsPostBack)// Si ce n'est pas un PostBack = premier chargement de
la page => On bind le contrôle
            {
                DropDownList1.DataSource = personneListe; //On défini la source (ici
avec le nom de la collection)
                DropDownList1.DataTextField = "Prenom"; //Nom de l'accesseur qui va
être récupéré pour afficher le texte
                DropDownList1.DataValueField = "Nom"; //Accesseur qui donnera la
valeur qui sera lié au texte
                DropDownList1.DataBind(); //On bind le contrôle
            }
            else //Sinon c'est un PostBack
            {
                Label1.Text = DropDownList1.SelectedValue; //On rempli le label avec
la valeur de la DropDownList
            }
        }
    }
}
```

VB.NET

```
Imports System.Collections.Generic 'Ne pas oublier de rajouter cette assembly pour
la création de la collection

Partial Public Class _Default
    Inherits System.Web.UI.Page

    Private personneListe As List(Of Personne) = Personne.GetList 'On récupère notre
collection

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load 'Lors du chargement de la page

        If Not IsPostBack Then 'Si ce n'est pas un PostBack = premier chargement de
la page => On bind le contrôle
            DropDownList1.DataSource = personneListe 'On défini la source (ici avec
le nom de la collection)
            DropDownList1.DataTextField = "prenom" 'Nom de l'accesseur qui va être
récupéré pour afficher le texte
            DropDownList1.DataValueField = "Nom" 'Accesseur qui donnera la valeur
qui sera lié au texte
            DropDownList1.DataBind() 'On bind le contrôle
        Else 'Sinon c'est un PostBack
            Label1.Text = DropDownList1.SelectedValue 'On rempli le label avec la
valeur de la DropDownList
        End If

    End Sub
End Class
```

N'oubliez pas de rajouter l'espace de noms `System.Collections.Generic` dans le code behind .

Nous avons créé une DropDownList qui se bind sur la collection que nous avons créé. Lors de la selection d'un item de cette liste, rien ne change. Lors de l'appui sur le bouton, un PostBack est effectué et la valeur séléctionnée est envoyée au serveur. Il y a execution du code : comme on est au Postback on ne recharge pas les item dans le DropDownList mais on affiche dans le label la valeur séléctionnée (et pas le texte affiché car nos champs valeur et texte ne sont pas les même !!).

A partir de maintenant et pour les autres contrôles de type liste vous penserez à ajouter l'assembly System.Collections.Generic quand on fait une liste et à créer la collection avec `private List<Personne> personneListe = Personne.GetList();`.

3.3.2 ListBox

C'est une liste d'item dans laquelle l'utilisateur peut sélectionner un ou plusieurs items. La propriété SelectionMode permet d'activer / désactiver le choix multiple. La propriété Rows permet de définir le nombre de ligne du ListBox (donc sa hauteur).
Voici un exemple avec une balise ListBox par défaut (`<asp:ListBox ID="ListBox1" runat="server" />`) dans la page ASPX :

C#

```
protected void Page_Load(object sender, EventArgs e)
{
     ListBox1.SelectionMode= ListSelectionMode.Multiple;

     ListBox1.DataSource = personneListe; //On défini la source (ici avec le nom de
la collection)
     ListBox1.DataTextField = "Prenom"; //Nom de l'accesseur qui va être récupéré
pour afficher le texte
     ListBox1.DataValueField = "Nom"; //Accesseur qui donnera la valeur qui sera lié
au texte
     ListBox1.DataBind(); //On bind le contrôle
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load 'Lors du chargement de la page

     ListBox1.SelectionMode = ListSelectionMode.Multiple

     ListBox1.DataSource = personneListe 'On défini la source (ici avec le nom de la
collection)
     ListBox1.DataTextField = "prenom" 'Nom de l'accesseur qui va être récupéré pour
afficher le texte
     ListBox1.DataValueField = "Nom" 'Accesseur qui donnera la valeur qui sera lié
au texte
     ListBox1.DataBind() 'On bind le contrôle
End Sub
```

A partir de maintenant on va estimer que vous savez faire un bind sur ce type de contrôle. Jusqu'au prochain type nous allons donc juste vous décrire les contrôles.

3.3.3 CheckBoxList et RadioButtonList

Ils sont assez similaires sur le fait que ce sont tous les deux des cases à cocher. Nous avions déjà vu les CheckBoxList dans le module précédent. Il est possible de sélectionner plusieurs éléments dans sa liste d'item. En revanche le RadioButtonList ne permet la sélection que d'un et un seul item. Une fois qu'on a activé le RadioButtonList on ne peut pas le désactiver au contraire du CheckBoxList (on ne peut pas décocher : on peut juste changer la sélection).

Ils possèdent la propriété RepeatColumns qui va définir le nombre de colonne sur lequel il va s'étendre (à l'instar d'un tableau). RepeatDirection définit dans quelle direction il va devoir se répéter (horizontal ou vertical).

3.3.4 BulletedList

Ce contrôle permet d'afficher une liste à puces ordonné ou non (c'est à dire avec ou sans numéro) qui correspond respectivement aux balises ol et ul en HTML.

Propriété	Valeur(s)	Définition

BulletStyle	Non ordonné : Disc, Circle, Square Ordonnée : Lower Alpha, Upper Alpha, Lower Roman, Upper Roman	Permet de choisir le style de puce (Attention ils ne sont pas focrement tous compatibles avec tous les navigateurs).
FirstBulletNumber	Nombre	Définit le premier nombre de la liste si elle est ordonnée.
DisplayMode	Text, LinkButton, HyperLink	Définit la façon dont ce sera affiché (pour les deux dernières valeurs possible un PostBack est effectué sur l'évènement onClick).

3.3.5 AdRotator

Ce contrôle permet d'afficher d'une manière aléatoire une bannière de pub ou d'annonce. Ce contrôle génère des balises <a> et <img /> en HTML. Il peut être fait à partir d'un fichier XML ou d'une base de données. Nous allons vous le présenter avec un fichier XML (qui va afficher une image comme celle de droite). Ce fichier doit être constitué d'une certaine manière pour que le contrôle puisse récupérer les informations seul. Le tableau montre de quoi doit être constitué le XML.

Tag	Définition
Keyword	Définit le mot clé de la bannière. Il peut être utilisé pour filtrer les bannières et ne retenir que celles qui sont intéressantes pour le site/ la page en question.
ImageUrl	L'url de l'image qui doit être affichée
NavigateUrl	L'url de navigation lors du clic sur l'image
AlternateText	Texte qui doit être affiché si l'image ne peut être affichée
Impressions	Prend comme valeur un nombre qui permet la pondération de la fréquence d'affichage de la bannière. Plus le nombre est grand par rapport aux autres plus il va s'afficher souvent. La somme de toutes les impressions ne doit pas dépasser 2 048 000 000
Height	Spécifie la hauteur de la bannière
Width	Spécifie la largeur de la bannière

Il faut faire attention à la configuration et à l'emplacement des fichiers de la bannière pour ne pas avoir de failles de sécurité. Les fichiers doivent être placés dans App_Data parcequ'Asp.NET ne permet pas aux navigateurs de requêter un fichier s'y trouvant (il autorise par contre un fichier local comme votre page ASPX à y accéder). Utiliser une extension de fichier comme .config pour le fichier XML car l'ASP.NET empêche l'accès aux navigateurs là aussi. Il faut mettre les permissions du fichier de sorte que l'ASP.NET ne puisse que le lire (read only).

Code de la page ASPX

```
<asp:AdRotator ID="AdRotator1" runat="server" DataSourceId="SourceXML" />
<asp:XmlDataSource DataFile="~/App_Data/Annonce.config" ID="SourceXML"
runat="server" />
```

```xml
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements xmlns="http://schemas.microsoft.com/AspNet/AdRotator-Advertisement-File-1.2">
  <Ad xmlns="">
    <Keyword>Pub</Keyword>
    <ImageUrl>~/Pub.png</ImageUrl>
    <NavigateUrl>#</NavigateUrl>
    <AlternateText>Voici ma jolie pub.</AlternateText>
    <Impressions>100</Impressions>
  </Ad>
  <Ad xmlns="">
    <Keyword>Microsoft</Keyword>
    <ImageUrl>~/Microsoft.png</ImageUrl>
    <NavigateUrl>http://www.microsoft.com</NavigateUrl>
    <AlternateText>Microsoft</AlternateText>
    <Impressions>100</Impressions>
  </Ad>
  <Ad xmlns="">
    <Keyword>Venez</Keyword>
    <ImageUrl>~/venezvoir.png</ImageUrl>
    <NavigateUrl></NavigateUrl>
    <AlternateText>Il faut le voir pour le croire !</AlternateText>
    <Impressions>50</Impressions>
  </Ad>
</Advertisements>
```

3.4 CompositeDataBoundControl

CompositeDataBoundControl est une classe qui va servir pour l'héritage de contrôles qui vont contenir d'autres contrôles sur lesquels se fera la liaison de données. Cette classe implémente l'interface INamingContainer. Toute classe qui implémente cette interface est un conteneur (qui aura des contrôles enfants).

GridView, DetailsView et FormView héritent de cette classe et font donc partie de la catégorie des contrôles composés.

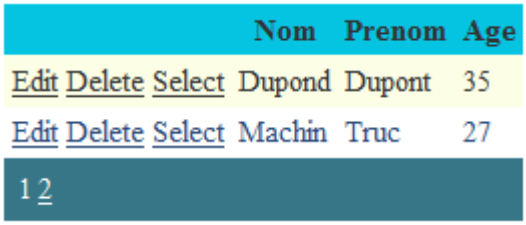
Pour ces contrôles nous utiliserons une autre classe nommée ClassePersonne dont voici le code :

C#

```
using System.Collections.Generic;

namespace WebApplication1
{
    public class ClassePersonne
    {
        private static List<Personne> personneListe;

        public static void Initialisation()
        {
            personneListe = Personne.GetList();
        }

        public List<Personne> Selection()
        {
            return personneListe;
        }

        public void MettreAJour(Personne updatePersonne)
        {
            Personne personneTrouve = personneListe.Find(
                delegate(Personne personne){ return personne.Nom ==
                  updatePersonne.Nom; });
            personneTrouve.Prenom = updatePersonne.Prenom;
            personneTrouve.Age = updatePersonne.Age;
        }

        public void Inserer(Personne personne)
        {
            personneListe.Add(personne);
        }

        public void Supprimer(Personne supPersonne)
        {
            Personne personneTrouve = personneListe.Find(
                delegate(Personne personne) { return personne.Nom ==
                  supPersonne.Nom; });
            personneListe.Remove(personneTrouve);
        }

        public int Count()
        {
            return personneListe.Count;
        }
    }
}
```

```
VB.NET

Imports System.Collections.Generic 'Ne pas oublier de rajouter cette assembly pour
la création de la collection

Public Class ClassePersonne
    Private Shared personneListe As List(Of Personne)
    Private current As Personne

    Private Function VerifNom(ByVal MatchPersonne As Personne) As Boolean
        Return IIf(current.Nom = MatchPersonne.Nom, True, False)
    End Function

    Public Shared Sub Initialisation()
        personneListe = Personne.GetList();
    End Sub

    Public Function [Selection]() As List(Of Personne)
        Return personneListe
    End Function

    Public Sub MettreAJour(ByVal updatePersonne As Personne)
        current = updatePersonne
        Dim personneTrouve As Personne = personneListe.Find(AddressOf VerifNom)
        personneTrouve.Prenom = updatePersonne.Prenom
        personneTrouve.Age = updatePersonne.Age
    End Sub

    Public Sub Inserer(ByVal personne As Personne)
        personneListe.Add(personne)
    End Sub

    Public Sub Supprimer(ByVal supPersonne As Personne)
        current = supPersonne
        Dim personneTrouve = personneListe.Find(AddressOf VerifNom)
        personneListe.Remove(personneTrouve)
    End Sub

    Public Function Count() As Integer
        Return personneListe.Count
    End Function

End Class
```

3.4.1 GridView



Le GridView est un contrôle très utilisé et complexe car il permet d'afficher des données sous forme de tableau. Il peut posséder des fonctionnalités (représenté avec des liens) comme Edit, Select, Delete et cela sans écrire beaucoup de code. Le GridView est constitué d'une collection d'objet GridViewRow (qui correspond aux lignes) et d'une collection d'objet DataControlField (qui constitue les colonnes). Grâce à cela on peut modifier les données d'une cellule depuis le code behind.

Pour pouvoir utiliser le GridView, on a besoin de définir la source de donnée. Dans notre cas ce sera un ObjectDataSource dont voici les propriétés importantes (pour un exemple voir celui qui se trouve en bas de cette partie) :

Propriété(s)	Valeur(s)	Définition
DataObjectTypeName	String	Nom de la classe qui permet de créer la liste d'objet
TypeName	String	Nom de la classe contenant les méthodes permettant de communiquer avec la liste d'objet
SelectMethod, DeleteMethod, UpdateMethod, InsertMethod	String	Définissent les méthodes à utiliser dans la classe déclaré dans la propriété TypeName pour, respectivement, la sélection, supprimer, la mise à jour et l'insertion de données.
InsertParameters, UpdateParameters, SelectParameters, DeleteParameters	-	Définissent les paramètres qui seront utilisés pour, respectivement, l'insertion, la mise à jour, la sélection et la suppression de données. Elles possèdent elles même deux propriétés importantes : Name qui défini le nom de l'accesseur qui se situe dans le classe qui créer la liste (déclaré dans DataObjectTypeName). Et la propriété Type qui défini le type attendu (String, Boolean, Char ….).

Passons aux propriétés du GridView :

Propriété	Valeur(s)	Définition
AutoGenerateColumns	Booléen	Par défaut mise à true, il affiche tous les champs. Mise à false, le GridView n'affichera que les champs défini dans la propriété Columns. S'il est à true et que Columns est défini alors il y aura une redondance des informations (il affichera d'abord les champs défini dans le Columns puis dans les colonnes suivantes tous les champs).
Columns	-	Permet de définir des propriétés comme BoundField, CommandField et autres qui servent à récupérer les champs envoyé par la source de données que l'on veut afficher. CommandField est la colonne des commandes. BoundField est une colonne que l'on bind, on peut leurs appliquer une propriété ReadOnly qui permet de n'accéder a ce champ qu'en lecture : on ne pourra donc pas le modifier par le biais de l'update.
DataSourceID	String	Prend en paramètre l'ID du contrôle serveur qui permet la communication avec la source de données.
DataKeyNames	String	Nom de l'accesseur qui sera en ReadOnly lors du bind automatique avec AutoGenerateColumns.
AllowPaging	Booléen	Par défaut elle est mise à false : tous les enregistrements sont affiché en même temps. En la mettant à true on les affiche par page (par défaut 10 enregistrements par page). On peut changer le nombre d'enregistrement par page grâce à la propriété PageSize qui attend un Integer.

PageSize	Integer	Nombre d'enregistrement sur une page (AllowPaging à true)

Le GridView contient en plus des propriétés pour modifier son style (son apparence). On peut cibler précisément quel élément du GridView on veut modifier :

Propriété	Définition
FooterStyle	Apparence du pied de page
RowStyle	Apparence des lignes
EditRowStyle	Apparence de la ligne en mode modifiaction (edit)
SelectedRowStyle	Apparence de la ligne sélectionné
PagerStyle	Apparence de la ligne du pagineur
HeaderStyle	Apparence de la ligne d'en-tête
AlternatingRowStyle	Apparence des lignes de données en alternance (1 ligne sur 2)

Voici un exemple de GridView avec la classe créé dans la partie juste au dessus :

Page ASPX

```
   <asp:ObjectDataSource ID="Test" runat="server"
         DataObjectTypeName="WebApplication1.Personne"
         TypeName="WebApplication1.ClassePersonne"
         SelectMethod="Selection" UpdateMethod="MettreAJour"
         DeleteMethod="Supprimer" InsertMethod="Inserer">
             <InsertParameters>
                <asp:Parameter Name="Nom" Type="String" />
                <asp:Parameter Name="Prenom" Type="String" />
                <asp:Parameter Name="Age" Type="Int32" />
             </InsertParameters>
             <UpdateParameters>
                <asp:Parameter Name="Nom" Type="String" />
                <asp:Parameter Name="Prenom" Type="String" />
                <asp:Parameter Name="Age" Type="Int32" />
             </UpdateParameters>
   </asp:ObjectDataSource>

   <asp:GridView ID="GridView1" runat="server"
         Style="position: absolute; top: 75px; left: 20px;"
         AllowPaging="True" PageSize="2" AutoGenerateColumns="False"
         DataSourceID="Test" Width="135px" CellPadding="4"
         DataKeyNames="Nom" ForeColor="#333333" GridLines="None">
         <Columns>
            <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
                 ShowSelectButton="True" />
            <asp:BoundField DataField="Nom" HeaderText="Nom" SortExpression="Nom"
                 ReadOnly="True" />
            <asp:BoundField DataField="Prenom" HeaderText="Prenom"
                 SortExpression="Prenom" />
            <asp:BoundField DataField="Age" HeaderText="Age" SortExpression="Age" />
         </Columns>
     <HeaderStyle BackColor="#05c4e2" Font-Bold="True" />
     <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
     <EditRowStyle BackColor="#0755b3" />
     <RowStyle BackColor="#fcffe6" ForeColor="#333333" />
     <PagerStyle BackColor="#367687" ForeColor="White" HorizontalAlign="Left" />
</asp:GridView>

<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Charger le
GridView" />
```

C#

```
public void Button1_Click(object sender, EventArgs e)
{
    ClassePersonne.Initialisation();
    GridView1.DataBind();
}
```

VB.NET

```
    Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
        ClassePersonne.Initialisation()
        GridView1.DataBind()
   End Sub
```

Explication : Ce code va afficher un bouton sur lequel on va cliquer pour afficher le GridView peuplé.

Le code behind permet le bind du GridView à la suite de l'évènement click. Il utilise la méthode Initialisation de la classe ClassePersonne qui va initialiser la liste de personne. La méthode DataBind permet de peupler le GridView avec les informations récupérées de la source de données.

Passons à l'explication du code de la page ASPX. Le contrôle ObjectDataSource contient les informations pour la communication entre le GridView et la classe ClassePersonne. Si vous regardez bien DataObjectTypeName et TypeName, ils sont précédés du nom du namespace pour que l'ASP.NET sache exactement où sont situés la classe et le type d'objet (ici le namespace étant WebApplication1). PageSize et AllowPaging définissent que ce sera affiché comme des pages et qu'il y a 2 enregistrements par page.

La mise à jour de donnée ne s'enregistrera pas car notre méthode MettreAJour ne fait que changer les valeurs dans la liste de personne et non dans le fichier XML.

3.4.2 DetailsView

Outre le fait que l'affichage est différent du GridView, il se bind exactement de la même façon. Ce contrôle permet d'avoir une présentation, et donc une approche, différente des données. Ce ne sont plus des données linéaires que l'on affiche les unes en dessous des autres mais des fiches qui présentent chaque donnée une par une.

Nom	Dupond
Prenom	Dupont
Age	35
Edit Delete New	
1 2 3	

Voici un exemple de ce que cela donne (l'image de présentation représente le code qui suit) :

ASPX

```
<asp:ObjectDataSource ID="Test" runat="server"
      DataObjectTypeName="WebApplication1.Personne"
      TypeName="WebApplication1.ClassePersonne"
      SelectMethod="Selection" UpdateMethod="MettreAJour"
      DeleteMethod="Supprimer" InsertMethod="Inserer">
</asp:ObjectDataSource>

<asp:DetailsView ID="DetailsView1" runat="server"  AllowPaging="True"
      CellPadding="4" DataSourceID="Test" ForeColor="#333333"
      GridLines="None" Height="50px"  Style="z-index: 103; left: 20px; position:
      absolute; top: 85px" Width="305px"
      AutoGenerateRows="False" DataKeyNames="Nom">
      <Fields>
         <asp:BoundField DataField="Nom" HeaderText="Nom" SortExpression="Nom"
              ReadOnly="True" />
         <asp:BoundField DataField="Prenom" HeaderText="Prenom"
              SortExpression="Prenom" />
         <asp:BoundField DataField="Age" HeaderText="Age" SortExpression="Age" />
         <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
              ShowInsertButton="True" />
      </Fields>

      <FieldHeaderStyle BackColor="#E9ECF1" Font-Bold="True" />
      <HeaderStyle BackColor="#05c4e2" Font-Bold="True" />
      <AlternatingRowStyle BackColor="White" />
      <EditRowStyle BackColor="#0755b3" />
      <RowStyle BackColor="#fcffe6" ForeColor="#333333" />
      <PagerStyle BackColor="#367687" ForeColor="White" HorizontalAlign="Left" />
      <CommandRowStyle BackColor="#007c05" ForeColor="White" />
</asp:DetailsView>

<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Charger le
DetailsView" />
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
    ClassePersonne.Initialisation();
    DetailsView1.DataBind();
}
```

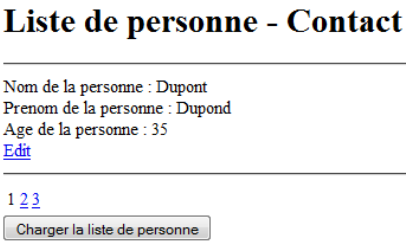
VB.NET

```
Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   ClassePersonne.Initialisation()
   DetailsView1.DataBind()
End Sub
```

3.4.3 FormView

Tout comme le DetailsView, ce contrôle est utilisé pour n'afficher qu'un seul enregistrement à la fois. La différence ce situe dans le fait que ce soit nous qui créons le template qui sera chargé de l'afficher à l'utilisateur. On peut créer différent template pour les vues d'affichage, de mise à jour, de suppression … C'est par conséquent un contrôle très flexible au niveau de l'affichage des données.

Dans la plupart des cas il est intéressant et/ou suffisant d'utiliser le GridView ou le DetailsView. On utilise ce contrôle que quand on veut vraiment afficher les données d'une manière différente du GridView et du DetailsView.



Aperçu d'un FormView

Liste des propriétés importantes (regardez dans le code de l'exemple pour bien comprendre comment s'en servir) :

Propriété	Définition
ItemTemplate	Template pour l'affichage de la sélection / de l'enregistrement en cours
EditItemTemplate	Template pour l'édition de l'enregistrement
EmptyDataTemplate	Template pour l'affichage par défaut (quand la liste est vide)
InsertItemTemplate	Template pour l'insertion
HeaderTemplate, FooterTemplate	Respectivement template pour le header (haut de page) et le footer (pied de page)
PagerTemplate	Template pour l'affichage des liens vers les différentes pages

Voici un exemple simple de ce que cela peut donner :

Page ASPX

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
       DataObjectTypeName="WebApplication1.Personne"
       TypeName="WebApplication1.ClassePersonne"
       SelectMethod="Selection" UpdateMethod="MettreAJour"
       DeleteMethod="Suppression" InsertMethod="Inserer">
</asp:ObjectDataSource>


<asp:FormView ID="FormView1" runat="server" AllowPaging="True"
   DataKeyNames="Nom" DataSourceID="ObjectDataSource1">
       <ItemTemplate>
          <asp:label runat="server">Nom de la personne : <%# Eval("Nom")
           %></asp:label><br />
          <asp:label runat="server">Prenom de la personne : <%# Eval("Prenom")
           %></asp:label><br />
          <asp:label runat="server">Age de la personne : <%# Eval("Age")
           %></asp:label><br />
          <asp:LinkButton ID="LinkButton1" runat="server" CausesValidation="False"
           CommandName="Edit" Text="Edit" />
    </ItemTemplate>

    <EditItemTemplate>
          <asp:label runat="server">Nom de la personne : <%# Eval("Nom")
           %></asp:label><br />
          <asp:label runat="server">Prenom de la personne : </asp:label> <br />
          <asp:TextBox id="TextBox1" runat="server" Text='<%# Bind("Prenom") %>'
           /><br />
          <asp:label runat="server">Age de la personne :</asp:label><br />
          <asp:TextBox ID="TextBox5" runat="server" Text='<%# Bind("Age") %>' /><br
           />
          <asp:LinkButton runat="server" CausesValidation="True"
           CommandName="Update" Text="Update" /> <br />
          <asp:LinkButton runat="server" CausesValidation="False"
           CommandName="Cancel" Text="Cancel" />
    </EditItemTemplate>

   <EmptyDataTemplate>
          <h1> Liste de personne vide </h1>
          <asp:LinkButton ID="LinkButton2" runat="server"  CausesValidation="False"
           CommandName="New" Text="New" />
   </EmptyDataTemplate>
```

Suite ASPX

```
   <InsertItemTemplate>
          <asp:label runat="server">Nom de la personne :</asp:label><br />
          <asp:TextBox ID="TextBox2" runat="server" Text='<%# Bind("Prenom") %>'
            /><br />
          <asp:label runat="server">Prenom de la personne : </asp:label> <br />
          <asp:TextBox ID="TextBox3" runat="server" Text='<%# Bind("Prenom") %>'
            /><br />
          <asp:label runat="server">Age de la personne :</asp:label><br />
          <asp:TextBox ID="TextBox4" runat="server" Text='<%# Bind("Age") %>' /><br
            />
        <asp:LinkButton runat="server" CausesValidation="True" CommandName="Insert"
           Text="Insert" /><br />
        <asp:LinkButton ID="LinkButton2" runat="server" CausesValidation="False"
           CommandName="Cancel" Text="Cancel" /><br />
   </InsertItemTemplate>

   <HeaderTemplate>
        <hr />
        <h1 style="font-weight:bold;">Liste de personne - Contact</h1>
        <hr />
   </HeaderTemplate>

   <FooterTemplate>
        <hr />
   </FooterTemplate>
</asp:FormView>

   <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Charger la liste de personne" />
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   ClassePersonne.Initialisation();
   FormView1.DataBind();
}
```

VB.NET

```
Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   ClassePersonne.Initialisation()
   FormView1.DataBind()
End Sub
```

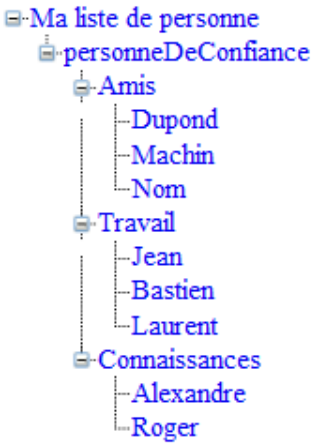
Au tout début, lors du lancement du projet, vous remarquerez que la liste est vide. C'est normal puisqu'on ne crée et charge la liste que lors d'un clique sur le bouton. D'ailleurs à cause de cela on ne peut pas faire New, la liste n'étant pas créée vous aurez une erreur. En dehors de cela, dans le code on trouve <%# Eval(« Nom ») %>. C'est du code C# qui va utiliser l'accesseur Nom pour récupérer le nom (Eval utilise Get et Bind utilise Get pour afficher et Set lors d'un update). Cet exemple n'est pas très avancé puisqu'il n'y a pas de CSS. Vous pouvez bien sur en appliquer comme d'habitude.

3.5 HierarchicalDataBoundControl

C'est un contrôle qui sert d'héritage pour les contrôles qui doivent afficher des données sous une forme hiérarchique. Les contrôles qui en héritent sont TreeView et Menu étudiés dans les parties suivantes.

3.5.1 TreeView

Ce contrôle va créer une hiérarchie à partir des données qui lui seront données. Elle aura la forme d'une arborescence de fichier et/ou de document comme vous le voyez sur l'image à droite. Pour cela ce contrôle attend qu'on lui fournisse des données à organiser. Les nœuds de ce contrôle peuvent être liés avec un fichier XML ou une base de données. Dans notre cas ce sera un fichier XML.



Ce contrôle peut permettre la navigation sur un site s'il est utilisé avec le contrôle SiteMapDataSource.

On peut accéder aux propriétés de ce contrôle, comme pour tous les autres, depuis le code behind. Chaque entrée dans l'arbre est appelée un nœud (par exemple dans le code que nous utilisons pour l'exemple plus bas, les nœuds seraient : groupe, personneDeConfiance ...). Un nœud qui contient un autre nœud est appelé un nœud parent, le nœud contenu dans un autre est un nœud enfant/fils. Un nœud qui n'a pas de nœud fils est appelé un nœud feuille (parce qu'il est au bout de l'arbre). Le nœud qui est à la base de tout les autres nœuds est appelé le nœud root ou le nœud racine. Généralement il n'y a qu'un seul nœud racine dans un fichier XML. Mais vous pouvez en mettre plusieurs.
Un nœud peut être configuré pour être sélectionnable ou être un lien. Pour qu'il soit configuré en lien il faut définir la propriété NavigateUrl. Si cette propriété est mise à nulle alors ce sera un nœud sélectionnable.

La propriété TextField va définir ce qui va correspondre au texte à afficher dans le TreeView. La propriété ValueField fait de même pour la valeur. Lorsque l'on rempli un contrôle on dit qu'on le peuple. Pour peupler ce contrôle il y a deux façon : le peupler avec du texte static ou le remplir avec un bind.

Pour le remplir avec du texte static on met un tag <Nodes> dans le TreeView puis on rajoute à l'intérieur des balises <asp:TreeNode>

Pour le bind on peut utiliser n'importe quelle source de données qui est implémentée dans l'interface IHierarchicalDataSource (qui est l'interface implémentée pour tous les contrôles que l'on peut lier à des données). Il suffit simplement de définir le DataSourceId avec l'ID du contrôle qui définit la source, par exemple <XmlDataSource /> ou <SiteMapDataSource />. L'exemple a été fait

avec XmlDataSource. Il faut rajouter le tag <DataBindings> pour le bind qui contiendra des balises <asp:TreeNodeBinding />. Si on ne met que la balise <DataBindings> le TreeView se remplira quand même mais ce ne sera pas forcément ce qu'on attend de lui puisque la propriété Text sera le nom de chaque nœud et que la propriété Value sera égale à Text.

Voici le code que l'on utilise pour l'image vu plus haut :

Fichier XML

```
<?xml version="1.0" encoding="utf-8" ?>
<personneListe id="1" name="Ma liste de personne">
  <personneDeConfiance>
    <groupe id="1.1" name="Amis">
      <Personne id="1.1.1" name="Dupond" prenom="Dupont" age="35"/>
      <Personne id="1.1.2" name="Machin" prenom="Truc" age="27"/>
      <Personne id="1.1.3" name="Nom" prenom="Prenom" age="42"/>
    </groupe>

    <groupe id="1.2" name="Travail">
      <Personne id="1.2.1" name="Jean" prenom="Papin" age="54"/>
      <Personne id="1.2.2" name="Bastien" prenom="Martinez" age="47"/>
      <Personne id="1.2.3" name="Laurent" prenom="Fournier" age="45"/>
    </groupe>

    <groupe id="1.3" name="Connaissances">
      <Personne id="1.3.1" name="Alexandre" prenom="Girault" age="23"/>
      <Personne id="1.3.2" name="Roger" prenom="Delmas" age="26"/>
    </groupe>
  </personneDeConfiance>
</personneListe>
```

Code de la page ASPX

```
<asp:XmlDataSource runat="server" DataFile="~/ArborescencePersonne.xml" ID="Source"
/>

<asp:TreeView ID="TreeView1" runat="server"  DataSourceId="Source" ShowLines="true">
   <DataBindings>
         <asp:TreeNodeBinding DataMember="personneListe" TextField="name"
valuefield="id" />
         <asp:TreeNodeBinding DataMember="groupe" TextField="name" ValueField="id"
/>
         <asp:TreeNodeBinding DataMember="Personne" TextField="Name" ValueField="id"
/>
   </DataBindings>
</asp:TreeView>
```

3.5.2 Menu

NavigationMenuSite
- Home
- Cours
  - AspNET
- Telechargement
  - Framework
  - Winform

Tout comme le TreeView, Menu est un contrôle hiérarchique qui peut être utilisé avec un contrôle SiteMapDataSource pour permettre la navigation sur le site.

Il peut lui aussi être peuplé statiquement, auquel cas on rajoute le tag <Items> dans le Menu puis on met dans ce tag des balises asp:MenuItem.

Pour le bind il s'agit de la même manipulation que le TreeView. Il faut définir une source de données puis informer la propriété DataSourceId du contrôle avec l'ID du contrôle définissant la source de donnée. Un exemple est donné à la fin de la partie avec une source XML qui reprend l'image de présentation.
Encore une fois comme le TreeView, si on ne définit rien il se bind automatiquement. Mais cela n'a pas beaucoup d'intérêt puisque comme on l'a vu, la propriété Text sera le nom du nœud et la Value aussi. On utilise donc là aussi la balise <DataBindings> que l'on rempli avec des <asp:MenuItemBinding>. Les propriétés importantes à remplir sont DataMember, TextField et ValueField. Nous faisons l'exemple avec le bind auto pour vous montrer ce que cela donne. Pour voir avec la balise <DataBindings> reportez vous au TreeView.

Fichier XML

```
<?xml version="1.0" encoding="utf-8" ?>
<NavigationMenuSite>
  <Home display="Page d'acceuil"/>

  <Cours display="Cours">
    <AspNET display="Webform"/>
    <Framework display="Framework" />
    <Winform display="Winform" />
  </Cours>

  <Telechargement display="Téléchargement" >
    <Webcast display="Webcast" />
    <Zip display="Fichier Zip" />
  </Telechargement>
</NavigationMenuSite>
```

Page ASPX

```
<asp:Label ID="Label1" runat="server" Text="Rien pour l'instant"></asp:Label><br />
<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>

<asp:Menu ID="Menu1" runat="server" DataSourceID="SourceXML"
OnMenuItemClick="Menu_Click" />
<asp:XmlDataSource DataFile="~/Navigation.xml" XPath="/NavigationMenuSite"
ID="SourceXML" runat="server" />
```

C#

```
protected void Menu_Click(object sender, MenuEventArgs e)
{
   Label1.Text = e.Item.ValuePath;
   Label2.Text = Menu1.SelectedValue;
}
```

VB.NET

```
Protected Sub Menu_Click(ByVal sender As Object, ByVal e As MenuEventArgs)
   Label1.Text = e.Item.ValuePath
   Label2.Text = Menu1.SelectedValue
End Sub
```

Le résultat est l'image de présentation du Menu se trouvant au début de cette partie.

4 Conclusion

Nous avons vu des contrôles spécialisés dans certains domaines comme l'affichage de données sous forme de liste (DropDownList, RadioButtonList, CheckBoxList) mais aussi d'autres spécialisés dans l'affichage de données récupéré à partir d'une source qui peut être une base de données, un fichier XML, une liste d'objet …. Le plus important de ces derniers étant le GridView qui est très utilisé.

Pour binder des données, nous avons vu des contrôles comme DataSourceObject et XmlDataSource, permettant de référencer la source et de donner les informations nécessaires pour récupérer et manipuler les données. Petite précision : hormis le DataSourceOject, aucun ne respecte la couche n-tiers car ils vont chercher directement dans les données (DatasourceObject utilise une classe comme interface avec la source de données).