

Construisez vos premiers contrôles serveur ASP.NET

Cet article est composé de ce document et d'un fichier contenant les sources des démonstrations vous permettant d'approfondir quelque peu vos connaissances.

A noter qu'il est essentiel de tester et de lire les commentaires de code des démonstrations pour bien comprendre cet article.

1. Les contrôles serveur

Le contrôle serveur est un objet qui s'exécute bien évidemment sur le serveur. Le contrôle fournit aussi un Namespace contenant des classes, méthodes et propriétés utiles au développeur l'implémentant pour son développement.

Comme il s'agit d'un contrôle ASP.NET, du code HTML peut être produit pour le client. De plus, des mécanismes de conservation d'état lors de post-back peuvent être utilisés.

Le développeur de contrôles peut mettre en place le design de ses contrôles pour Visual Studio. Il peut aussi faire en sorte de permettre de renseigner plus facilement les propriétés de ceux-ci.

Pour construire son propre contrôle, on peut aussi hériter d'un contrôle serveur existant. Bien que réaliser un contrôle serveur ne soit pas insurmontable, la tâche est toutefois un peu plus ardue que de réaliser un contrôle utilisateur.

Le contrôle serveur dérive obligatoirement de `System.Web.UI.Control`. Cela permet en particulier de pouvoir bénéficier des mécanismes internes du Runtime ASP.NET

Plus couramment, on développe des `WebControls` (héritage de `System.Web.UI.WebControls.WebControl`).

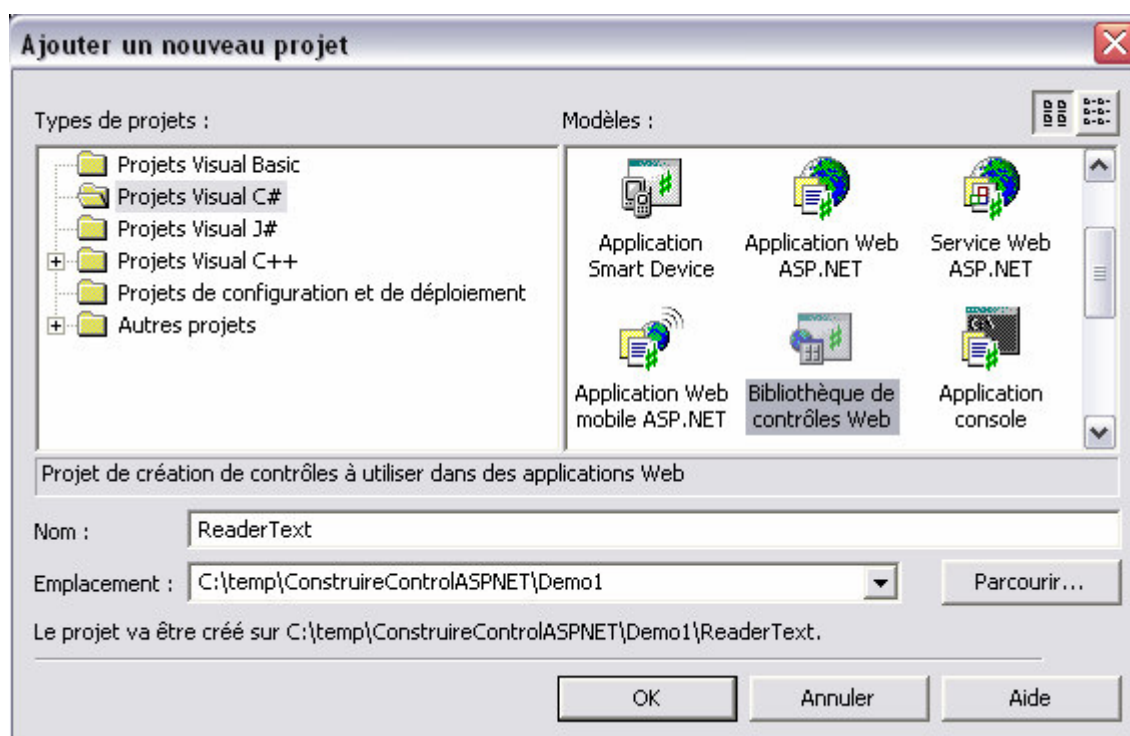
A noter qu'il se trouve forcément dans une assembly différente de votre projet ASP.NET afin de pouvoir entre autres être réutilisés dans d'autres projets. On peut imaginer concevoir une bibliothèque de contrôles dans une seule assembly.

2. DEMO 1 : Mon premier WebControl : Exemple d'un composant permettant de visualiser un fichier texte

Création rapide d'un contrôle de base

On se sert en général pour plus de facilité d'un squelette de contrôle que l'on obtient en créant un projet « Bibliothèque de contrôles Web ».

.NET passionnément, tout simplement



➤ Ajouter le contrôle dans la Boîte à outils (ToolBox)

Pour être visuellement utilisable dans la ToolBox de Visual Studio, on peut fournir une icône à chaque contrôle serveur développée. On choisira un fichier icône que l'on encapsulera comme ressource dans l'assembly.

Pour que l'icône soit visible, on utilise l'attribut `ToolboxBitmapAttribute` du namespace `System.Drawing` qu'il est nécessaire de référencer.

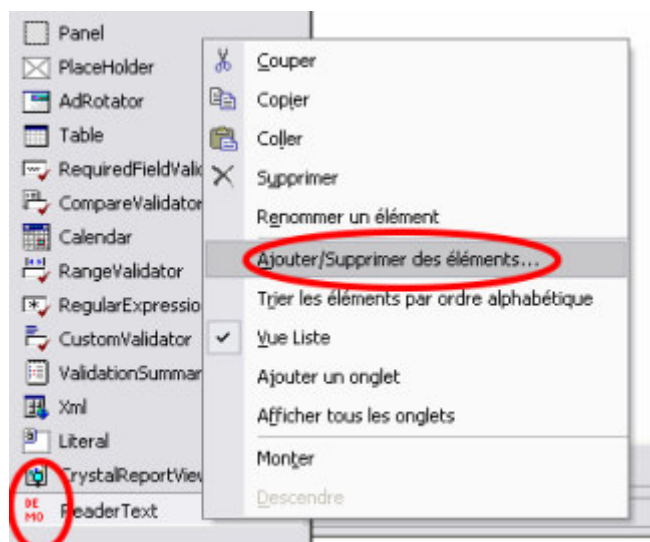
```
1 using System;
2 using System.Web.UI;
3 using System.Web.UI.WebControls;
4 using System.ComponentModel;
5 using System.IO;
6 using System.Drawing;
7
8 namespace Demo1
9 {
10     [DefaultProperty("FilePath"),
11     ToolboxData("<(0):ReaderText, runat=server></(0):ReaderText>"),
12     ToolboxBitmap(typeof(ReaderText), "Icon.ico")]
13     public class ReaderText : System.Web.UI.WebControls.WebControl
14     {
```

La démonstration permet de lire un fichier texte et d'effectuer un rendu intégral entre des balises Html `<PRE>`.

.NET passionnément, tout simplement

On notera la facilité pour effectuer le rendu. Il suffit de surcharger la méthode Render de la classe WebControl dont on dérive. Le rendu Html se fait grâce à l'objet HtmlTextWriter.

On compile la dll contenant le WebControl et on l'ajoute à la Toolbox de Visual Studio .NET. Un Drag and Drop classique permet ensuite d'intégrer le contrôle dans votre page.



Le changement de propriétés du contrôle dans Visual Studio est « live ». On peut empêcher la prise en compte d'une propriété par le designer de Visual Studio grâce à l'attribut DesignerSerializationVisibilityAttribute.

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public string FileTextPath
{
    get
    {
        return _fileTextPath;
    }

    set
    {
        _fileTextPath = value;
    }
}
```

🔧 Choisir son éditeur de propriétés

En mode design, comment faire en sorte que de ma fenêtre de propriétés, je puisse avoir éditeur qui puisse me permettre de choisir un fichier texte sur mon disque dur ?

Cela est relativement simple il suffit de rajouter un attribut :

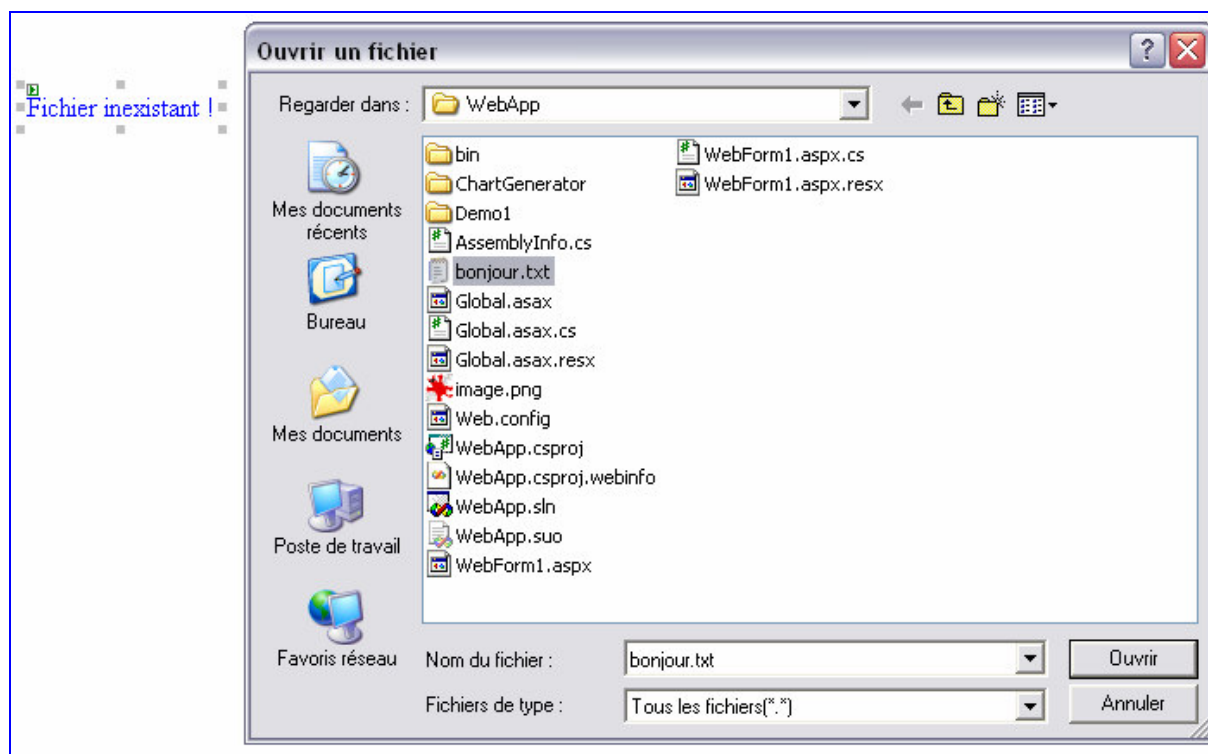
.NET passionnément, tout simplement

`[Editor(typeof(System.Windows.Forms.Design.FileNameEditor), typeof(System.Drawing.Design.UITypeEditor))]`

Compilons. Erreur ! Il nous manque une référence à une assembly visiblement. Il nous faut référencer dans l'assembly la dll suivante :

`System.Design.dll`

Un petit éditeur, ça rend tout de même la vie plus facile :



3. Demo 2 : Propriétés complexes et Designer de Visual Studio : Exemple d'un composant Camembert

Approfondissons nos connaissances en implémentant une propriété dépassant le cadre d'une simple valeur. Nous avons choisi une collection d'un objet qui est simple (composé de trois propriétés : une valeur, une légende, une couleur). Cet objet va permettre de définir chaque tranche du camembert final.

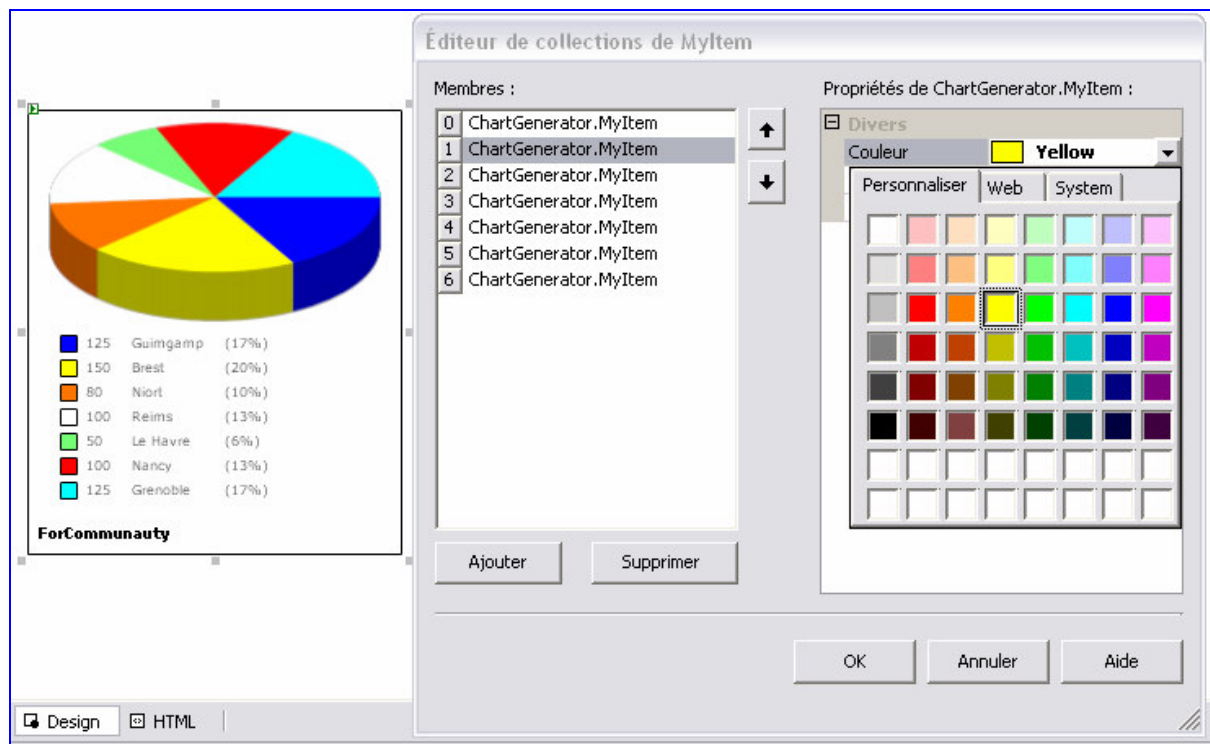
Nous ne détaillerons pas ici la fabrication proprement dite du camembert (voir pour cela le code).

L'éditeur de collection

.NET passionnément, tout simplement

Comment faire en sorte que l'on puisse alimenter la collection MyItemCollection via la fenêtre de propriétés de Visual Studio ? Comment avec peu d'effort utiliser un éditeur de propriété (sans en recoder un soi-même) ? Et bien, le designer s'en charge tout seul !

Voici une vue en mode design : (cool, non ?)



Puisque MyItemCollection implémente l'interface ICollection, Visual Studio attribut à la propriété de notre contrôle un éditeur de collection. Il n'est donc pas nécessaire de placer ceci comme attribut :

```
[Editor(typeof(System.ComponentModel.Design.CollectionEditor),  
typeof(System.Drawing.Design.UITypeEditor))]
```

🚩 La persistance des données

Parfait, mais nous notons que les propriétés contenues dans la collection ne persistent pas. Comment faire pour que les éléments de la collection soient visibles dans le code afin d'être parser par ASP.NET ?

Nous devons ajouter un attribut à la propriété ListItems (constituée de MyItemCollection) :

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
```

Ce qui génère le code suivant :

.NET passionnément, tout simplement

```
ListItems-IsReadOnly="False" ListItems-SyncRoot="System.Collections.ArrayList"
ListItems-IsSynchronized="False" ListItems-IsFixedSize="False" ListItems-Count="2"
ListItems="(Collection)"
```

Ce qui n'est pas suffisant en terme de persistance de données. Nous ajoutons alors un attribut permettant de gérer la persistance :

```
[PersistenceMode(PersistenceMode.InnerDefaultProperty)]
```

Ce qui génère le code suivant :

```
<cc2:Myltem Legende="Marseille" Couleur="Red" Value="100"></cc2:Myltem>
<cc2:Myltem Legende="PSG" Couleur="Blue" Value="10"></cc2:Myltem>
<cc2:Myltem Legende="Monaco" Couleur="Yellow" Value="60"></cc2:Myltem>
<cc2:Myltem Legende="Lyon" Couleur="Lime" Value="150"></cc2:Myltem>
```

Compilons et testons la page. Une erreur ! Il ne trouve pas cc2 :Myltem !

En fait, nous avons oublié de placer un attribut sur la classe :

```
[ParseChildren(true, "ListItems")]
```

Cet attribut autorise le parseur d'ASP.NET à parser les propriétés enfants, ici en particulier notre collection ListItems.

La page résultat :



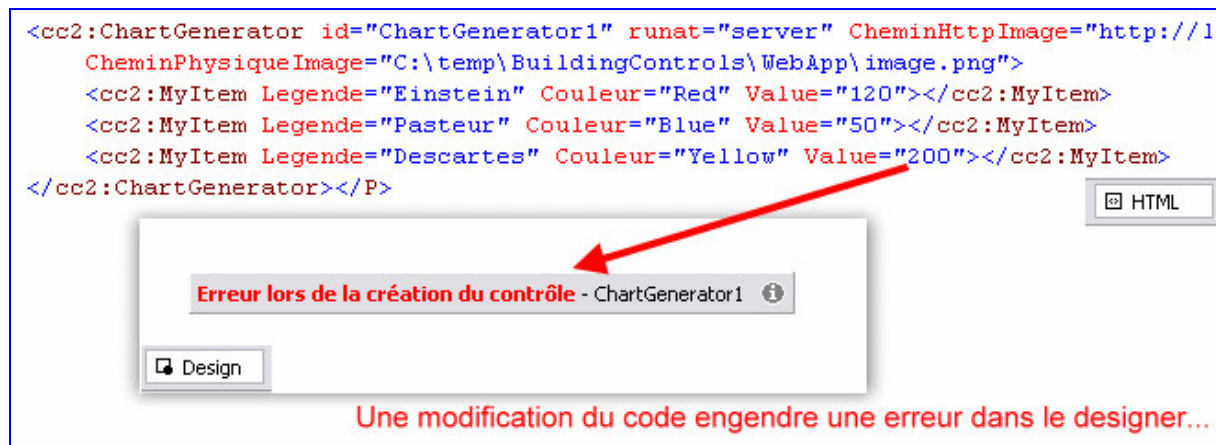
✚ Erreur de code ou bug ?

Dernière particularité :

.NET passionnément, tout simplement

Si on modifie le code et que l'on revient dans le designer, on obtient une erreur. S'agit-il d'une erreur dans le code du contrôle, ou bien est-ce un bug du parseur de Visual Studio ?

J'y ai passé pas mal de temps, sans succès. Si vous avez la réponse, merci de me contacter.



<http://support.microsoft.com/default.aspx?scid=kb;en-us;823194&Product=NETFrame>

4. En savoir plus :

Nous verrons dans de prochains articles comment prendre en compte le ViewState, Comment ajouter du javascript sur le client, comment gérer les événements...

Guide du développement de contrôles serveur ASP.NET

<http://msdn.microsoft.com/library/fre/default.asp?url=/library/FRE/cpguide/html/cpcondevelopingwebformscontrols.asp>

Le livre : Developing ASP.NET Server Controls and Components

<http://www.microsoft.com/mspress/books/5728.asp>

Contactez l'auteur :

Frédéric Mélantois

Email : fmelantois@free.fr