

ASP.NET 2.0 : Quelques nouveautés sur les custom controls

Cet article a été réalisé à partir de la Beta 1 puis de la beta 2 de Visual Studio 2005. Il a pour but de vous initier à quelques nouveautés d'ASP.NET 2.0.

1. Introduction

Nous avons vu à travers quelques articles comment créer ses propres contrôles serveur (Custom Controls) sous ASP.NET 1.1

Nous allons voir à travers cet article quelques nouveautés intéressantes de la version 2.0. Elles sont très nombreuses, difficile de toutes vous les rapporter, aussi ne vous attendez pas à un état exhaustif. Je ne vais évoquer que celles qui ont retenu mon attention au fil de mes développements.

2. Le contrôle composite (CompositeControl)

Il s'agit d'un contrôle qui est composé de plusieurs contrôles, autant dire qu'il s'agit du type de custom control le plus répandu.

Sous ASP.NET 1.1, pour mettre en place un contrôle composite, il nous fallait réaliser plusieurs opérations :

- Héritage d'un WebControl
- Implémentation de l'interface INamingContainer
- S'assurer que les contrôles enfants sont bien créés lors des différents appels, en particulier en mode design (apparence du contrôle sous Visual Studio).

Sous ASP.NET 2.0, la tâche de création d'un contrôle composite se trouve simplifiée. Il suffit que notre contrôle hérite de CompositeControl. Il n'est pas nécessaire de faire appel à la méthode EnsureChildControls() constamment pour s'assurer que les contrôles enfants sont bien créés par exemple dans le mode Design. Rien de révolutionnaire, il est vrai mais cela fait gagner beaucoup de temps lors de la création du contrôle (ceux qui en ont codés me comprendront).

Voici une petite illustration :

```
public class ControlComposite : CompositeControl
{
    private HtmlAnchor _bouton;
    private Label _label;

    protected override void CreateChildControls()
    {
        base.CreateChildControls();
        _bouton = new HtmlAnchor();
        _bouton.InnerText = "Un bouton";
        _bouton.HRef = "http://microsoft.com/france/msdn/";

        _label = new Label();
        _label.Text = " Un label";
    }

    protected override void Render(HtmlTextWriter output)
    {
        _bouton.RenderControl(output);
        _label.RenderControl(output);
    }
}
```

```
}  
}
```

En interne, le CompositeControl surcharge la collection Controls, en appelant la méthode EnsureChildsControls(), assurant l'instanciation des controls, avant de renvoyer la collection.

3. L'état de contrôle (Control State)

Pour les utilisateurs de contrôles fournis en standard par le framework ASP.NET 1.x (DataGrid, DropDownList, ...), comme pour les créateurs de contrôles, la conservation de l'état d'un control est importante lors du rechargement d'une page, en particulier lors des actions dites Post-Back. La conservation de l'état se fait en particulier en associant à la page cliente des données sérialisées. Le post de la page en cours permet au serveur de récupérer ces données et de les dé-sérialiser, « récupérant » ainsi certaines propriétés du contrôle. Vous aurez sans doute reconnu le ViewState.

Sous ASP.NET 2.0, un nouveau type d'état apparaît : il s'agit de l'état de contrôle (ou Control State). Sa particularité est que la propriété EnableViewState mise à false ne l'affecte pas. En effet, on peut parfaitement ne pas souhaiter conserver toutes les données concernant un contrôle sans se priver de tous les états d'un contrôle. Il est connu que le ViewState lorsqu'il est activé peut générer un volume très important pour la page cliente en cours. Bien que la sérialisation, en particulier la compression, ait été nettement améliorée en ASP.NET 2.0, on n'hésitera pas à se priver de son usage. En effet, sous ASP.NET 1.1, on a pris l'habitude de bien faire attention à désactiver le ViewState quand celui-ci n'était pas indispensable. Mais c'est évidemment au prix de se priver de fonctionnalités essentielles de certains de nos contrôles (l'exemple du paging du DataGrid est une bonne illustration).

Avec le control state, tout objet simple, tel que entier, string, tableau, HashTable, peut être « conservé » lors d'un post-back.

Passons à la pratique :

Prenons un cas très élémentaire. Nous construisons un contrôle qui comporte un label et un bouton. Lorsqu'on appuie sur le bouton, on incrémente le label qui affiche une valeur qui est initialement à zéro.

Sous ASP.NET 1.1, l'utilisateur de notre contrôle devra impérativement ne pas désactiver le ViewState, pour ne pas gêner l'incrémementation à chaque clic sur le bouton.

Voici un exemple de code en utilisant le ViewState (à l'ancienne) comme sous ASP.NET 1.1 :

```
public class ControlViewState : CompositeControl ,  
IPostBackEventHandler  
{  
    private Label _label;  
    private HtmlAnchor _bouton;  
  
    protected override void CreateChildControls()  
    {  
        base.CreateChildControls();  
        _bouton = new HtmlAnchor();  
        _bouton.InnerText = "Ajouter";  
    }  
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
        _bouton.HRef = "#";
        _bouton.ID = "Button" + this.UniqueID;
        _bouton.Attributes["OnClick"] =
Page.ClientScript.GetPostBackEventReference(this, "Click");

        _label = new Label();
        _label.Text = _compteur.ToString();
    }

    private int _compteur
    {
        get
        { return (ViewState["Compteur"] != null) ?
Int32.Parse(ViewState["Compteur"].ToString()) : 0; }
        set
        {
            ViewState["Compteur"] = value;
        }
    }

    #region IPostBackEventHandler Members

    void IPostBackEventHandler.RaisePostBackEvent(string
eventArgument)
    {
        if (eventArgument == "Click")
        {
            _compteur++;
        }
    }

    #endregion

    protected override void Render(HtmlTextWriter output)
    {
        _bouton.RenderControl(output);
        _label.RenderControl(output);
    }
}
```

Voici le même exemple élémentaire codé sous ASP.NET 2.0. On implante ici le Control State :

```
    public class ControlControlState : CompositeControl,
IPostBackEventHandler
    {
        private Label _label;
        private HtmlAnchor _bouton;

        protected override void OnInit(EventArgs e)
        {
            base.OnInit(e);
            //on enregistre le controle pour que puisse se faire le
LoadControlState
            Page.RegisterRequiresControlState(this);
            //car le LoadControlState se fait après...

        }

        protected override void LoadControlState(object savedState)
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
    {
        //exemple volontaire pour montrer comment traiter
plusieurs objets
        //même si dans notre exemple, nous n'avons que l'objet
compteur

        object[] allStates = (object[])savedState;
        if (allStates[0] != null)
            base.LoadControlState(allStates[0]);

        _compteur = Convert.ToInt32(allStates[1].ToString());
    }

protected override object SaveControlState()
{
    string prop = _compteur.ToString();
    object baseState = base.SaveControlState();
    object[] allStates = new object[2];
    allStates[0] = baseState;
    allStates[1] = prop;

    return allStates;
}

protected override void CreateChildControls()
{
    base.CreateChildControls();
    _bouton = new HtmlAnchor();
    _bouton.InnerText = "Ajouter";
    _bouton.HRef = "#";
    _bouton.Attributes["OnClick"] =
Page.ClientScript.GetPostBackEventReference(this, "Click");

    _label = new Label();
    _label.Text = _compteur.ToString();
}
private int _compteur = 0;

#region IPostBackEventHandler Members

void IPostBackEventHandler.RaisePostBackEvent(string
eventArgument)
{
    if (eventArgument == "Click")
    {
        _compteur++;
    }
}

#endregion

protected override void Render(HtmlTextWriter output)
{
    _bouton.RenderControl(output);
    _label.RenderControl(output);
}
}
```

Lors de l'utilisation de ce contrôle dans une page, n'oubliez pas de désactiver le ViewState afin que vous puissiez vous rendre compte de l'utilité du Control State.

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

Dans notre exemple, l'état de contrôle est conservé par un tableau d'objets.

Par cette ligne :

```
base.LoadControlState(allStates[0])
```

on ne fait que récupérer un objet (qui peut être aussi un tableau d'objet) du contrôle duquel on hérite. Dans notre cas, le contrôle `ControlControlState` hérite de `CompositeControl` qui hérite lui-même de `WebControl`.

Puisque `WebControl` n'a pas de propriétés en control state, il n'était pas nécessaire dans notre exemple de récupérer le control state du contrôle parent. Mais pour des raisons pédagogiques, nous avons émis ce code général afin de bien comprendre la mécanique du Control State.

Concrètement, sur la page cliente, le control state est maintenu par le champ caché suivant : `"__VIEWSTATE"`.

Aucun champ supplémentaire n'a été ajouté côté client pour le fonctionnement du control state.

On peut lire le contenu du champ `"__VIEWSTATE"` comme on le faisait sous ASP.NET 1.1 mais la manière de sérialiser les données a changé.

La chaîne contenu dans le champ caché `"__VIEWSTATE"` est en base 64. Il nous suffit de convertir cette chaîne via :

```
public string ConvertFromBase64(string chaine)
{
    byte[] tbyte = Convert.FromBase64String(chaine);
    UTF8Encoding encodage = new UTF8Encoding(false);
    return encodage.GetString(tbyte);
}
```

Et c'est de cette manière, que l'on peut observer les différences entre les données sérialisées en ASP.NET 1.1 et celles en ASP.NET 2.0.

Sous ASP.NET 2.0, vous remarquerez qu'il y a un certain nombre de caractères non visibles qui remplace le système de balise qui existait sous ASP.NET 1.1. C'est ainsi que la compression des données a été améliorée.

On peut avoir un aperçu de ces caractères en traitant cette chaîne via une fonction que vous retrouverez dans l'un de mes précédents articles (Construire un contrôle Wysiwyg) :

```
public string ConvertSymbolToHtml(string str)
{
    Hashtable h = new Hashtable();
    for (int i = 1; i <= 48; i++)
    {
        h.Add(System.Web.HttpUtility.HtmlDecode("&#" +
i.ToString() + ";").ToString(), "&#" + i.ToString() + ";");
    }

    foreach (DictionaryEntry dic in h)
    {
        str = str.Replace(dic.Key.ToString(),
dic.Value.ToString());
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
    }  
    return str;  
}
```

Ensuite, chaque caractère non visible correspond à un élément. Par exemple, ; correspond au premier noeud.

Je vous invite à lire pour de plus grandes précisions l'excellent article de MSDN Magazine d'octobre 2004 :

<http://msdn.microsoft.com/msdnmag/issues/04/10/ViewState/default.aspx>

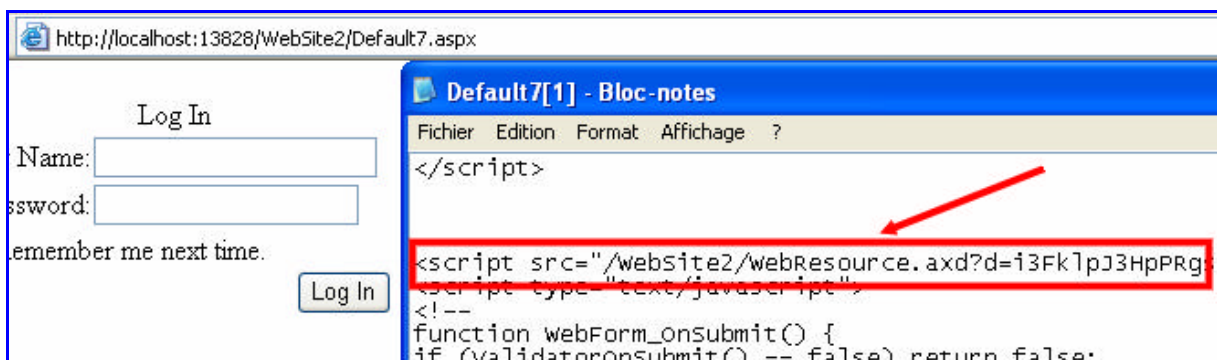
Vous aurez remarqué aussi, dans le code plus haut, que la méthode Page.GetPostBackEventReference devient obsolète sous ASP.NET 2.0, elle est remplacée par la méthode Page.ClientScript.GetPostBackEventReference.

4. L'utilisation des Web Ressources

Les Web Ressources représentent une grande avancée dans la construction de contrôles. De quoi s'agit-il exactement ?

Pour bien comprendre, nous allons nous mettre dans la peau d'un simple utilisateur de contrôles ASP.NET 2.0 Le nouveau Framework est riche en nouveaux contrôles. Dans une page, déposez de la boîte à outils le contrôle Login. Sauvez la page, puis lancez le navigateur pour visualiser votre page.

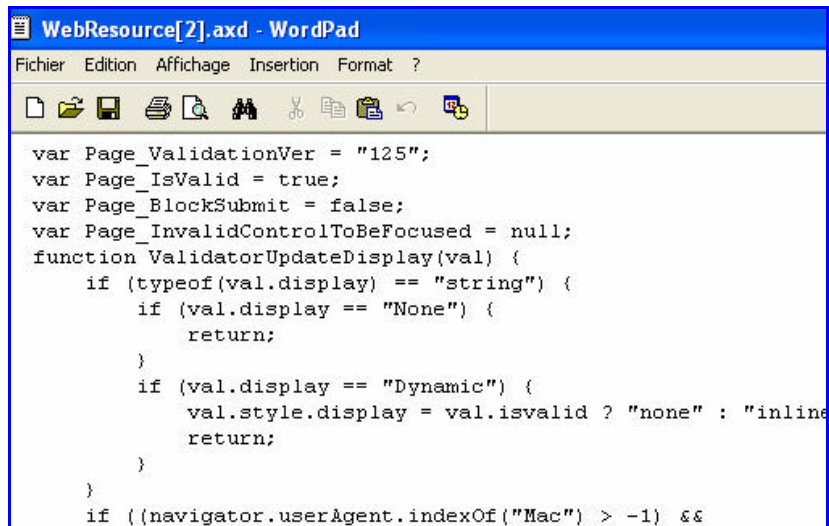
A partir de celui-ci, regardez la source de la page cliente. Vous pouvez voir le script suivant :



Copiez dans votre navigateur l'url du script client, puis ouvrez le fichier généré :

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls



```
var Page_ValidationVer = "125";
var Page_IsValid = true;
var Page_BlockSubmit = false;
var Page_InvalidControlToBeFocused = null;
function ValidatorUpdateDisplay(val) {
    if (typeof(val.display) == "string") {
        if (val.display == "None") {
            return;
        }
        if (val.display == "Dynamic") {
            val.style.display = val.isvalid ? "none" : "inline";
            return;
        }
    }
}
if ((navigator.userAgent.indexOf("Mac") > -1) &&
```

Nous retrouvons le javascript nécessaire au fonctionnement du contrôle. Nous venons de mettre en évidence une Web Ressource.

Une Web Ressource n'est pas uniquement un fichier de script client, il peut s'agir aussi d'images. On voit nettement l'intérêt d'inclure des Web Ressources à l'assembly d'une bibliothèque de custom controls. Il n'est plus nécessaire de faire déployer physiquement sur le serveur des images ou des fichiers js, ou encore des CSS.

Nos fichiers Web Ressources sont en fait tout simplement pris en charge par un Http Handler.

Mettons en oeuvre une Web Ressource pour illustrer nos propos. Très simplement, nous allons définir un fichier texte contenant le script suivant :

```
alert('l'Http Handler WebResource.axd me prend en charge pour que mon navigateur affiche cette alerte');
```

Il faut ajouter ce fichier texte que nous appellerons « fichiertexte.txt » à la solution et le mettre en ressources incorporées.

Nous ajoutons ensuite à la description de notre assembly dans le fichier assemblyinfo.cs, un nouvel attribut :

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: WebResourceAttribute("WebControlLibrary2.fichiertexte.txt",
    "application/x-javascript")]
[assembly: AssemblyTitle("WebControlLibrary2")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("WebControlLibrary2")]
[assembly: AssemblyCopyright("Copyright © 2004")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

Celui-ci se trouve dans le namespace suivant :

```
using System.Web.UI;
```

Il précise la ressource qui sera gérée via l'Http Handler « WebRessource.axd ».

Ensuite, très simplement, il nous suffit de faire appel à notre Web Ressource dans notre custom control :

```
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
    Page.ClientScript.RegisterClientScriptResource( typeof(WebCustomControl4),
    "WebControlLibrary2.fichiertexte.txt" );
}
```

Il nous suffit de générer notre dll et de l'utiliser dans une page aspx pour vérifier que nous avons bien une alerte javascript au chargement de la page dans le navigateur.

Nul doute que vous trouverez un très grand intérêt à mettre en Web Ressources vos images et fichiers de script lors de la construction de contrôles très élaborés.

NB : La beta 1 ne permettait pas de faire fonctionner les scripts clients à cause d'un bug. Celui-ci a été résolu à partir de la beta 2.

5. Les thèmes sur les propriétés des controls

Avec l'arrivée de la version 2.0, il est possible de « skinner » ses contrôles lors du chargement d'une page. Comme vous avez pu le voir dans un de mes précédents articles ou encore lors des DevDays 2005, la localisation physique des thèmes se trouve par défaut dans le répertoire « App_Themes », où sont définis tous les contrôles que l'on souhaite en thèmes. Toutes les propriétés des contrôles ne sont pas « skinnables » comme vous avez pu le remarquer. En effet, il n'y a aucun intérêt à le faire pour certaines propriétés.

Lorsque nous développons nos propres contrôles, par défaut, l'utilisateur pourra rattacher la propriété à un thème. Pour éviter que certaines propriétés (le contenu texte d'un control par exemple) puissent être « skinnables », on applique un attribut :

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
[Themeable(false)]
public string Text
{
    get { return _text; }
    set
    {
        _text = value;
        EnsureChildControls();
        _label.Text = _text;
    }
}
```

Par défaut, le ThemeableAttribute est à true.

6. Le Design-Time : smart-tags

Vous avez pu vous apercevoir que les nouveaux contrôles fournis dans Visual Studio 2005 étaient très riches fonctionnellement. Ils sont accompagnés en mode design de smart tags. Cet ajout très appréciable est disponible aussi pour les créateurs de contrôles. Vous allez voir que la mise en oeuvre d'un smart tag est un véritable jeu d'enfant.

Nous allons prendre un exemple des plus simples : la création d'un contrôle générant un div avec du texte et un fond de couleur. Bien entendu, l'idée est de créer un smart tag associé à ce contrôle permettant de gérer la couleur et le texte.

Voici le squelette de notre contrôle :

```
[Designer(typeof(DesignerAvecSmartTag)), ParseChildren(true),
PersistChildren(true)]
public class ControlAvecSmartTag : CompositeControl
{
    private HtmlGenericControl _bloc;
    private Label _label;

    private string _text = "Mon Label";
    public string Text
    {
        get { return _text; }
        set
        {
            _text = value;
            EnsureChildControls();
            _label.Text = _text;
        }
    }

    private Color _couleur = Color.Transparent;
    public Color Couleur
    {
        get
        {
            return _couleur;
        }
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
        set
        {
            _couleur = value;
            EnsureChildControls();
            _bloc.Style[HtmlTextWriterStyle.BackgroundColor] =
            _couleur.ToKnownColor().ToString();
        }
    }

    protected override void CreateChildControls()
    {
        base.CreateChildControls();

        _bloc = new HtmlGenericControl("DIV");
        _label = new Label();
        _label.Text = _text;
        _bloc.Style[HtmlTextWriterStyle.BackgroundColor] =
        _couleur.ToKnownColor().ToString();

        _bloc.Controls.Add(_label);
    }

    protected override void Render(HtmlTextWriter output)
    {
        EnsureChildControls(); //ne pas oublier pour le designer
        _bloc.RenderControl(output);
    }
}
```

Ce code ne nécessite pas de commentaires particuliers si ce n'est la présence de la méthode "EnsureChildControls" qui nous sera indispensable quand notre contrôle sera « manipuler » par le designer de Visual Studio 2005. Je vous ai dit que le fait de dériver de CompositeControl permettait de s'épargner énormément d'appel à la méthode « EnsureChildControl ». Comme celle-ci ne peut être implémentée automatiquement dans une propriété que vous venez de créer, il est nécessaire de faire cet appel quand le designer interagit sur le control, ce qui n'est pas le cas lors du rendu sur le site Web. C'est pourquoi, vous trouvez dans ce code des appels à « EnsureChildControls ».

Nous allons maintenant créer le « control designer » pour que notre contrôle puisse être géré par le designer de Visual Studio.

```
public class DesignerAvecSmartTag : ControlDesigner
{
    private ControlAvecSmartTag ctrl;

    public override void
Initialize(System.ComponentModel.IComponent component)
    {
        if (component is ControlAvecSmartTag)
        {
            ctrl = (ControlAvecSmartTag)component;
        }
        base.Initialize(component);
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
public override DesignerActionListCollection ActionLists
{
    get
    {
        DesignerActionListCollection al = new
DesignerActionListCollection();
        al.AddRange(base.ActionLists);
        al.Add(new ActionListForSmartTag(ctrl));
        return al;
    }
}
```

Ceux qui ne sont pas habitués à utiliser le designer de Visual Studio, penseront à référencer l'assembly « System.Design.dll » dans leur projet.

Rien de particulier par rapport à ce que l'on faisait en ASP.NET 1.1 (on dérive de ControlDesigner) si ce n'est l'apparition de cette nouvelle propriété « ActionLists » qui référence les actions disponibles dans le designer.

Nous allons voir maintenant notre DesignerActionList (en fait de quoi va être constitué notre smart tag) :

```
public class ActionListForSmartTag : DesignerActionList
{
    private ControlAvecSmartTag _monControl;

    public ActionListForSmartTag(ControlAvecSmartTag monControl)
        : base(monControl)
    {
        _monControl = monControl;
    }

    public void WinFormForText()
    {
        //C'est bien un appel à une winform !
        FormText form = new FormText(_monControl.Text);
        if (form.ShowDialog() == DialogResult.OK)
        {
            Texte = form.Texte;
        }
    }

    public System.Drawing.Color BackColorProperty
    {
        get
        {
            return _monControl.Couleur;
        }

        set
        {
            if (_monControl != null)
            {
                PropertyDescriptor pd =
TypeDescriptor.GetProperties(_monControl)[ "Couleur" ];
                if (pd != null)
                    pd.SetValue(_monControl, value);
            }
        }
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

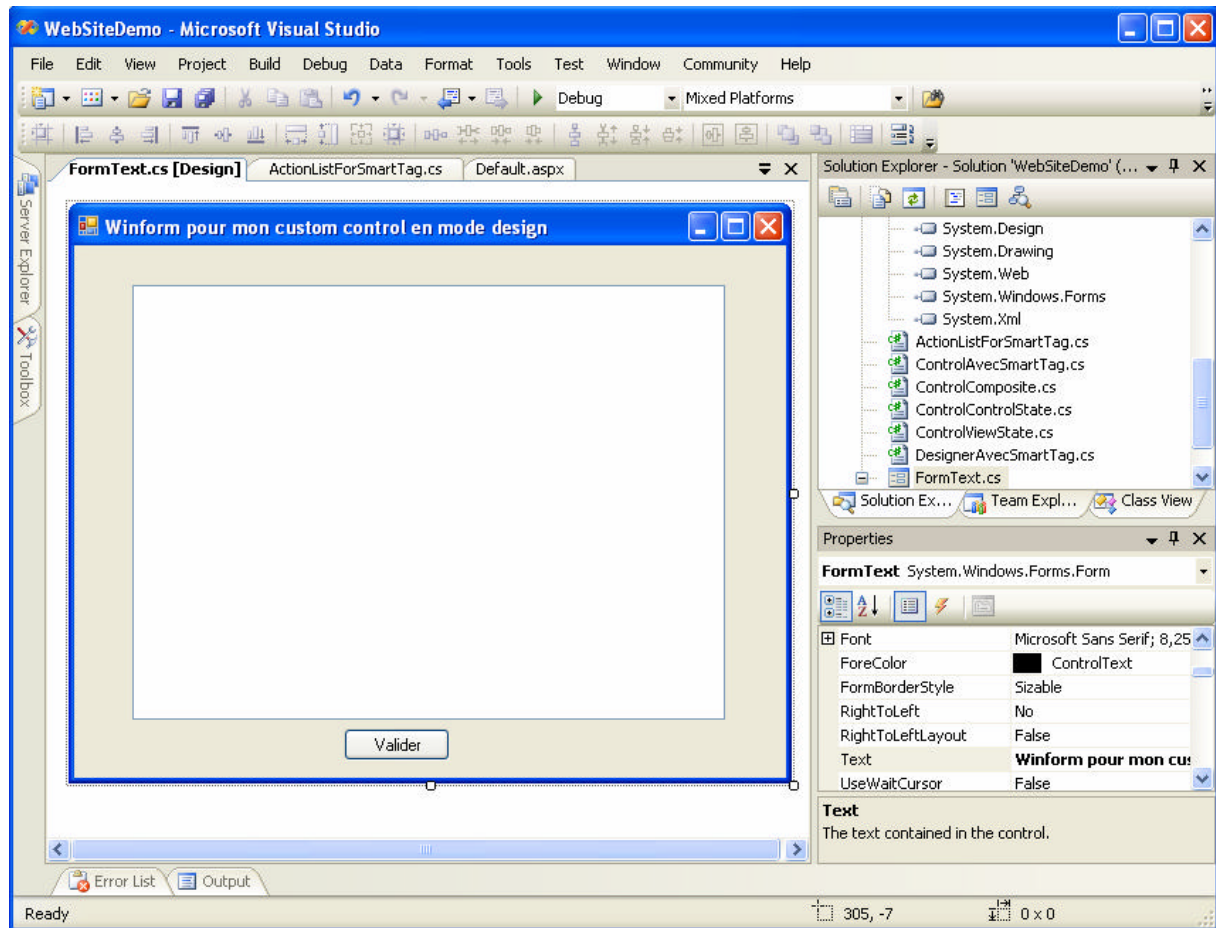
```
    }  
    }  
  
    public string Texte  
    {  
        get  
        {  
            return _monControl.Text;  
        }  
        set  
        {  
            if (_monControl != null)  
            {  
                PropertyDescriptor pd =  
TypeDescriptor.GetProperties(_monControl)[ "Text" ];  
                if (pd != null)  
                    pd.SetValue(_monControl, value);  
            }  
        }  
    }  
  
    public override DesignerActionItemCollection  
GetSortedActionItems()  
    {  
        DesignerActionItemCollection ai = new  
DesignerActionItemCollection();  
        ai.Add(new DesignerActionHeaderItem("Liste des actions  
disponibles"));  
        ai.Add(new DesignerActionMethodItem(this,  
"WinFormForText", "Changer le texte du contrôle", false));  
        ai.Add(new  
DesignerActionPropertyItem("BackColorProperty", "Couleur de fond :"));  
        return ai;  
    }  
}
```

Pensez à référencer l'assembly « System.Windows.Forms.dll » dans votre projet. Et oui, nous faisons appel à une Winform !

Je vous laisse créer la winform suivant ce modèle :

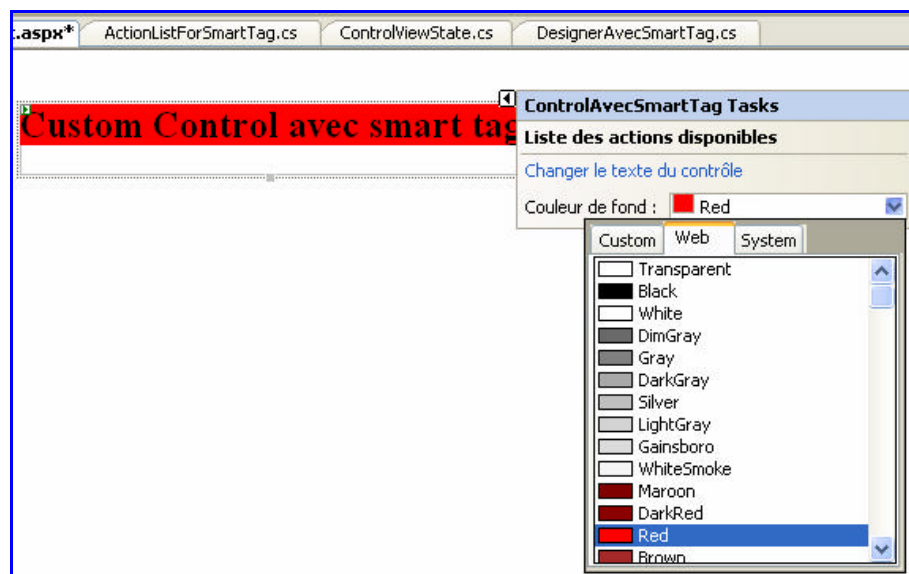
.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls



Une textbox, un bouton pour notre exemple simple.

La redéfinition de `GetSortedActionItem` va nous permettre de générer la liste des actions disponibles pour notre smart tag. Nous avons choisi plusieurs types d'actions : de l'affichage via `DesignerActionHeaderItem`, de lancer une méthode via `DesignerActionMethodItem`, de modifier une propriété du control via `DesignerActionPropertyItem`.



.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

Notez que l'on passe par un TypeDescriptor pour affecter la propriété du contrôle. Il faut bien comprendre que l'on pourrait parfaitement modifier la propriété du contrôle directement, le changement par exemple de couleur serait bien pris en compte et répercuté dans l'éditeur de propriétés du contrôle. Mais le changement ne serait pas visible dans le designer de Visual Studio. C'est pourquoi, on passe par l'intermédiaire du TypeDescriptor.

7. Mise en application : un control avec coins arrondis

Lors des DevDays 2005, dans le stand communautaire, Aurélien Verla, expert reconnu de la Communauté ASP.NET, faisait une démonstration « off-line » des possibilités des feuilles de style et de la puissance du javascript. Impressionnés, nous le fûmes tous. Il nous indiqua un lien sur Internet dont je mis beaucoup de temps à me souvenir. L'article précisait comment réaliser des coins arrondis sur des blocs de texte sans utiliser d'images.

Plus de précisions ici :

<http://pro.html.it/esempio/nifty/>

C'est en écrivant cet article que je me suis dit qu'un contrôle permettant d'écrire des blocs de texte à coins arrondis sans l'utilisation d'images pourrait résumer ce que nous avons vu précédemment. Nous allons donc construire ce contrôle de la façon la plus simple possible, toujours dans un souci pédagogique. Nul doute que vous apporterez de très grandes améliorations à celui-ci. Je n'ai pas non plus étudié en profondeur l'article « Nifty Corners », Aurélien Verla trouvera donc ici sujet à améliorations.

Nous pouvons reprendre une partie du code vu précédemment. Nous garderons intégralement les classes suivantes :

DesignerAvecSmartTag et ActionListForSmartTag.

Nous allons garder le squelette de la classe ControlAvecSmartTag et porter les améliorations suivantes :

- Prise en compte du Control State pour la propriété Couleur
- Intégration d'une feuille css en WebResource permettant d'arrondir les angles
- Mise en place des quelques tags nécessaires pour les coins arrondis

Ajoutons un fichier css à notre projet : styleCorners.css
Dont le contenu est le suivant :

```
B.rtop {  
    DISPLAY: block; BACKGROUND: #fff  
}  
B.rbottom {  
    DISPLAY: block; BACKGROUND: #fff
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
}
B.rtop B {
    DISPLAY: block; OVERFLOW: hidden; HEIGHT: 1px
}
B.rbottom B {
    DISPLAY: block; OVERFLOW: hidden; HEIGHT: 1px
}
B.r1 {
    MARGIN: 0px 5px
}
B.r2 {
    MARGIN: 0px 3px
}
B.r3 {
    MARGIN: 0px 2px
}
B.rtop B.r4 {
    MARGIN: 0px 1px; HEIGHT: 2px
}
B.rbottom B.r4 {
    MARGIN: 0px 1px; HEIGHT: 2px
}
```

Nous mettons le fichier en Ressources incorporées. Puis, nous allons en faire une Web Resource. Pour cela, il faut modifier le fichier AssemblyInfo.cs :

```
using System.Web.UI;

[assembly: WebResource("WebSiteCornersLibrary.styleCorners.css",
"text/css")]
[assembly: AssemblyTitle("WebSiteCornersLibrary")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("WebSiteCornersLibrary")]
[assembly: AssemblyCopyright("Copyright © 2005")]
```

Il nous reste plus qu'à adapter notre squelette ControlAvecSmartTag comme ceci :

```
[Designer(typeof(DesignerAvecSmartTag)), ParseChildren(true),
PersistChildren(true)]
public class ControlAvecSmartTag : CompositeControl
{
    private HtmlGenericControl _bloc;
    private Label _label;
    private Literal _header;
    private Literal _footer;

    /*-----*/
    /*-----Prise en compte du Control State -----*/
    /*-----*/
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        Page.RegisterRequiresControlState(this);
    }

    protected override void LoadControlState(object savedState)
    {
```


.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
    }

    protected override void CreateChildControls()
    {
        base.CreateChildControls();

        _bloc = new HtmlGenericControl("DIV");
        _header = new Literal();
        _label = new Label();
        _footer = new Literal();

        _header.Text = ChangeCorner(PositionCorner.top,
_couleur);
        _label.Text = _text;
        _footer.Text = ChangeCorner(PositionCorner.bottom,
_couleur);

        _bloc.Style[HtmlTextWriterStyle.BackgroundColor] =
_couleur.ToKnownColor().ToString();
        _bloc.Style[HtmlTextWriterStyle.Margin] = "0px 10%";

        _bloc.Controls.Add(_header);
        _bloc.Controls.Add(_label);
        _bloc.Controls.Add(_footer);
    }

    protected override void Render(HtmlTextWriter output)
    {
        EnsureChildControls();
        _bloc.RenderControl(output);
    }

    /*----- */
    /*-----Les propriétés----- */
    /*----- */
    private string _text = "<H1>Mon Label</H1>";
    public string Text
    {
        get { return _text; }
        set
        {
            _text = value;
            EnsureChildControls();
            _label.Text = _text;
        }
    }

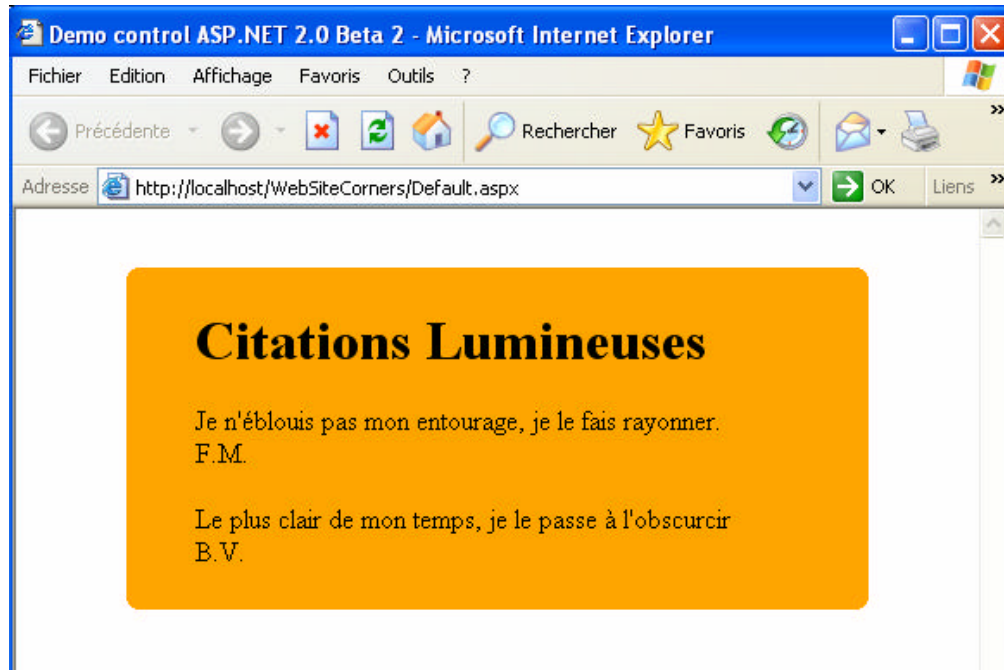
    private Color _couleur = Color.Aqua;
    public Color Couleur
    {
        get
        {
            return _couleur;
        }
        set
        {
            _couleur = value;
            EnsureChildControls();
            _bloc.Style[HtmlTextWriterStyle.BackgroundColor] =
_couleur.ToKnownColor().ToString();
        }
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

```
        _header.Text = ChangeCorner( PositionCorner.top,
_couleur);
        _footer.Text = ChangeCorner( PositionCorner.bottom,
_couleur);
    }
}
```

Une compilation puis l'intégration de la référence de l'assembly dans la boîte à outils de Visual Studio et l'ajout de notre contrôle dans une page, nous voici avec le rendu suivant dans le navigateur :



Je suis parfaitement conscient que le contrôle est sujet à amélioration mais mon but était purement pédagogique en vous le présentant.

Je tiens vivement à remercier Aurélien Verla qui involontairement m'a permis de trouver un exemple d'illustration des quelques nouveautés que je vous ai exposé à travers cet article.

Nous avons exposé ici qu'une petite partie des nouveautés des Custom Controls d'ASP.NET 2.0. Il y a matière à écrire d'autres articles sur le sujet.

8. En savoir plus

ASP.NET 2.0 Beta 2 and Visual Web Developer Beta 2
<http://msdn.microsoft.com/asp.net/beta2/default.aspx>

Speed Up Your Site with the Improved View State in ASP.NET 2.0
<http://msdn.microsoft.com/msdnmag/issues/04/10/ViewState/default.aspx>

.NET passionnément, tout simplement

ASP.NET 2.0 : Quelques nouveautés sur les Custom Controls

Handling Client Files in ASP.NET Whidbey

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/webresource.asp>

Nifty Corners: rounded corners without images

<http://pro.html.it/esempio/nifty/>

9. Notes

Certains produits mentionnés ne sont pas encore commercialisés. Ils sont en phase de test. Si vous souhaitez obtenir Visual Studio 2005 en version beta ou en version finale dès sa disponibilité, vous pouvez souscrire un abonnement MSDN

<http://www.microsoft.com/france/msdn/abonnements/presentation.asp>

10. Remerciements

Aurélien Verla (Merci pour ton partage de connaissances aux devdays 2005)

<http://blogs.developpeur.org/aurelien/>

Cyril Durand (Merci pour ta relecture précieuse et ton avis)

<http://blogs.developpeur.org/cyril/>

Merci à Elise, Laurent, Sébastien, Nicolas pour leur relecture.

N'hésitez pas à me contacter :

Frédéric Mélantois

Email : fmelantois@free.fr

Blog : <http://blog.developpeur.org/tkfe/>