Présentation d'Asp.net 3.5 Dynamic Data

par Ludovic Lefort (Site web) (Blog)

Dernière mise à jour :

A l'aide d'exemples simples je vais vous présenter une des nouveautés d'Asp.net 3.5 : Dynamic Data

0 - Introduction

Dynamic Data est une nouvelle technologie inclue dans l' "Asp.net 3.5 Extension CTP ".
Elle fournit une architecture permettant de construire des applications orientées données.
Il est donc possible d'afficher dynamiquement des pages basées sur le schéma d'une base de données.
Asp.net Dynamic Data fournit de base un Template de page pour afficher les données.
Bien que cet article se base sur la version de Dynamic Data présente dans l'"Asp.net 3.5 Extension CTP". Le projet continue à se développer sous forme de preview téléchargeables sur le site de Microsoft.

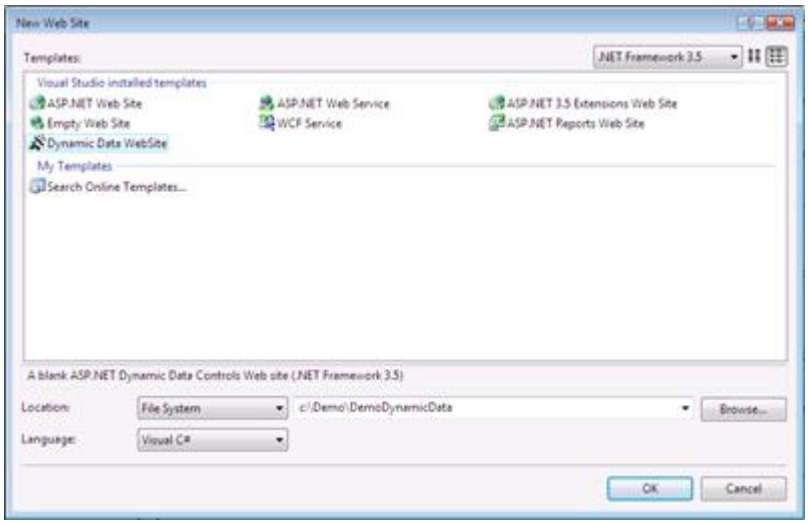
1 - Pré-requis pour cet exemple

- Microsoft Visual Studio 2008 (**Virtual pc**)
- SQL Server 2005 (ou Sql 2005 Express) (**http://www.microsoft.com/downloads/details.aspx?familyid=220549b5-0b07-4448-8848-dcc397514b41&displaylang=en**
- Asp.net 3.5 Extensions CTP (**http://www.microsoft.com/downloads/details.aspx?familyid=A9C6BC06-B894-4B11-8300-35BD2F8FC908&displaylang=en**
- La base de données Northwind (**http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en**)

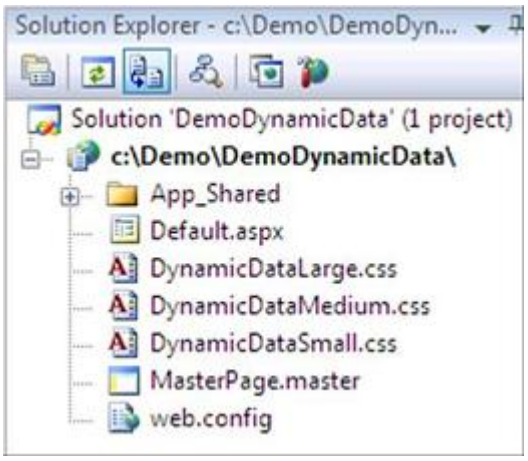
UPDATE : Les extensions sont maintenant inclues dans le SP1 de Visual Studio 2008 et du framework .net 3.5

2 - Création d'un premier projet Dynamic Data

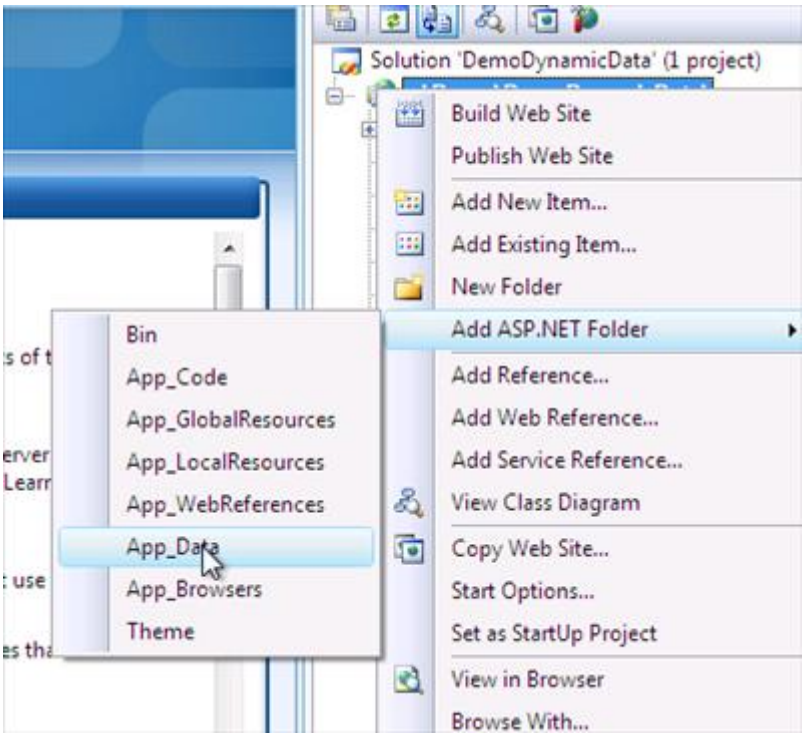
Nous allons débuter par la création d'un nouveau site web dans Visual Studio : File > New Web Site



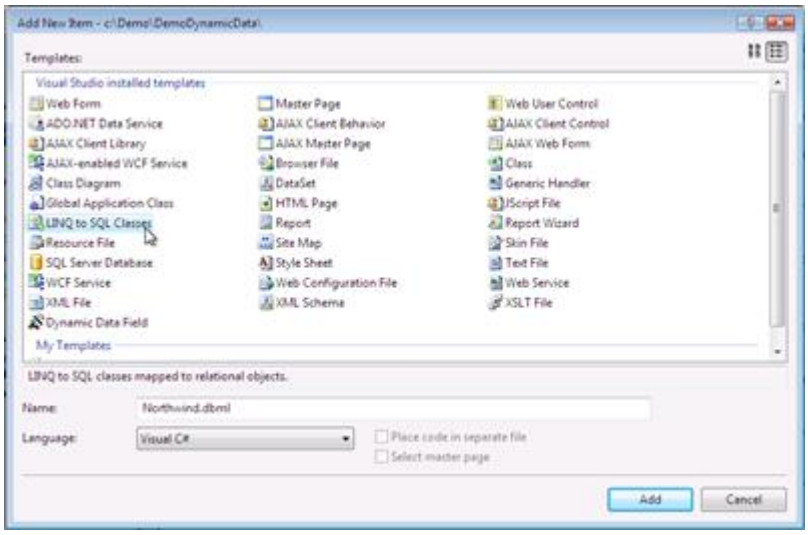
Choisissez le template **Dynamic Data WebSite**. Si ce template n'est pas présent c'est que vous n'avez pas installé les extensions pour Asp.net 3.5.
Je vais nommer mon projet **DemoDynamicData**
Visual Studio va automatiquement créer la structure de votre projet



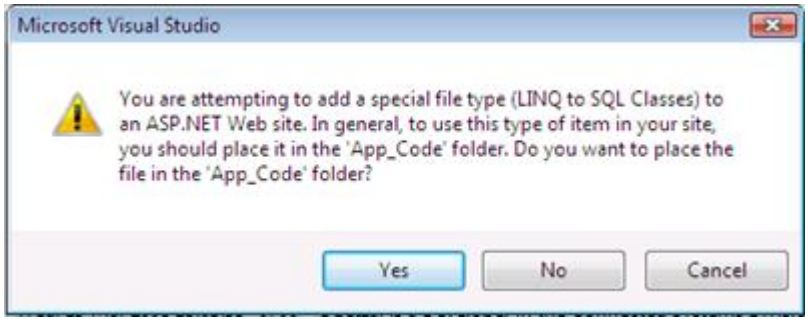
Dynamic Data permet la création de site basé sur une source de données, il nous faut donc lui en fournir une. Dans cet exemple, une fois n'est pas coutume, nous allons utiliser la base de données Northwind de Microsoft. Ajoutons un répertoire **App_Data** à notre site : clic droit sur la racine du site > Add Asp.net Folder > APP_Data



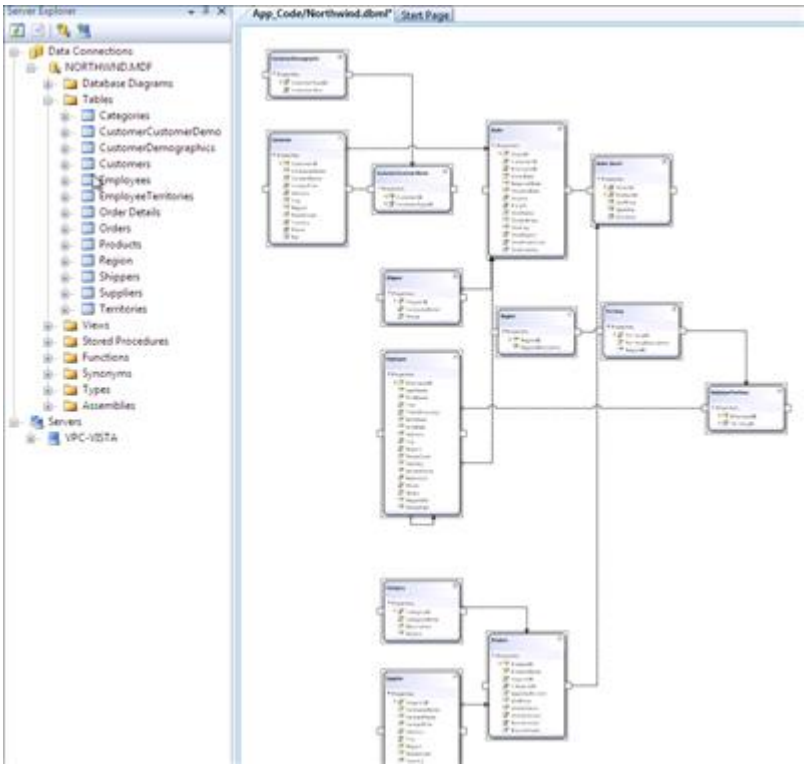
Il faut à présent ajouter le fichier Nortwind.mdf dans ce répertoire : clic droit sur le répertoire > add existing items. Sélectionnez le fichier sur votre disque et importez-le.
Prochaine étape : Ajoutez une classe Linq qui gérera le mapping entre notre site et notre base de données. Grâce à Linq toutes les requêtes vers les bases de données sont générées sans que nous devions écrire la moindre ligne SQL.
Faites un clic droit sur la racine de votre site et cliquez sur **Add new item**



Nous allons ajoutez un élément de type **LINQ to SQL Classes** que nous appellerons Northwind.dbml
Visual studio vous conseil de mettre cet objet dans le répertoire **App_code**. Acceptez et cliquez sur Yes.



Le fichier Linq est créé mais ne contient aucune table, pour ajouter toutes les tables de la base de données faites les glisser dans le fichier depuis le **Server Explorer**.
Le fichier doit maintenant contenir le schéma de la base de données avec les relations.



Il nous reste une dernière étape avant de pouvoir tester notre site.
Editez le fichier web.config et localisez la section **<dynamicData>**.
Il nous faut modifier son attribut **enableTemplates** et le mettre à true

web.config

```
<dynamicData dataContextType="" enableTemplates="true">
```

Appuyez sur la touche F5 pour compiler et exécuter le site, si vous avez suivie correctement toutes les étapes vous devriez voir s'afficher une page avec la liste des tables se trouvant dans votre fichier Linq

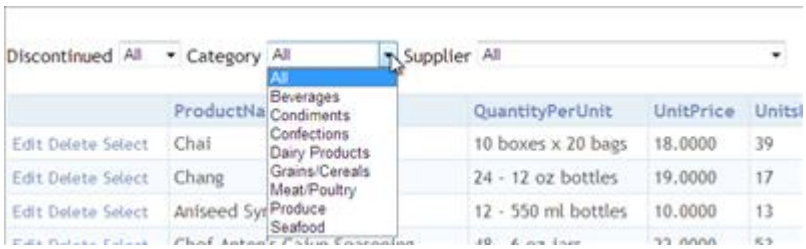
3 - Présentation des fonctionnalités de base

Maintenant que notre site est créé et opérationnel nous pouvons explorer les différentes fonctionnalités existantes de base sans avoir à taper la moindre ligne de code.
Comme dit dans le chapitre précédent la première page affiche la liste des tables, choisissez la table **Customers**.

Une nouvelle page s'affiche et vous montre la liste des clients ainsi que le détail du client sélectionné.
Pour afficher les informations d'un autre client cliquez sur le lien **select**

Par défaut toutes les pages sont identiques et générées automatiquement par le site. On parle de template de page.
Cette vue vous donne également la possibilité de supprimer, de modifier et d'ajouter un enregistrement.

Pour illustrer la manière dont l'application permet de filtrer les données nous allons utiliser la table **Product**.
Les listes déroulantes situées en haut de la liste permettent de filtrer les enregistrements :



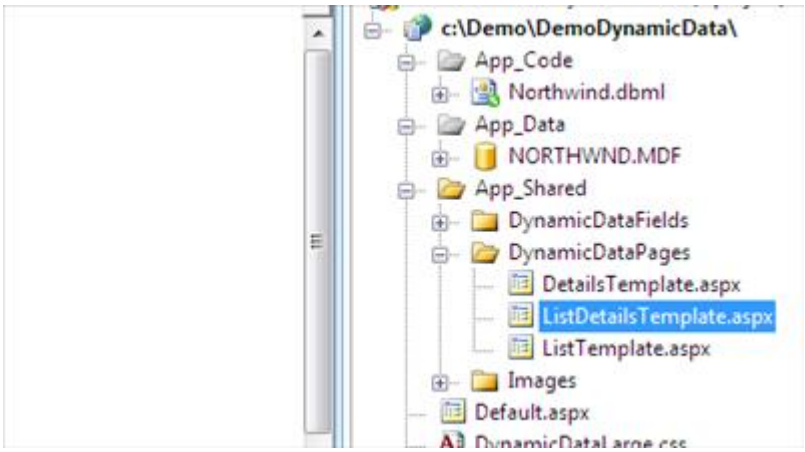
Par défaut les tables peuvent être filtrées sur les champs de type booléan et sur les clefs étrangères.
Les relations sont également affichées dans la liste. Vous pouvez par exemple à partir d'un produit naviguer vers la fiche du fournisseur lié mais également obtenir la liste de toutes les commandes reprenant cet article.
En mode édition les champs de clefs étrangères se transforment en liste déroulante.

Autre point intéressant : l'application vérifie les contraintes de la base de données et avertit l'utilisateur si les données entrées sont incorrecte. Par exemple si un champ obligatoire a été omis
De plus il vérifie la taille maximum pour un champ de type texte et limite automatiquement la longueur des zones de textes.

4 - Personnalisation des pages

Sans aucun développement le site web répond déjà à beaucoup de besoin mais il est également possible de personnaliser l'application.

Vous pouvez par exemple modifier la page générique partagée par toutes les tables. Cela signifie que vos modifications seront visibles partout.
Comme exemple je vais ajouter le logo de ma société en haut de toutes les pages.
La page concernée se trouve dans votre solution à cet endroit : App_Shared > DynamicDataPages > ListDetailsTemplate.aspx

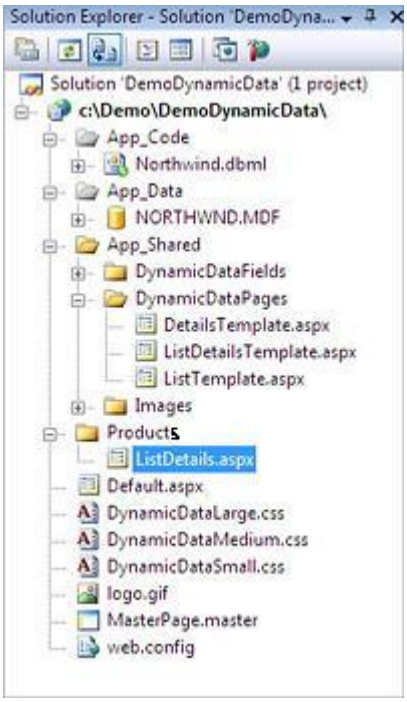


Il s'agit d'une simple page aspx contenant entre autres une GridView et un detailView.
Modifiez cette page et ajoutez un control aps.net Image avec le logo de votre société :

```
<asp:Image runat="server" ImageUrl="~/logo.gif"/>
```

Démarrer le site web et vous pouvez constatez que le logo est présent sur le haut de toutes les pages.

Avec cet exemple la page a bien été modifiée mais reste identique pour toutes les tables. Admettons que nous voulons une page différente pour la table Products affichant la date du jour.
Commencer par créer un répertoire à la racine de votre site web portant le nom de la table, de notre cas Products
Copier/coller ensuite le fichier ListDetailTemplate.aspx dans ce répertoire et renommez le en ListDetails.aspx.



Cette man#uvre permet de modifier une page à partir de zéro.
Enlever la ligne de code affichant le logo et remplacez la par celle ci pour afficher la date :

```
<h2><%=DateTime.Now.ToString("dd/MM/yyyy")</h2>
```

Relancez votre application et constatez que toutes les pages contiennent le logo de votre société excepté la table Products qui affiche l'heure.

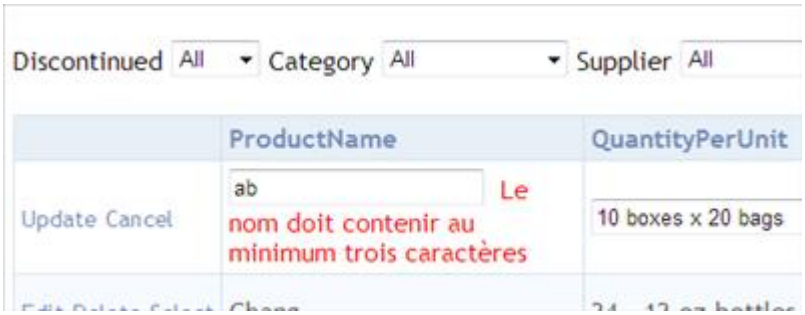
5 - Les validations personnalisées

Dans le troisième chapitre je vous ai montré que Dynamic Data se basait sur les contraintes de la base de données pour ajouter des validations sur la page.
Il est également possible de définir des validations supplémentaires sur le site.
Nous voulons par exemple que le nom d'un produit contienne au minimum trois caractères.
Pour cela ajoutez une classe nommé Product.cs dans votre répertoire App_Code dont voici le contenu :

Product.cs

Product.cs
```
public partial class Product
{
 partial void OnProductNameChanging(string value)
 {
  if (value.length < 3)
  {
   throw new Exception("Le nom doit contenir au moins trois caractères");
  }
 }
}
```

Il est important de définir la classe et la méthode comme étant **partial** cela signifie que votre code viendra étendre la classe Product générée par l'application.
Si on regarde de plus près le code on s'aperçoit qu'il est possible d'étendre toute une série de méthodes comme dans ce cas **OnProductNameChanging** qui sera exécuté à chaque modification du champ ProductName de notre table. Si la valeur proposée est plus courte que trois caractères une exception sera levée.



Une autre manière de faire consiste à utiliser des attributs de classe pour définir par exemple un range de valeurs valide.
Essayons par exemple d'obliger l'utilisateur à rentrer un prix unitaire compris entre 0 et 100.
Pour commencer nous devons ajoutez un using à notre classe :

```
using System.Web.DynamicData;
```

Ajoutons ensuite un attribut à notre classe :

```
[Range("UnitPrice",0,100)]
public partial class Product
{
 ...
}
```

L'attribut Range reçoit trop paramètres : le premier définit le champ à valider, le deuxième le minimum accepté et le troisième le maximum.
Essayons maintenant de mettre 150 comme prix pour l'un de nos produits :

Sans trop de surprise notre prix est refusé.

6 - Personnalisation des champs

La personnalisation ne se limite pas aux pages mais peu aussi modifier les champs.
Pour illustrer ce concept nous allons remplacer la case à cocher servant à afficher un champ booléen par deux radio buttons (oui/non).
Le répertoire App_Shared > DynamicDataFields contient tout les contrôles utilisateurs asp.net servant à afficher les champs. Tous les contrôles sont doublés : un template pour l'affichage et un autre (_Edit.ascx) pour l'édition.
Ajoutons deux contrôles utilisateurs **RadioBoolean.ascx** et **RadioBoolean_Edit.ascx** dont voici le code :

RadioBoolean.ascx

```
<%@ Control Language="C#" Inherits=System.Web.DynamicData.FieldTemplateUsercontrolBase" %>
<script runat="server">
 protected override void OnDataBinding(EventArgs e) {
  base.OnDataBinding(e);

  object val = DataValue;
  if (val != null)
  {
   rdoYes.Checked = (bool) val;
  }
 }
</script>

<asp:RadioButton GroupName="Boolean" runat="server" ID="rdoYes" Enabled="False" Text="Oui" /></br>
<asp:RadioButton GroupName="Boolean" runat="server" ID="rdoNo" Enabled="False" Text="Non" />
```

RadioBoolean_Edit.ascx

```
<%@ Control Language="C#" Inherits=System.Web.DynamicData.FieldTemplateUsercontrolBase" %>
<script runat="server">
 protected override void OnDataBinding(EventArgs e) {
  base.OnDataBinding(e);

  object val = DataValue;
  if (val != null)
  {
   rdoYes.Checked = (bool) val;
  }
 }

 protected override void ExtractValues(IOrderedDictionnary dictionary) {
  dictionary[DataField] = rdoYes.Checked;
 }
</script>

<asp:RadioButton GroupName="Boolean" runat="server" ID="rdoYes" Enabled="True" Text="Oui" /></br>
<asp:RadioButton GroupName="Boolean" runat="server" ID="rdoNo" Enabled="True" Text="Non" />
```

Il reste à présent à lier nos contrôles avec un champ de la table Product. Cela se fait via un attribut de notre classe Product :

```
[RenderHint("Discontinued","RadioBoolean")]
public partial class Product
{
 ...
}
```

L'attribut **RenderHint** permet de faire la liaison, il reçoit deux paramètres : le premier pour définir le champ à modifier et le deuxième qui spécifie le contrôle à utiliser.

A partir de maintenant le champ Discontinued de la table product sera toujours représenté par des radio buttons

7 - Définir le champ à utiliser pour les clés étrangères

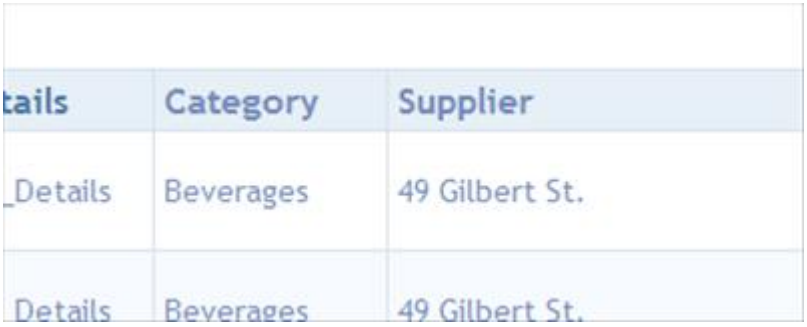
Dans les listes Dynamic Data utilise un hyper lien pour représenter une clé étrangère et permettre de naviguer entre les tables

Il est possible de modifier le texte choisit pour l'hyper lien. Dans la capture d'écran l'on peut voir que le nom du fournisseur (Supplier) est utilisé.
Essayons de le remplacer par l'adresse. Ce ne serait pas vraiment intéressant dans un cas réel mais cela nous permet d'illustrer ce concept.
Créez une nouvelle classe App_Code et nommez la **Supplier.cs**

Supplier.cs
```
<br/>

[DisplayColumn("Address")]
public partial class Supplier
{
}
```

Le code est relativement simple il suffit de définir la classe comme partial et d'ajouter un attribut **DiplayColumn** pour désigner le champ à utiliser.
Ouvrez à nouveau votre table Products



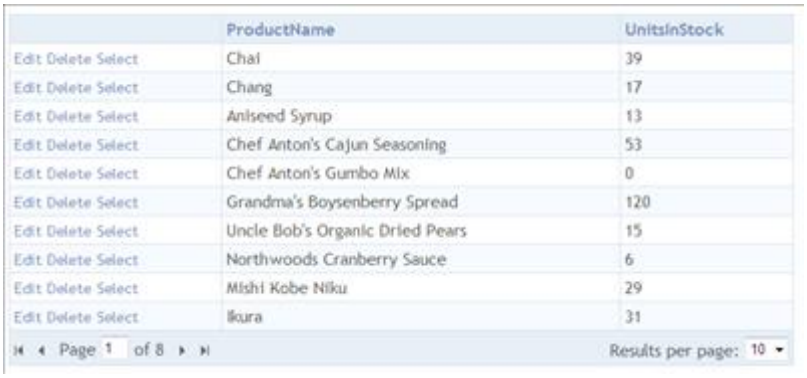
8 - Définir les champs à afficher dans la liste

De base Dynamic Data affiche tous les champs de la base de données dans la liste mais il est évidement possible de spécifier les champs à afficher.
Nous pouvons faire cela dans notre fichier Products > ListDetails.aspx.
Commencer par localiser la définition du contrôle <asp:DynamicGridView> et ajoutez lui la propriété **AutoGenerateColumns="false"**.
De cette manière le site ne va plus créer automatiquement les colonnes, il faut maintenant définir manuellement les colonnes à afficher. Voici le code final pour le contrôle :

ListDetails.aspx

```
<asp:DynamicGridView ID="GridView1" runat="server" DataSourceID="GridDataSource"
        AutoGenerateSelectButton="True" AutoGenerateEditButton="True" AutoGenerateDeleteButton="true"
        AllowPaging="True" AllowSorting="True" EnableQueryStringSelection="True" OnDataBound="OnGridViewDataBound
        OnRowEditing="OnGridViewRowEditing" OnSelectedIndexChanging="OnGridViewSelectedIndexChanging"
        CssClass="gridview" AlternatingRowStyle-CssClass="even" AutoGenerateColumns="false">
        <Columns>
            <asp:DynamicField DataField="ProductName" />
            <asp:DynamicField DataField="UnitsInStock" />
        </Columns>
        <PagerStyle CssClass="Footer"/>
        <PagerTemplate>
            <uc1:GridViewPager runat="server" />
        </PagerTemplate>
        <EmptyDataTemplate>
            There are currently no items in this table.
        </EmptyDataTemplate>
    </asp:DynamicGridView>
```

Si l'on exécute à nouveau le site on peut remarquer que seules les colonnes définies dans la section <Columns> sont affichées.



9 - "Templatisation" des colonnes

Nous avons jusqu'a présent utilisé des **DynamicField** pour afficher les colonnes de notre GridView, cela est suffisant dans la majorité des cas, cependant il est parfois intéressant de recourir à des colonnes "templatisée". Autrement dit ne pas utiliser la simple vue de type tableau.
Une nouvelle fois ce changement se fera directement dans le code de notre contrôle **DynamicGridView** :

ListDetails.aspx

```
<asp:DynamicGridView ID="GridView1" runat="server" DataSourceID="GridDataSource"
        AutoGenerateSelectButton="True" AutoGenerateEditButton="True" AutoGenerateDeleteButton="true"
        AllowPaging="True" AllowSorting="True" EnableQueryStringSelection="True" OnDataBound="OnGridViewDataBound
        OnRowEditing="OnGridViewRowEditing" OnSelectedIndexChanging="OnGridViewSelectedIndexChanging"
        CssClass="gridview" AlternatingRowStyle-CssClass="even" AutoGenerateColumns="false">
        <Columns>
            <asp:DynamicTemplateField>
                <ItemTemplate>
                    Nom du produit
  : <asp:DynamicControl runat="server" DataField="ProductName" /><br />
                    Unités en
 stock : <asp:DynamicControl runat="server" DataField="UnitsInStock" /><br />
                </ItemTemplate>
                <EditItemTemplate>
                    Nom du produit
  : <asp:DynamicEditControl runat="server" DataField="ProductName" /><br />
                    Unités en
 stock : <asp:DynamicEditControl runat="server" DataField="UnitsInStock" /><br />
                </EditItemTemplate>
                </asp:DynamicTemplateField>
        </Columns>
        <PagerStyle CssClass="Footer"/>
        <PagerTemplate>
            <uc1:GridViewPager runat="server" />
        </PagerTemplate>
        <EmptyDataTemplate>
            There are currently no items in this table.
        </EmptyDataTemplate>
    </asp:DynamicGridView>
```

Nous avons retiré les définitions de colonnes faites précédemment pour les remplacer par un **<DynamicTemplateField>**.
Deux templates différents doivent être définis : un premier pour l'affichage (<ItemTemplate>) et un second pour l'édition (<ItemEditTemplate>).
Nous ajoutons respectivement des contrôles de type <DynamicControl> et <DynamicEditControl>. Ces deux contrôles s'occupent d'afficher les données en utilisant les templates se trouvant dans le répertoire App_Shared > DynamicDataFields.

10 - Conclusion

Cet article a pour objectif de présenter les principales fonctionnalités et possibilités d'Asp.net 3.5 Dynamic Data. Il est évident qu'il est possible de faire beaucoup plus complet et complexe à l'aide de cette technologie. Mais cela aurait dépassé le scope de cet article.

11 - Liens

Web cast sur Dynamic Data par Cardi : **http://jlambert.developpez.com/webcast/dynamicdatapreviewapril/** Mon site : **http://lefortludovic.developpez.com**
Mon blog : **http://blogs.ezos.com/blog/lle**

Je tiens à remercier **RideKick** pour la correction orthographique de cet article.