Les nouveautés d'ASP.NET 4

par Louis-Guillaume Morand (Espace Perso) (Blog) Philippe Vialatte (Espace Perso) (Blog)

Date de publication : 07 October 2009

Dernière mise à jour :

Les nouveautés d'ASP.NET 4 fait partie d'une suite d'articles écrits par l'équipe .NET de Developpez.com, et destinée à vous faire découvrir les nouveautés en ce qui concerne Visual Studio 2010, la version 4 du Framework .NET les langages C# et VB.NET, ainsi que les technologies associées comme WPF 4, ASP.NET 4, WF 4, WCF 4, Entity Framework 4 et autres nouveautés autour de la plateforme .NET.
Dans cet article, nous allons nous concentrer sur le framework ASP.NET, en excluant les nouveautés venant du framework MVC et d'Ajax, qui feront l'objet d'un autre article.
N'hésitez pas à laisser votre avis sur le contenu de l'article directement via notre forum :

I - Introduction

Avec la sortie de la version 4 du framework imminente, cet article va se concentrer sur la partie Webform d'ASP.NET.

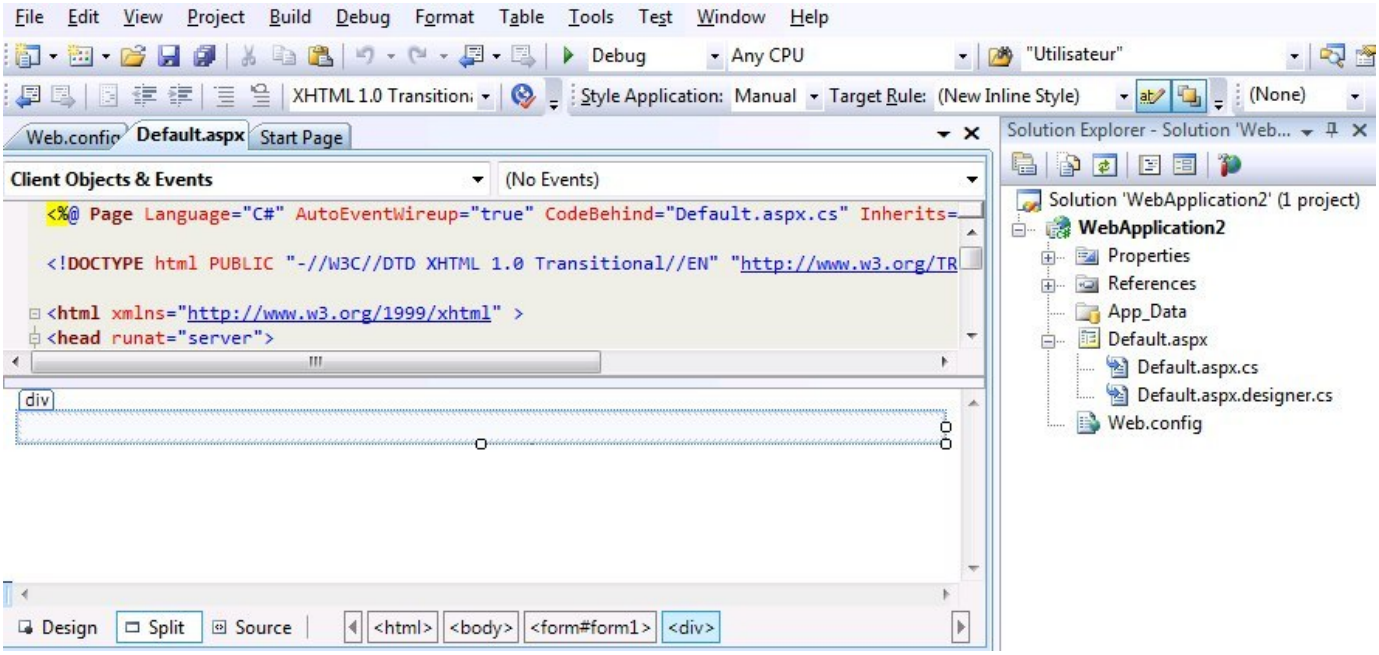
La version 4 n'apporte pas de nouveautés aussi marquantes et importantes que Linq, ou autre expressions lambda, mais comporte de nombreuses améliorations de points douloureux pour les développeurs Web, ainsi que quelques nouvelles fonctionnalités.

Il est aussi à noter que de gros efforts ont été faits à cette occasion pour se rapprocher encore plus des standards du Web.

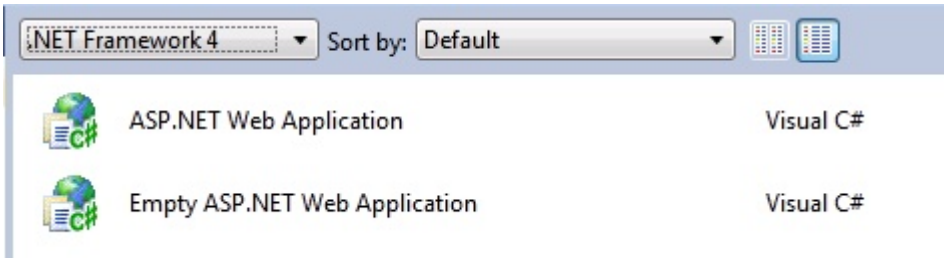
II - Modification des modèles de projet

Les modèles de projets Web ont été repensés pour permettre à chaque développeur de commencer un projet de la façon dont il l'entends.

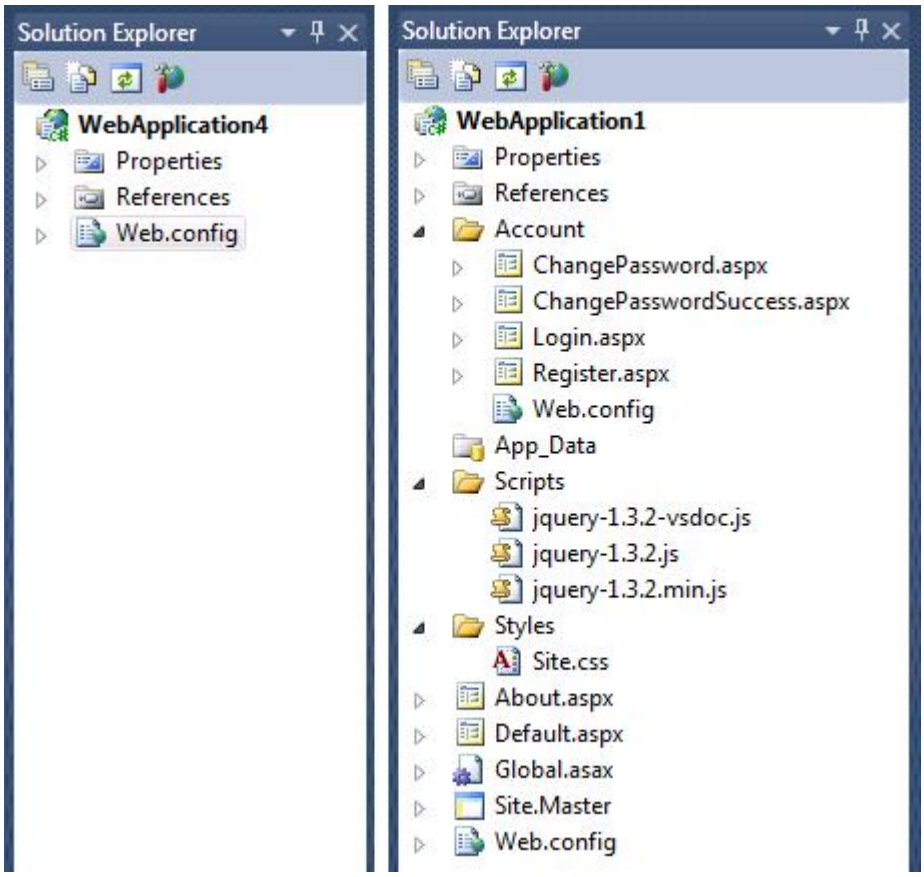
Avec Visual Studio 2008, la création d'un nouveau projet de type "Application Web" nous permettait d'obtenir les fichiers suivants :



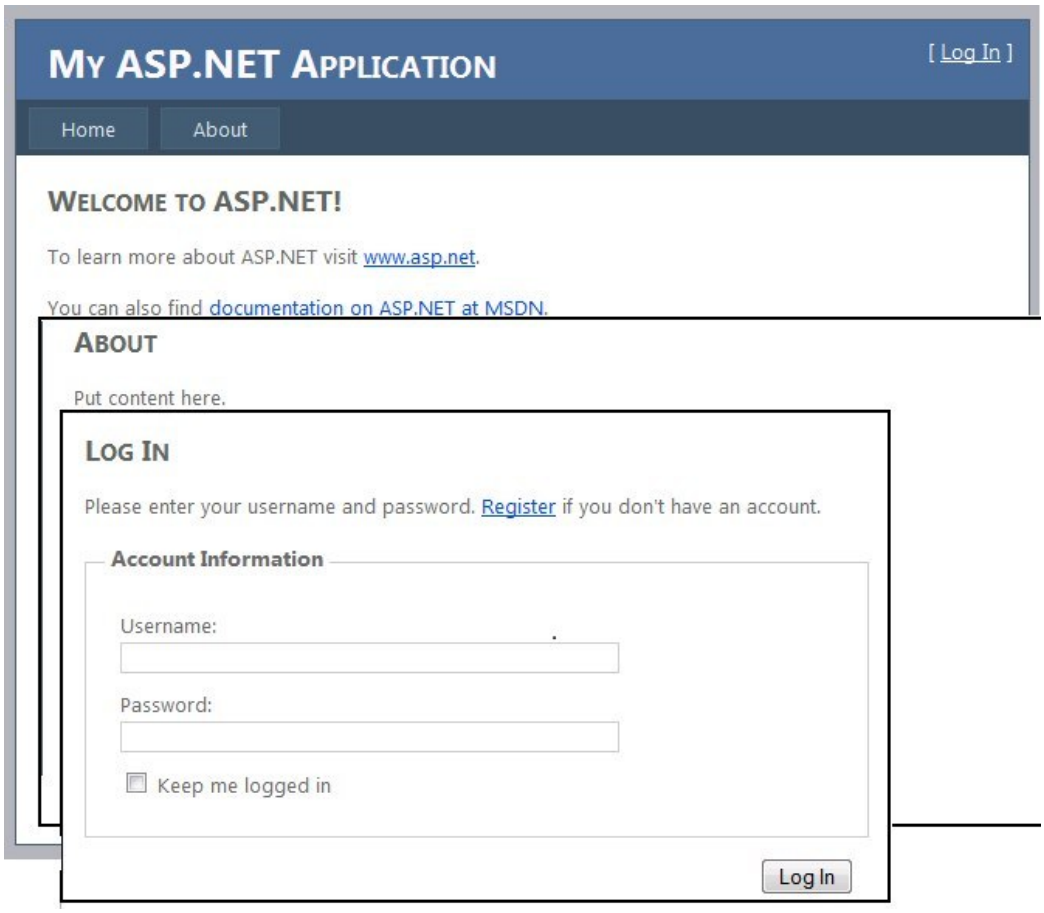
Or, si cette façon de commencer un projet suffisait dans la majorité des cas, elle ne pouvait pas satisfaire l'ensemble des développeurs Web.
En effet, si certains préfèrent partir d'un projet minimaliste, de nombreux développeurs veulent commencer avec un "squelette" d'application sur laquelle bâtir de nouvelles fonctionnalités.
C'est avec ces différentes façons de commencer un projet qu'ont été pensés les modèles de projet de Visual Studio 2010. Désormais, lorsque l'on démarre un nouveau projet visant le framework 4, il nous est proposé deux types de projet Web, à savoir une application Web standard *(ASP.NET Web Application)*, ou une application web vide *(Empty ASP.NET Web Application)*.



Si on crée un projet pour chacun de ces types d'application, on voit tout de suite les différences entre les deux.

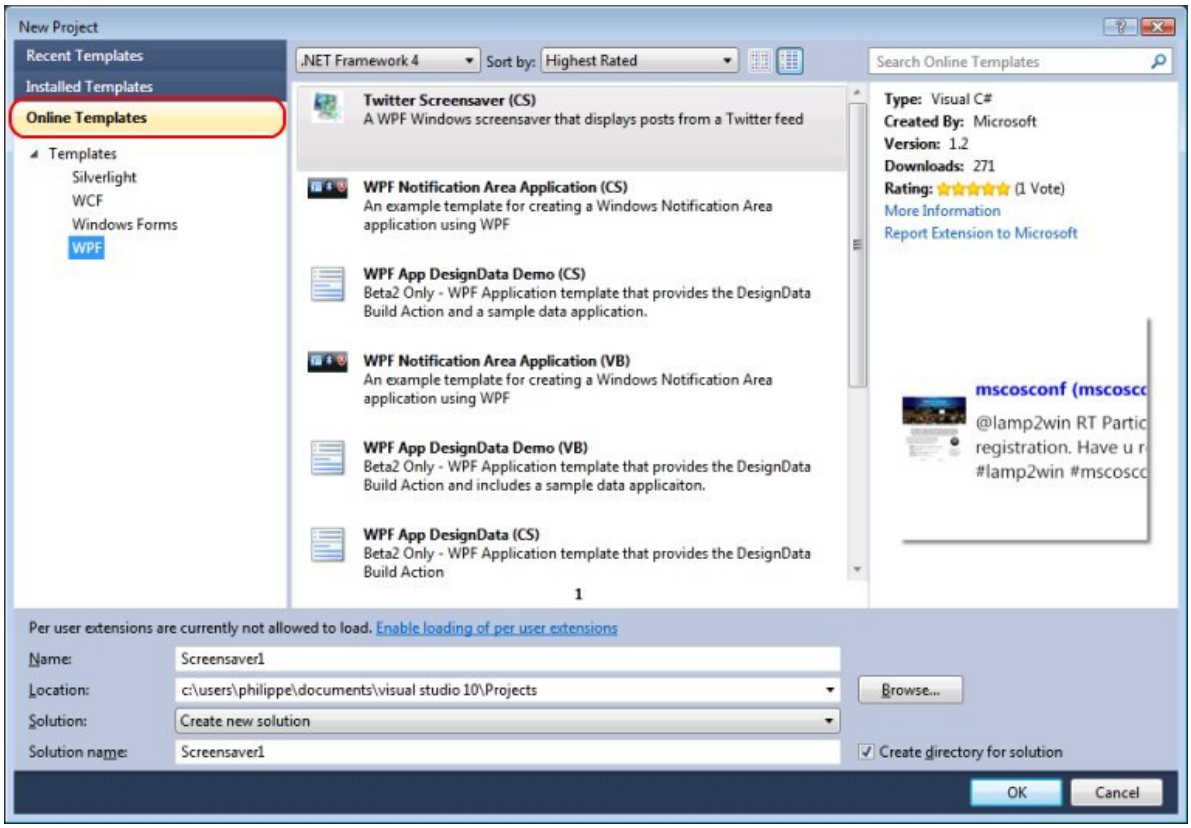


Si, dans le premier cas, on ne dispose même pas d'une page web vide, dans le second cas, on dispose d'une application déjà fonctionnelle, incluant JQuery, une page maître, des feuilles de style css, la configuration d'une authentification et un ensemble de pages pour gérer les comptes utilisateur.



Tout l'affichage du site est géré en CSS, et bénéficie des nouvelles fonctionnalités d'ASP.NET 4.

Il est de plus désormais possible d'accéder à des modèles en ligne, directement depuis l'interface de création de projets de Visual Studio.



III - Gestion personnalisée du ClientId

Historiquement, pour pouvoir assurer l'unicité des ID côté client, le Framework Webforms les redéfinissait, en les préfixant par l'ID des conteneurs parents si il en existait. Par exemple, imaginons qu'on définisse un contrôle de type TextBox dans le code aspx d'une page utilisant une page maître, de la façon suivante :

```
<asp:TextBox ID="maTextBox" runat="server" />
```

On pourrait s'attendre, coté client, à trouver ce code HTML :

```
<input name="maTextBox" type="text" id="maTextBox">
```

Et en fait, si le contrôle est situé directement dans la page, sans conteneur autour, ce sera le cas. Mais dès qu'on va utiliser un quelconque conteneur, le code aura tendance à ressembler à ceci :

```
<input name="ctl00$ContentPlaceHolder1$maTextBox" type="text" id="ContentPlaceHolder1_maTextBox" />
```

Tout développeur qui à eu à faire du développement web avec du javascript à été confronté un jour ou l'autre à cette situation. Pour la contourner, la solution mise en oeuvre de façon générale est d'utiliser la propriété ClientID des contrôles, contenant la valeur de l'ID côté client, que ce soit dans le code-behind ou dans le javascript. Par exemple, pour afficher le contenu de notre Textbox et être sur d'obtenir le contrôle que l'on veut, on va généralement écrire le code suivant :

```
<script type="text/javascript">
alert('<%= maTextBox.ClientID %>');
</script>
```

Avec la version 4 du Framework, il est désormais possible, pour chaque composant héritant de la classe Control (incluant donc les contrôles Page et MasterPage) d'utiliser la propriété ClientIDMode pour déterminer la façon dont sera géré le ClientID.

Les valeurs existantes pour cette propriété sont les suivantes :

III-A - AutoID

En mode AutoID, les contrôles se comportent exactement de la même façon en ASP.NET V4 que dans les versions précédentes

En conséquence, notre textbox aura toujours le même rendu coté client :

```
<input name="ctl00$ContentPlaceHolder1$maTextBox" type="text" id="ContentPlaceHolder1_maTextBox" />
```

III-B - Inherit

Ce mode est le mode par défaut. Il se base sur la valeur donnée au parent pour déterminer la valeur du contrôle. Il est à noter que si le ClientIDMode d'aucun parent n'est défini, les contrôles se comporteront comme s'ils étaient en mode Legacy.

III-C - Predictable

Le mode prédictible est surtout utile lorsque l'on veut définir une façon simple de retrouver des ensembles de contrôle côté client, et est donc particulièrement adapté à des GridView ou des Repeater. Une fois qu'un contrôle de l'arborescence est défini avec cette valeur de ClientIDMode, les descendants de ce contrôle auront comme ID la concaténation de tous les IDs de plus haut niveau, jusqu'au contrôle en question.

Par exemple, pour un Repeater défini ainsi :

```
<asp:Content ID="MainContent" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <asp:Repeater ID="Clients" runat="server">
        <ItemTemplate>
            Nom :
            <asp:Label ID="LblNom" runat="server"><%# Eval("Nom") %></asp:Label>
            Prénom :
            <asp:Label ID="LblPrenom" runat="server"><%# Eval("Prenom") %></asp:Label><br />
        </ItemTemplate>
    </asp:Repeater>
</asp:Content>
```

Le code html généré sera le suivant :

```
Nom :
<span id="ContentPlaceHolder1_Clients_LblNom_0">nom1</span>
Prénom :
<span id="ContentPlaceHolder1_Clients_LblPrenom_0">prenom1</span><br />

Nom :
<span id="ContentPlaceHolder1_Clients_LblNom_1">nom2</span>
Prénom :
<span id="ContentPlaceHolder1_Clients_LblPrenom_1">prenom2</span><br />
...
```

Comme on le voit, l'ID des contrôles peut être facilement prédit, quel que soit le nombre de contrôles générés. Pour bien voir la différence, voici ce que donne le même contrôle, avec la propriété ClientIDMode à AutoID :

```
Nom :
<span id="ctl00_ContentPlaceHolder1_Clients_ctl00_LblNom">nom1</span>
Prénom :
<span id="ctl00_ContentPlaceHolder1_Clients_ctl00_LblPrenom">prenom1</span><br />

Nom :
<span id="ctl00_ContentPlaceHolder1_Clients_ctl01_LblNom">nom2</span>
Prénom :
<span id="ctl00_ContentPlaceHolder1_Clients_ctl01_LblPrenom">prenom2</span><br />
...
```

Pour plus de souplesse, il est aussi possible de définir un suffixe pour les Id générés, par l'intermédiaire de la propriété RowClientIDSuffix, disponible pour les contrôles GridView et ListView. Cette propriété permet de définir une ou plusieurs clés qui seront recherchées dans les données du contrôle, de façon à le suffixer. Par exemple, si on remplace notre Repeater par la grille suivante :

```
<asp:GridView ID="Clients" ClientIDMode="Predictable" RowClientIDSuffix="Nom, Prenom" runat="server" AutoGenerate
    <Columns>
        <asp:TemplateField HeaderText="Nom">
            <ItemTemplate>
                <asp:Label ID="LblNom" runat="server" Text='<%# Eval("Nom") %>' />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Prenom">
            <ItemTemplate>
                <asp:Label ID="LblPrenom" runat="server" Text='<%# Eval("Prenom") %>' />
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

Le HTML généré sera le suivant, l'Id de chaque label étant suffixé par le nom et le prénom de l'employé

```
<table cellspacing="0" rules="all" border="1" id="ContentPlaceHolder1_Clients" style="border-
collapse:collapse;">
 <tr>
  <th scope="col">Nom</th><th scope="col">Prenom</th>
 </tr><tr>
  <td>
            <span id="ContentPlaceHolder1_Clients_LblNom_nom1_prenom1">nom1</span>

        </td><td>
            <span id="ContentPlaceHolder1_Clients_LblPrenom_nom1_prenom1">prenom1</span>
        </td>
 </tr><tr>
  <td>
            <span id="ContentPlaceHolder1_Clients_LblNom_nom2_prenom2">nom2</span>
        </td><td>
            <span id="ContentPlaceHolder1_Clients_LblPrenom_nom2_prenom2">prenom2</span>

        </td>
......
```

III-D - Static

Cette option est certainement la plus naturelle de toutes, à savoir qu'elle laisse le développeur totalement libre de la gestion des collisions entre Id. Dans notre exemple de TextBox, le HTML généré, si on affecte la valeur Static à la propriété ClientIDMode sera désormais le suivant :

```
<input name="ctl00$ContentPlaceHolder1$maTextBox" type="text" id="maTextBox" />
```

Le mode Static enlevant toute modification automatique des ID, il augmente la responsabilité des développeurs. N'allez pas utiliser Static dans des Contrôles Utilisateurs qui peuvent potentiellement être instanciés plusieurs fois dans une même page...

IV - Support natif du routage

Le routage est une fonctionnalité qui permet de modifier le modèle de navigation d'un site web, de façon à ce que le site puisse gérer des URL qui ne correspondent pas directement aux fichiers physiques composant l'application, mais des URL sémantiquement significatives pour les utilisateurs. De cette façon, les adresses des différentes pages du site web sont plus simples à retenir, à comprendre, et peuvent aider à l'optimisation du moteur de recherche (SEO).

Un exemple simple est le suivant : imaginons que l'on ait une application de gestion de contacts. Avec une application n'utilisant pas le routage, l'URL de la page affichant les prospects aura probablement l'aspect suivant :

```
http://monApplicationDeContacts/Contacts.aspx?type=2
```

A contrario, en utilisant le mécanisme de routage, on pourra accéder à la même page avec l'URL suivante :

```
http://monApplicationDeContacts/Contacts/Prospects
```

Le routage n'est pas à 100% une nouveauté, vu qu'il était déjà présent dans le SP1 du Framework 3.5, mais la version 4 d'ASP.NET apporte son lot de nouveautés pour cette fonctionnalité. Voyons ce qui se rapporte à cette fonctionnalité

- La propriété RouteTable.Routes permets d'accéder et de définir les différentes routes définies dans l'application. Si, avant la v4, on devait utiliser la méthode Add, il est désormais possible d'appeler la méthode MapPageRoute, plus lisible.
- Le PageRouteHandler est un simple HttpHandler, utilisé pour définir des routes, et transférer des données aux pages vers lesquelles le traffic est re-routé
- Deux propriétés supplémentaires, HttpRequest.RequestContext et Page.RouteData. Ces propriétés permettent un accès plus facile aux informations extraites depuis l'URL par le mécanisme de routage.
- Deux ExpressionBuilder ont été ajoutés, un permettant de créer une URL correspondant à une route (sans toutefois le coder en dur), l'autre permettant d'extraire des données depuis le contexte de Routage

Pour définir la route décrite ci-dessus, il va me falloir, dans mon Global.asax, ajouter une nouvelle route à la table de Routes. On peut le faire des deux façons suivantes :

En VB.NET

```
RouteTable.Routes.Add("ContactsRoute", New
 Route("Contacts/{typeUser}", _
 New PageRouteHandler("~/Contacts.aspx")))
  ' ancienne version

RouteTable.Routes.MapPageRoute("ContactsRoute", "Co
{typeUser}", "~/
Contacts.aspx") ' nouvelle version, moins verbeuse
```

En C#

```
RouteTable.Routes.Add("ContactsRoute", new
 Route("Contacts/{typeContact}",
 new PageRouteHandler("~/Contacts.aspx")));
  /// ancienne version

RouteTable.Routes.MapPageRoute("ContactsRoute", "Contacts/
{typeContact}", "~/
Contacts.aspx"); /// nouvelle version, moins verbeuse
```

MapPageRoute possède un paramètre supplémentaire, checkPhysicalUrlAccess, qui permet de définir si les restrictions d'accès s'appliquent à la route ou pas. Plus clairement, cela veut dire que si la page Contacts.aspx est protégée par la configuration du Web.Config, et que l'on affecte false à checkPhysicalUrlAccess, la page Contacts.aspx sera inaccessible en direct, mais accessible par la Route.

Une fois la route définie, il faut pouvoir accéder aux données transférées à la page par la route. Pour cela, le plus simple est d'utiliser Page.RouteData (ou HttpRequest.RequestContext.RouteData, ce qui revient au même), de la façon suivante :

En VB.NET	En C#
`Dim typeContact as String = Page.RouteData.Values("typeUser");`	`string typeContact = Page.RouteData.Values["typeContact"] as string;`

Il est aussi possible d'accèder directement aux mêmes informations depuis le code de la page aspx, en utilisant l'ExpressionBuilder RouteValue :

```
<asp:Label ID="monLabel" runat="server" Text="<%$RouteValue:typeContact%>" />
```

Inversement, plutot que de créer un lien de façon manuelle vers l'URL d'une page routée, il est possible d'utiliser l'ExpressionBuilder RouteUrl :

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="<%$RouteUrl:typeUser=Prospects%>">Accéder à la liste des prospects</asp:HyperLink>
```

Dans ce cas précis, la Route sera générée automatiquement vers le bon URL, car une seule route est définie, et le moteur de routage reconnait de plus le paramètre typeUser, ce qui lui permets d'identifier notre route. Dans le cas ou plusieurs Routes avec le même nom de paramètre existent, on peut spécifier le nom de la Route à utiliser

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="<%$RouteUrl:RouteName=ContactsRoute,typeUser=Prospects%>">Accéder à la liste des prospects</asp:HyperLink>
```

V - Amélioration de la granularité du ViewState

Avant de décrire l'évolution en question, on va reparler brièvement du ViewState. Le ViewState est un mécanisme dont le but est de stocker l'état de chaque (ou presque) propriété des contrôles de la page, de façon à pouvoir restaurer l'état des contrôles sans avoir à les re-charger explicitement. Cette déclaration simplifie un peu le ViewState (qui possède d'autre finesses), mais elle est suffisante pour le contexte actuel.

Le gros défaut de ce mécanisme est que, TOUTES les propriétés d'une page étant stockées et sérialisées, il arrive régulièrement que le ViewState prenne un pourcentage important de la page, alla parfois jusqu'à 50% du code HTML rendu côté client, ce qui peut pénaliser les performances, surtout pour des données volumineuses.

Il est néanmoins possible de désactiver le viewstate, en affectant la valeur *false* à la propriété **EnableViewState** d'un contrôle, de la page entière ou du Web.Config.

Désactivation au niveau d'un contrôle

```
<asp:Label ID="monLabel" runat="server" Text="Mon Label" EnableViewState="false" />
```

Désactivation au niveau d'une pagee

```
<%@ Page enableViewState="False" %>
```

Désactivation au niveau du Web.Config

```
<configuration>
 <system.web>
  <pages EnableViewState="true" />
```

Désactivation au niveau du Web.Config

```
  </system.web>
</configuration>
```

Le problème, dans les version précédentes d'ASP.NET, était qu'une fois que le ViewState était désactivé au niveau de la page, il ne pouvait plus être ré-activé au niveau dees contrôles de la page. Si il était activé, il pouvait être désactivé séléctivement.
Au final, régler le ViewState revenait à désactiver le ViewState sur chaque contrôle que l'on ne voulait pas stocker dedans, un à un.

La version 4 d'ASP.NET à vu l'apparition d'une nouvelle propriété, nommée ViewStateMode, et présente sur tous les contrôles serveurs.

Cette propriété indique si le ViewState est actif pour le contrôle, inactif, ou si il hérite du parent. Il est désormais possible, avec cette propriété, de désactiver le ViewStateMode au niveau du parent, avec EnableViewState, et de l'activer séléctivement pour chaque contrôle.

Par exemple, le code suivant, en ASP.NET 4 :

```
<%@ Page Language="C#" AutoEventWireup="true"  EnableViewState="false" %>
<!doctype html public "-//w3c//dtd xhtml 1.0 transitional//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="monLabel1" runat="server" Text="Mon Label 1" ViewStateMode="Enabled" />
    <asp:Label ID="monLabel2" runat="server" Text="Mon Label 2"/>
    <asp:Label ID="monLabel3" runat="server" Text="Mon Label 3"/>
    <asp:Label ID="monLabel4" runat="server" Text="Mon Label 4"/>
    <asp:Label ID="monLabel5" runat="server" Text="Mon Label 5"/>
    <asp:Label ID="monLabel6" runat="server" Text="Mon Label 6"/>
    <asp:Label ID="monLabel7" runat="server" Text="Mon Label 7"/>
    <asp:Label ID="monLabel8" runat="server" Text="Mon Label 8"/>
    <asp:Label ID="monLabel9" runat="server" Text="Mon Label 9"/>
    <asp:Label ID="monLabel10" runat="server" Text="Mon Label 10"/>
    </div>
    </form>
</body>
</html>
```

Produira sensiblement le même ViewState que le code suivant :

```
<%@ Page Language="C#" AutoEventWireup="true" EnableViewState="true" %>
<!doctype html public "-//w3c//dtd xhtml 1.0 transitional//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:Label ID="monLabel1" runat="server" Text="Mon Label 1" />
    <asp:Label ID="monLabel2" runat="server" Text="Mon Label 2" EnableViewState="false"/>
    <asp:Label ID="monLabel3" runat="server" Text="Mon Label 3" EnableViewState="false"/>
    <asp:Label ID="monLabel4" runat="server" Text="Mon Label 4" EnableViewState="false"/>
    <asp:Label ID="monLabel5" runat="server" Text="Mon Label 5" EnableViewState="false"/>
    <asp:Label ID="monLabel6" runat="server" Text="Mon Label 6" EnableViewState="false"/>
    <asp:Label ID="monLabel7" runat="server" Text="Mon Label 7" EnableViewState="false"/>
    <asp:Label ID="monLabel8" runat="server" Text="Mon Label 8" EnableViewState="false"/>
```

```
    <asp:Label ID="monLabel9" runat="server" Text="Mon Label 9" EnableViewState="false"/>
    <asp:Label ID="monLabel10" runat="server" Text="Mon Label 10" EnableViewState="false"/>
    </div>
    </form>
</body>
</html>
```

```
<!doctype html public "-//w3c//dtd xhtml 1.0 transitional//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>

</title></head>
<body>
    <form method="post" action="WebForm2.aspx" id="form1">
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKLTM5OTM0NTc0MGRkQmrSfKJDzc9VS4HprTBGapKgXj6r+zX4uXldWvcrARI=" />
```

```
<!doctype html public "-//w3c//dtd xhtml 1.0 transitional//en" "http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>

</title></head>
<body>
    <form method="post" action="WebForm1.aspx" id="form1">
<div class="aspNetHidden">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKLTM5OTM0NTc0MGRkdr4Ug6/ywO5SaKZnpWDHYt5pT91sFql9jLVRP2E61C4=" />
```

VI - Améliorations des contrôles standards

Chaque version d'ASP.NET apporte son lot de nouveaux composants, mais également l'amélioration des contrôles existants. Cette fois-ci, l'accent a été mis sur la personnalisation de ces derniers ainsi que sur le respect des standards Web.

VI-A - Amélioration du contrôle ListView

Le contrôle ListView qui était apparu avec ASP.NET pour le plus grand bonheur des développeurs a été simplifié pour ne plus requérir la déclaration d'un template de mise en forme.

Dorénavant, une simple déclaration des éléments à afficher, via l'élément ItemTemplate est suffisant :

```
<asp:ListView ID="ListView1" runat="server">
  <ItemTemplate>
    <% Eval("Nom")%>
  </ItemTemplate>
</asp:ListView>
```

VI-B - Amélioration des contrôles CheckBoxList et RadioButtonList

Ici, une simple amélioration de la propriété RepeatLayout qui permet dorénavant d'ajouter les valeurs OrderedList et UnorderedList, deux valeurs destinés à obtenir un rendu utilisant les balises listes XHTML <OL> et <UL>. Cette petite modification permet entre autres, par rapport à la génaration de tableaux (<table>), une meilleure personnalisation CSS et un meilleur respect des standards d'accessibilité.

VI-C - Amélioration du contrôle Menu

Je ne sais pas pour vous, mais il y a bien longtemps que je n'utilise plus le contrôle Menu car celui-ci, bien que pouvant se générer seul à partir d'un fichier Web.Sitemap, générait une série de tableaux imbriqués, rendant la personnalisation difficile et n'était absolument pas respectueux des standards d'accessibilité (WAI). J'utilise donc un composant personnalisé de ma composition, permettant de mixer le fonctionnement du Menu de base, à Jquery mais également en respectant les standards Web en générant des listes, plutot que des tableaux.

La nouvelle version du contrôle corrige cela et génère une liste non ordonnée d'éléments :

Code HTML généré

```
<div id="Menu1">
  <ul>
    <li><a href="#">Accueil</a></li>
    <li><a href="#">Page1</a></li>
  </ul>
</div>
```

Mieux encore, l'accessibilité a été fortement améliorée car, lorsque le menu a le focus, il est possible de naviguer au sein de celui-ci à l'aide des touches du clavier (ARIA).

Enfin, il est possible de totalement paramétrer le rendu CSS du menu, en désactivant la génération CSS automatique en mode block, en mettant à faux, la propriété *IncludeStyleBlock*.

Pour récupérer facilement la feuille de style à éditer, placez un composant Menu avec avec la propriété IncludeStyleBlock réglée sur true, puis via le smarttag, cliquez sur Auto Format et choisissez un thème puis compilez votre projet pour afficher la page. Regardez alors la source pour extraire le CSS généré, puis copiez le dans une feuille CSS bien à vous. Revenez dans Visual Studio, et réglez la propriété IncludeStyleBlock sur false. Il ne vous reste plus qu'à personnaliser le CSS récupéré :

CSS généré

```
<style type="text/css">
 #MainContent_Menu1 { background-color:#F7F6F3; }
 #MainContent_Menu1 img.icon { border-style:none;vertical-align:middle; }
 #MainContent_Menu1 img.separator { border-style:none;display:block; }
 #MainContent_Menu1 ul { list-style:none;margin:0;padding:0;width:auto; }
 #MainContent_Menu1 ul.dynamic { background-color:#F7F6F3;z-index:1;margin-left:2px; }
 #MainContent_Menu1 a { color:#7C6F57;font-family:Verdana;font-size:0.8em;
      text-decoration:none;
      white-space:nowrap;
      display:block; }
 #MainContent_Menu1 a.static { padding:2px 5px 2px 5px;text-decoration:none; }
 #MainContent_Menu1 a.popout {
   background:url("/WebResource.axd?d=ZpQ_VdlsJ4NsCH6tI9Ff5pqkbJux-
tl9iyjWiOQrwWU1&amp;t=633927035656763451")
       no-repeat right center;padding-right:14px; }
 #MainContent_Menu1 a.dynamic { padding:2px 5px 2px 5px;text-decoration:none; }
 #MainContent_Menu1 a.static.selected { background-color:#5D7B9D;text-decoration:none; }
 #MainContent_Menu1 a.dynamic.selected { background-color:#5D7B9D;text-decoration:none; }
 #MainContent_Menu1 a.static.highlighted { color:White;background-color:#7C6F57; }
 #MainContent_Menu1 a.dynamic.highlighted { color:White;background-color:#7C6F57; }
</style>
```

VI-D - Contrôles templates

Toujours dans l'amélioration et la personnalisation du rendu des contrôles existants, différents contrôles ont vu l'apparition de la propriété RenderOuterTable. Cette propriété, qui est maintenant disponible pour les composants :
- ChangePassword
- CreateUserWizard
- FormView
- Login
- PasswordRecovery
- Wizard

permet de désactiver la génération d'un tableau extérieur englobant le rendu HTML du contrôle. En effet, par défaut, le simple fait de positionner l'un de ces contrôles entrainait la génération d'un tableau dans lequel était placé le contrôle, afin de pouvoir positionner du style inline. La finalité finale est bien entendu d'alléger le code HTML généré, et surtout d'avoir un meilleur contrôle du HTML final, afin de pouvoir le personnaliser finement à l'aide de feuilles de style CSS.

VII - Gestion des déploiements Web

Le déploiement d'une application Web est une action sensible qui peut être complexe, longue et fastidieuse. Heureusement pour nous, développeurs ou responsables de déploiement, cette étape est devenue plus aisée grâce aux différentes améliorations qu'apporte Visual Studio 2010.

VII-A - Web packaging

Le déploiement passe dorénavant par un package Web, un fichier zip auto-documenté qui permet de contenir les sources à déployer, mais également les paramètres IIS à appliquer, les schémas de base de données et les scripts à déployer, les certificats, les paramètres registres, les composants à placer dans le GAC, etc.
L'intérêt du Web package provient aussi de sa facilité d'utilisation. Il suffit en effet de le copier sur le serveur Web où l'on souhaite déployer puis, via la console de gestion de IIS, on charge le fichier et c'est tout.

Il est également possible de déployer le package web via des scripts en ligne de commande pour automatiser cette tâche.

VII-B - Web.config transformation

Petite merveille que nous attendions tous, la possibilité d'avoir des fichiers de configuration dynamiques permettant de passer rapidement d'une configuration de développement, à celle de test ou encore celle de production. Avec Visual Studio, cela se fait avec XML Document Transform (XDT), fonctionnalité qui permet de générer un fichier de configuration Web.config à partir d'autres fichiers de configuration.
Certains avaient peut-être l'habitude de passer par Nant ou par un module additionnel de Visual Studio, c'est maintenant chose incluse.

Au niveau fonctionnement, c'est extrêmement simple, vous placez vos différentes configurations au sein de fichiers de configuration complémentaires (ex: web.Debug.config, web.Release.config, etc, sans limite), puis, au moment de la compilation, en fonction du profil de configuration choisi, Visual Studio ira piocher les éléments au sein de ces fichiers complémentaires pour les placer dans le Web.config qui sera exporté et déployé.

Par exemple, supposons un fichier de configuration web.debug.config qui contiendrait juste un bout de configuration : la chaine de connexion

web.debug.config

```
<connectionStrings xdt:Transform="Replace">
  <add name="connexionBase" connectionString="blabla connexion à mon serveur" />
</connectionStrings>
```

Notez l'attribut xdt:Transform="Replace". Cela signifie qu'à la compilation, l'élément connectionStrings du fichier web.config sera automatiquement remplacé par celui-là.

Pour un exemple pas-à-pas sur l'utilisation de XDT, je vous encourage à lire **l'article suivant**.

VII-C - Publication 1-click

Visual Studio propose désormais de s'interfacer directement avec les outils de gestion distante d'IIS, afin de déployer le site web sur un serveur distant de façon automatique.
En pratique, cette fonctionnalité ressemble à la fonctionnalité de publication existante depuis quelques versions, mais sous le capot, elle prépare en fait une vraie migration, incluant toutes les dépendances nécessaires.
Il est de plus possible de publier sur plusieurs serveurs de façon simultanée, avec toutefois la limite de 50 profils de déploiement dans un projet...soit 50 serveurs publiés simultanément...

VIII - Autres améliorations

VIII-A - Integration du contrôle ASP.NET Chart

Parmi les autres nouveautés, même si c'en est pas tout à fait une, se trouve l'ajout au Framework 4, du composant ASP.NET Chart Control, composant de graphes, apparu au sein du SP1 du Framework 3.5. Ce composant destiné à dessiner des graphiques permet de rendre vos applications plus parlantes, et de créer des tableaux de reporting professionnels. Si beaucoup d'entre vous étiez habitués à utiliser des composants tiers (ZedGraph, Aspose Chart, Infragistics, et bien d'autres), sachez que vous avez maintenant à disposition, un composant complet, simple à utiliser,facilement personnalisable et proposant un grand nombre de types de diagrammes disponibles.

Aucun paramétrage particulier n'est requis puisqu'il est possible de changer le type de diagramme avec une seule propriété.

Voici d'ailleurs la liste exhaustive des 35 types

Type de diagramme	Description
100% Stacked Area	Affiche plusieurs séries de données sous forme de zones empilées
100% Stacked Bar	Affiche plusieurs séries de données sous forme de barres empilées
100% Stacked Column	Affiche plusieurs séries de données sous forme de colonnes empilées
Area	Emphase le dégré de changement sur la durée et montre la relation des différentes parties dans un ensemble
Bar	Illustre la comparaison parmi des éléments individuels
Box plot	Consiste en un ou plusieurs symboles de type boite qui résument la distribution des données dans un ou plusieurs ensembles
Bubble	Une variation du diagramme à point où les points sont remplacés par des bulles de différentes tailles
Candlesticks	Utilisé pour
Column	Un histogramme qui affiche une séquence de colonnes pour comparer des valeurs parmi plusieurs catégories
Doughnut	Similaire au diagramme camembert (Pie) mais avec un trou au milieu
Error bar	Consiste en des lignes avec des marqueurs qui sont utilisés pour montrer des information statistiques
FastLine	Une variation du diagramme Line qui réduit de façon conséquente le temps d'affichage lorsque les séries contiennent un grand nombre de points
FastPoint	Une variation du diagramme Point qui réduit de façon conséquente le temps d'affichage

	lorsque les séries contiennent un grand nombre de points
Funnel	Affiche une forme d'entonnoir dont la valeur totale vaut 100%
Kagi	Affiche une série de lignes verticales connectées où l'épaisseur et la direction des lignes sont dépendent de l'action de la valeur du prix
Line	Illustre les évolutions des données sur la durée
Pie	Diagramme camembert qui affiche les catégories sous forme visuelle de proportion
Point	Utilise des points pour représenter des points de données
Point and figure	Oubli la durée et affiche seulement les changements de prix
Polar	Un diagramme circulaire sur lequel les points de données sont affichés utilisant l'angle et la distance par rapport au point central
Pyramid	Affiche les données sous forme de pyramide. Le total vaut toujours 100%
Radar	Un diagramme circulaire qui est utilisé principalement comme outil de comparaison de données
Range	Affiche une plage de données en positionnant deux valeurs Y par point, avec chaque valeur Y dessinée sous forme de diagramme Line
Range bar	Affiche différents événements qui ont des valeurs de début et de fin
Range column	Affiche une plage de données en positionnant 2 valeurs Y par point
Renko	Affiche une série de lignes verticales connectées où l'épaisseur et la direction des lignes sont dépendent de l'action de la valeur du prix
Spline	Un diagramme Line qui positionne une courbe entre chaque point
Spline area	Un diagramme Area qui positionne une courbe entre chaque point
Spline range	Affiche une plage de données en positionnant deux valeurs Y par point de donnée, où chaque valeur Y est dessinée sous forme d'une diagramme Line
Stacked area	Un diagramme Aera qui empile deux séries (ou plus) l'une sur l'autre
Stacked bar	Affiche des séries du même diagramme mais sous forme de barres empilées
Stacked Column	Affiche des séries du même diagramme mais sous forme de colonnes empilées. Est utilisé

	pour comparer la contribution de chaque valeur sur un total.
Step line	Similaire au diagramme Line but en utilisant des lignes verticales et horizontales pour connecter les différents points
Stock	Affiche différents points de prix significatifs en incluant les points d'ouverture, de fermeture, le plus haut et le plus bas
Three line break	Affiche une série de boites verticales, ou des lignes, qui reflètent les changements de valeur de prix

Si vous voulez une documentation complète sur l'utilisation du contrôle Chart, téléchargez le **document suivant**.

VIII-B - Gestion des balises Meta

Lorsque ASP.NET 2.0 apporta les MasterPages, les développeurs trouvèrent un moyen simple de partager des metatags (le **Dublin Core**) au sein de toutes les pages de leurs sites Web. Néanmoins, il perdait par la même occasion de rendre facilement ces metatags dynamiques pour certaines pages et dépendant du contenu de la page. ASP.NET apporte une solution à ce problème en autorisant la modification en code-behind, de ces métatags.

Ainsi, avec une simple ligne de code, vous pourrez éditer leur contenu :

```
private string _variable = "developpez";
protected void Page_Load(object sender, EventArgs e)
{
    this.MetaDescription = String.Format("Description dynamique {0}", _variable);
    this.MetaKeywords = "developpez, tutoriel, microsoft";
}
```

Et vous obtiendrez deux belles balises :

```
<meta name="description" content="Description dynamique developpez" />
<meta name="keywords" content="developpez, tutoriel, microsoft" />
```

VIII-C - Simplification du Web.Config

Si le fichier Web.Config est une chose merveilleuse pour centraliser la sécurité, la configuration et le comportement de vos sites Web, c'est aussi, au fil des fonctionnalités ajoutées à ASP.NET, devenu un vrai foutoir. Il n'est pas rare de rencontrer des Web.Config de plusieurs centaines de lignes et ces fichiers deviennent difficiles à maintenir.
Avec ASP.NET 4, le Web.Config a subi un régime forcé et ainsi, lorsque vous créez un projet ASP.NET vide, la seule chose qu'il contient est la version du Framework ciblée :

```
<?xml version="1.0"?>
<configuration>

    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
    <system.webServer>
      <modules runAllManagedModulesForAllRequests="true"/>
    </system.webServer>

</configuration>
```

Ceci est notamment possible grâce au déplacement des paramètres par défaut au niveau du machine.config, permettant d'appliquer des paramètres à tous les sites Webs de la machine. Ce découpage, plus logique, vous permet de faire évoluer des fchiers Web.config qui ne contiennent que ce qui est propre à leur application, tout en enlevant le superflu.

VIII-D - Effectuer un filtre grâce au contrôle QueryExtender

La façon de retourner un set de données a beaucoup évolué ces dernières années, en imposant peu à peu l'utilisation d'ORM, puis via le langage de requêtage objet Linq. Ainsi, la façon de filtrer des données Linq ou Entity est devenue extrêmement simple grâce à ces requêtes objet, mais ne répond pas pourtant à tous les besoins courants des sites Web.
Prenons un exemple simple, une grille affichant la liste des utilisateurs et une textbox permettant de faire une recherche sur ces derniers. Pour le moment, rien de complexe puisqu'il suffit de créer une fonction qui génère une requête filtrée. Maintenant, imaginions que nous pouvions nous passer de ce code superflu tout en gardant cette possibilité de filtre. Ne serait-il pas merveilleux? Et bien c'est maintenant possible grâce au contrôle QueryExtender.

Ce contrôle permet de filtrer une source de données avant que celle-ci ne retourne ses données aux autres composants. Dans notre exemple précédant, nous allons par exemple pouvoir filtrer en live les données.

```
<asp:LinqDataSource ID="maSource" runat="server"> TableName="Utilisateurs">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="maSource" runat="server">
  <asp:SearchExpression DataFields="Nom"
      SearchType="StartsWith">
    <asp:ControlParameter ControlID="txtSearch" />
  </asp:SearchExpression>
</asp:QueryExtender>
```

Aussi simple que cela, les données seront filtrées en fonction du texte saisi dans la textBox.

Il est nécessaire que la source de données implémente IQueryableDataSource, sinon QueryExtender ne marchera pas et une exception sera lancée à l'exécution.

Le QueryExtender va beaucoup plus loin car il est aussi capable de retourner des éléments en fonction de certaines propriétés, par exemple un checkBox indiquant de ne retourner que les utilisateurs mâles. Il s'agit ici d'utiliser non plus une SearchExpression mais une PropertyExpression qui, lorsque la comparaison sera vraiment, retournera l'élément traité.

```
<asp:LinqDataSource ID="maSource" runat="server"> TableName="Utilisateurs">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="maSource" runat="server">
 <asp:PropertyExpression>
    <asp:ControlParameter ControlID="ckbOnlyMen" Name="IsAMan" />
  </asp:PropertyExpression>
</asp:QueryExtender>
```

Enfin, pour finir, et pour aller encore plus loin, il est possible d'utiliser une CustomExpression qui utilisera une méthode écrite par le développeur.

```
<asp:LinqDataSource ID="maSource" runat="server"> TableName="Utilisateurs">
</asp:LinqDataSource>
<asp:QueryExtender TargetControlID="maSource" runat="server">
  <asp:CustomExpression OnQuerying="maMethode" />
</asp:QueryExtender>
```

Qui appelera automatiquement ma méthode :

```
public void maMethode(object sender, CustomExpressionEventArgs e)
{
  e.Query = from u in e.Query.Cast<Utilisateur>()
      where u.Adresse.Ville = "Courbevoie"
      select u;
}
```

Il est tout à fait possible de cumuler plusieurs expressions au sein d'un seul et même contrôle QueryExtender.

VIII-E - Conserver la sélection d'une ligne dans une GridView ou une ListView

Une propriété supplémentaire a fait son apparition au sein des contrôles GridView et ListView: la propriété EnablePersistedSelection. Cette propriété permet à vos contrôles de mémoriser les lignes qui étaient sélectionnées lorsqu'un utilisateur quitte la page puis y revient ensuite.
L'exemple typique serait une GridView avec de la pagination, lorsque vous cochez un élément puis passez à la page suivante, lorsque vous revenez à la première page, votre élément se sélectionne automatiquement.

VIII-F - Améliorations du Session State

A l'heure de l'écriture de cet article, il n'existe que deux moyens de stoquer le Session State, soit dans un serveur SQL grâce à un provider dédié, soit sur un serveur distant dédié à la tâche de stockage. Ces deux cas de stockage sont notamment utilisés dans le cas de fermes serveurs.
Au niveau du fonctionnement, le Session State est sérialisé puis envoyé au provider qui le stocke. Cette sérialisation a un coût, notamment en quantité de données transmises.

ASP.NET 4 apporte alors la possibilité de compresser (et décompresser) ce flux de données à l'aide de classe System.IO.Compression.GZipStream. Pour cela, il suffit simplement d'activer la compression dans le Web.Config à l'aide de l'attribut CompressionEnabled :

```
<sessionState
  mode="SqlServer"
  sqlConnectionString="blabla"
  allowCustomSqlDatabase="true"
  compressionEnabled="true"
/>
```

VIII-G - Redirections permanentes

Il arrivera sûrement un jour où vous souhaiterez déplacer une page ou réorganiser votre site Web sans vouloir que les anciennes pages donne une page d'erreur manquante (erreur 404). Que cela soit pour l'utilisateur ou au niveau du référencement sur les moteurs de recherche, il est important d'utiliser les codes d'erreur de convention et indiquer dans la réponse renvoyée au navigateur client, que la page a été déplacée et d'indiquer le nouvel emplacement. En Web, cela se fait via le code d'erreur **HTTP 301**.

Maintenant, plutôt que de générer à la main la réponse HTTP à retourner, il vous est possible d'utiliser la méthode RedirectPermanent qui se charge de faire tout le travail à votre place.

```
this.Response.RedirectPermanent("nouvellepage.aspx", true);
```

VIII-H - Encodage HTML automatique du code en ligne

Pour finir, parlons maintenant des expressions encodées automatiquement dans le code in-line. Par exemple un simple :

```
<%=monObjet.Nom %>
```

Qui nécessite, pour éviter un rendu incorrect ou de l'injection volontaire/involontaire de code, d'encoder le texte avant de le retourner au client.

```
<%= HttpUtility.HtmlEncode(monObjet.Nom) %>
```

Avec ASP.NET, pour arriver à la même chose, il vous suffit d'utiliser la nouvelle syntaxe (notez le deux points en lieu et place du symbole égal) :

```
<%: monObjet.Nom %>
```

Jusque-là, les deux derniers bouts de code fournissent exactement le même résultat. Il devient donc une bonne habitude d'utiliser **exclusivement** cette nouvelle syntaxe.
Malheureusement, un problème survient, si nous utilisons la nouvelle syntaxe et que le contenu est encodé, comment faire lorsque l'objet fourni est déjà encodé? Un double encodage produirait un résultat peu réussi. La solution numéro 1, consisterait à repasser à l'ancienne syntaxe (signe égal). Mais pour éviter de jongler entre les deux, et se poser la question à chaque fois, de savoir si oui ou non, il faut utiliser la nouvelle syntaxe, ASP.NET apporte l'interface IHtmlString qui permet de préciser que la chaine est déjà encodée. Elle ne sera ainsi par réencodée, malgré l'utilisation de la nouvelle syntaxe. Exemple :

```
<%: new HtmlString("<b>Hello world</b>") %>
```