

## Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

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. Faire du vieux avec du neuf

#### Du vieux code...

Vous avez certainement mis en place sur vos sites ASP, PHP, Coldfusion, pour la partie Back-Office, des pseudo-composants (je les appelle ainsi car il y avait tout un système d'includes très loin de la notion d'encapsulation) WYSIWYG pour apporter plus de richesses à des clients avertis (avertis des limites de tels outils...). Un certain nombre d'entre nous se sont basés sur l'Active X dhtmléd pour fabriquer leur propre pseudo-composant. Le principe était simple : on plaçait un Active X sur sa page et on dialoguait avec, via des fonctions javascript. A la soumission du formulaire, toujours par javascript, on plaçait le contenu de l'Active X dans un champ caché. On récupérait alors le contenu côté serveur via ce champ caché.

#### Faire du neuf...

L'idée est de réutiliser ce vieux code (en particulier, toute la fastidieuse plomberie javascript) afin d'éviter d'acheter des composants .NET, et le cas échéant de faire évoluer notre nouveau contrôle en fonction des besoins. Nous verrons que bons nombres de composants payants n'implémentent même pas certaines fonctionnalités basiques que je me propose de vous présenter à travers cet article. Nous allons donc essayer de créer un webcontrol (que nous pourrons ajouter à la boîte à outils (ToolBox) de Visual Studio NET). Vous allez voir que cela est beaucoup plus simple qu'il n'y paraît.

Je peux recommander la lecture d'un de mes précédents articles afin de ne pas débiter :

Construisez vos premiers contrôles serveur ASP.NET  
( voir <http://www.dotnet-tech.com/tutoriels/> )

### 2. DEMO 1 : Mise en place des propriétés, rendu en fonction des navigateurs clients

#### Définissons une interface pour les propriétés

Notre contrôle a besoin pour l'essentiel des propriétés suivantes :

- une largeur
- une hauteur
- une valeur (le contenu Html)
- un chemin virtuel pour les images que nous utiliserons pour proposer à l'utilisateur les fonctionnalités du composant.

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

- Le ClientID que nous utiliserons abondamment pour nommer nos éléments Html
- un texte (le contenu nettoyé du code Html)

L'idéal est d'utiliser une interface que notre contrôle va implémenter. Cette interface nous facilitera le développement par la suite.

### C#

```
internal interface IProperties
{
    System.Web.UI.WebControls.Unit Width
    {get; set;}

    System.Web.UI.WebControls.Unit Height
    {get; set;}

    string Value
    {get; set;}

    string VirtualPathComponent
    {get; set;}

    string TextOnly
    {get;}

    string ClientID
    {get;}
}
```

### VB

```
Friend Interface IProperties

    ReadOnly Property ClientID() As String

    Property Height() As System.Web.UI.WebControls.Unit

    ReadOnly Property TextOnly() As String

    Property Value() As String

    Property VirtualPathComponent() As String

    Property Width() As System.Web.UI.WebControls.Unit

End Interface
```

A noter que nous ne surchargerons pas « ClientID ». Nous n'avons pas à modifier son comportement.

### Evaluation du navigateur client

Comme nous utilisons un Active X, pour la sortie cliente, il nous faut déterminer les possibilités du navigateur client, afin que notre contrôle s'adapte à celui-ci. La classe « System.Web.HttpBrowserCapabilities » du framework va nous offrir le matériel nécessaire à cette évaluation.

Pour que notre contrôle fonctionne correctement côté client, il faut que le client :

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

- possède Internet Explorer
- que sa version majeure soit au moins la version 5
- que le navigateur autorise les Active X
- que le javascript fonctionne

### C#

```
public static bool IsCompatible(HttpContext request)
{
    System.Web.HttpBrowserCapabilities capacity = request.Browser;
    if ((capacity.Browser.ToUpper().IndexOf("IE") > -1)
        && (capacity.MajorVersion > 4)
        && (capacity.ActiveXControls)
        && (capacity.JavaScript)
        && (capacity.Win32)
        )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

### VB

```
Public Shared Function IsCompatible(ByVal request As HttpContext) As_
Boolean
    Dim capacity As HttpBrowserCapabilities = request.Browser
    If (capacity.Browser.ToUpper.IndexOf("IE") > -1 And _
        capacity.MajorVersion > 4 And _
        capacity.ActiveXControls And _
        capacity.JavaScript And _
        capacity.Win32) Then
        Return True
    End If
    Return False
End Function
```

Si notre client ne possède pas ces pré-requis, est-ce pour autant que le notre contrôle ne rendra rien ? Non, nous allons proposer une alternative : au moins une balise « textarea ».

Comment peut-on rendre le code plus modulaire, car nous n'allons pas développer plusieurs versions de notre contrôle ? Le pattern « Stratégie » semble être adapté à notre cas.

#### Le pattern « Stratégie »

Ce pattern n'est pas des plus compliqués. Il nous faut définir une interface « IHtmlTextWriter » comportant une méthode qui écrit le rendu du contrôle et deux chaînes contenant les blocs javascript nécessaire au fonctionnement du contrôle côté client.

Nous avons estimé la nécessité de distinguer deux types de blocs javascript :

- le javascript commun à toute multiplication de notre contrôle sur une même page. Nous l'avons appelé « WriteGeneralJavascriptBlock ».

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

- le javascript spécifique à chaque implémentation du contrôle dans la page, que nous avons nommé « WriteSpecificJavascriptBlock ».

### C#

```
internal interface IHtmlTextWriterStrategy
{
    void WriteComponent(IProperties prop, ref HtmlTextWriter writer);

    string WriteGeneralJavascriptBlock(IProperties prop);

    string WriteSpecificJavascriptBlock(IProperties prop);
}
```

### VB

```
Friend Interface IHtmlTextWriterStrategy
    Sub WriteComponent(ByVal prop As IProperties, _
ByRef writer As HtmlTextWriter)
        Function WriteGeneralJavascriptBlock(ByVal prop As IProperties)_
As String
            Function WriteSpecificJavascriptBlock(ByVal prop As IProperties)_
As String
        End Interface
```

Nous allons maintenant créer deux classes implémentant cette interface. Une des classes nous permettra d'exprimer le rendu du contrôle pour un navigateur « compatible », l'autre pour le reste de la population des navigateurs clients. « HtmlTextWriterWYSIWYG » nous fournira le rendu et le code javascript nécessaire lorsque la configuration cliente sera favorable. « HtmlTextWriterTEXT » traitera des autres cas.

Ensuite, il nous faut décider (donc définir une stratégie) quel code sera utilisé en fonction du navigateur. La classe « Rendu » fait la décision.

### C#

```
internal class Rendu
{
    private IHtmlTextWriterStrategy strategy;

    public Rendu(HttpRequest request)
    {
        if (BrowserForComponent.IsCompatible(request))
        {
            strategy = new HtmlTextWriterWYSIWYG();
        }
        else
        {
            strategy = new HtmlTextWriterTEXT();
        }
    }
}
```

### VB

```
Friend Class Rendu

    Private strategy As IHtmlTextWriterStrategy
    Public Sub New(ByVal request As HttpRequest)
        If BrowserForComponent.IsCompatible(request) Then
```

## .NET passionnément, tout simplement

Construire un contrôleur Wysiwyg, 1<sup>ère</sup> partie

```
        Me.strategy = New HtmlTextWriterWYSIWYG
    Else
        Me.strategy = New HtmlTextWriterTEXT
    End If
End Sub
```

Elle nous fournit le code nécessaire au rendu de notre contrôleur :

### C#

```
public void WriteComponent(IProperties prop, ref HtmlTextWriter writer)
{
    strategy.WriteComponent(prop, ref writer);
}

public string WriteGeneralJavascriptBlock(IProperties prop)
{
    return strategy.WriteGeneralJavascriptBlock(prop);
}

public string WriteSpecificJavascriptBlock(IProperties prop)
{
    return strategy.WriteSpecificJavascriptBlock(prop);
}
```

### VB

```
Public Sub WriteComponent(ByVal prop As IProperties, ByRef writer As_
HtmlTextWriter)
    Me.strategy.WriteComponent(prop, writer)
End Sub

Public Function WriteGeneralJavascriptBlock(ByVal prop As IProperties)_
As String
    Return Me.strategy.WriteGeneralJavascriptBlock(prop)
End Function

Public Function WriteSpecificJavascriptBlock(ByVal prop As IProperties)
As String
    Return Me.strategy.WriteSpecificJavascriptBlock(prop)
End Function
```

L'intérêt de cette façon de faire est que nous pouvons très facilement modifier notre stratégie d'accès à notre contrôleur par les navigateurs clients. Libre à vous de définir votre propre stratégie et d'assurer la compatibilité de votre code client avec les versions de navigateurs que vous aurez choisi d'implémenter.

Grâce à cette stratégie, le code suivant va opérer si le navigateur client est IE6 :

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

### C#

```
public void WriteComponent(IProperties prop,ref HtmlTextWriter writer)
{
    //écriture OBJECT
    writer.AddAttribute("height",prop.Height.ToString());
    writer.AddAttribute("width",prop.Width.ToString());
    writer.AddAttribute("classid", "clsid:2D360201-FFF5-11D1-8D03-00A0C959BC0A");
    writer.AddAttribute("class",prop.ClientID+"_nameHTML");
    writer.AddAttribute(HtmlTextWriterAttribute.Id,prop.ClientID+"_nameHTML");
    writer.RenderBeginTag(HtmlTextWriterTag.Object);
    writer.RenderEndTag();
}
```

### VB

```
Public Sub WriteComponent(ByVal prop As IProperties, ByRef writer As System.Web.UI.HtmlTextWriter) Implements _
IHtmlTextWriterStrategy.WriteComponent
    writer.AddAttribute("height", prop.Height.ToString)
    writer.AddAttribute("width", prop.Width.ToString)
    writer.AddAttribute("classid", "clsid:2D360201-FFF5-11D1-8D03-00A0C959BC0A")
    writer.AddAttribute("class", (prop.ClientID & "_nameHTML"))
    writer.AddAttribute(HtmlTextWriterAttribute.Id, (prop.ClientID & "_nameHTML"))
    writer.RenderBeginTag(HtmlTextWriterTag.Object)
    writer.RenderEndTag()
End Sub
```

S'il s'agit de Netscape 4.7 par exemple, le contrôle exécutera le code suivant :

### C#


```
public void WriteComponent(IProperties prop,ref HtmlTextWriter writer)
{
    //écriture TEXTAREA
    writer.AddAttribute(HtmlTextWriterAttribute.Id,prop.ClientID+"_id");
    writer.AddAttribute(HtmlTextWriterAttribute.Name,prop.ClientID+"_name");
    //pour garder des proportions de textarea similaires avec notre composant sous Internet Explorer
    writer.AddAttribute(HtmlTextWriterAttribute.Rows,DivisionBy((int)prop.Height.Value,13).ToString());
    writer.AddAttribute(HtmlTextWriterAttribute.Cols,DivisionBy((int)prop.Width.Value,9).ToString());
    writer.AddAttribute(HtmlTextWriterAttribute.Wrap,"VIRTUAL");//pour netscape 4 ;- )
    writer.RenderBeginTag(HtmlTextWriterTag.Textarea);
    writer.Write(prop.Value);
    writer.RenderEndTag();
}

private int DivisionBy(int number,int by)
{
    return (int) (number/by);
}
```

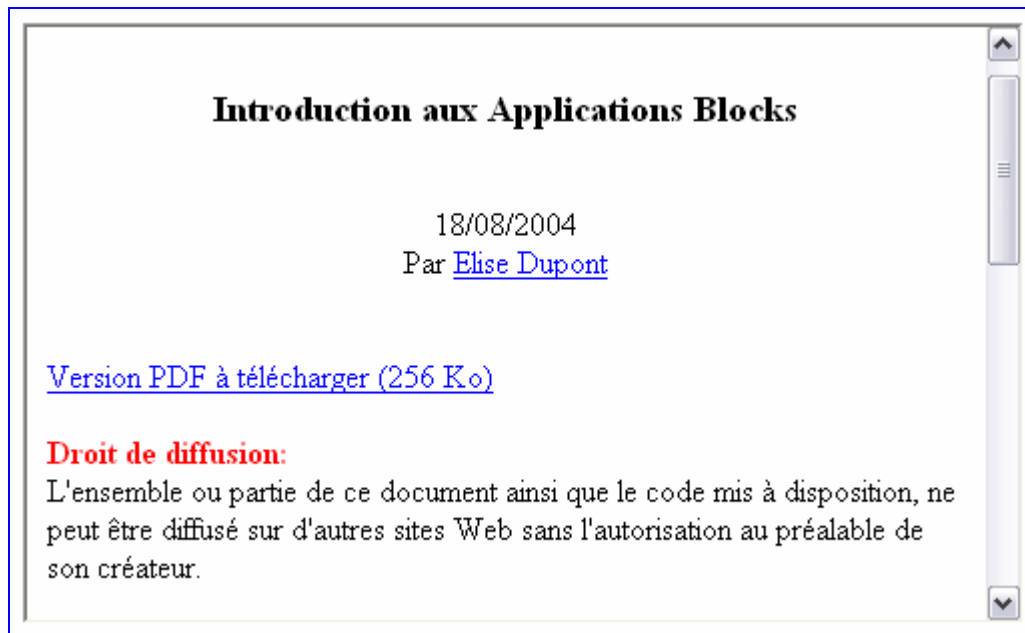
### VB

```
Public Sub WriteComponent(ByVal prop As IProperties, ByRef writer As HtmlTextWriter) Implements _
IHtmlTextWriterStrategy.WriteComponent
    writer.AddAttribute(HtmlTextWriterAttribute.Id, (prop.ClientID & "_id"))
    writer.AddAttribute(HtmlTextWriterAttribute.Name, (prop.ClientID & "_name"))
    Dim num1 As Integer = Me.DivisionBy(CType(prop.Height.Value, Integer), 13)
    writer.AddAttribute(HtmlTextWriterAttribute.Rows, num1.ToString)
    Dim num2 As Integer = Me.DivisionBy(CType(prop.Width.Value, Integer), 9)
    writer.AddAttribute(HtmlTextWriterAttribute.Cols, num2.ToString)
    writer.AddAttribute(HtmlTextWriterAttribute.Wrap, "VIRTUAL")
    writer.RenderBeginTag(HtmlTextWriterTag.Textarea)
    writer.Write(prop.Value)
    writer.RenderEndTag()
End Sub

Private Function DivisionBy(ByVal number As Integer, ByVal by As Integer) As Integer
    Return (number / by)
End Function
```

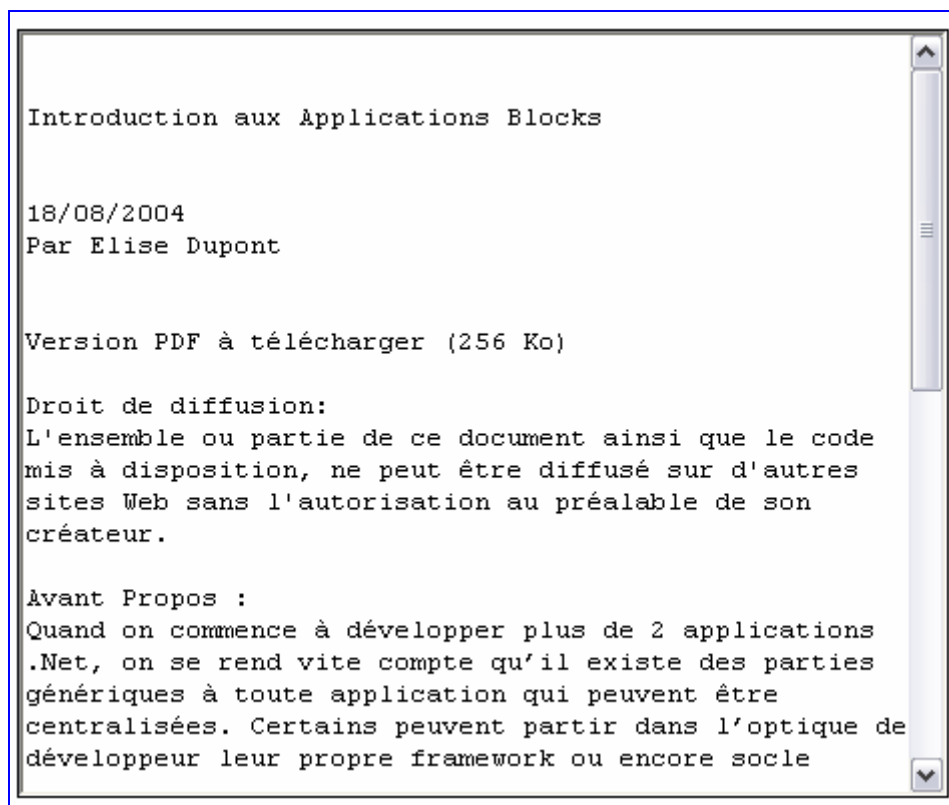
 Test de notre rendu

Sous Internet Explorer 6 :



Un glisser-déposer d'une partie d'une page HTML donne un rendu HTML dans le contrôle. On peut aussi agir sur ce texte grâce aux combinaisons de touches du clavier (par exemple : Ctrl + i pour mettre le texte en italique).

Sous Netscape 4.7 :



## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

Un glisser-déposer d'une partie d'une page HTML donne un rendu seulement texte dans le contrôle.

### 3. Demo 2 : Récupérer les valeurs clientes du côté serveur

Comment allons nous récupérer le Html ? Nous avons deux problématiques :

La première est que la balise object n'est pas récupérable.

La deuxième est qu'il faut faire en sorte que notre contrôle serveur récupère la valeur.

Quand nous utilisons la balise object chez le client, il nous faut prévoir un champ caché que nous pourrons récupérer côté serveur, un champ de type « textarea ». Cela permet de rendre récupérable la valeur. Nous n'avons qu'à l'ajouter à la méthode « WriteComponent » de la classe « HtmlTextWriterWYSIWYG » :

#### C#

```
//écriture du TEXTAREA caché
writer.AddAttribute(HtmlTextWriterAttribute.Id,prop.ClientID+"_id");
writer.AddAttribute(HtmlTextWriterAttribute.Name,prop.ClientID+"_name");
writer.AddAttribute(HtmlTextWriterAttribute.Rows,"1");
writer.AddAttribute(HtmlTextWriterAttribute.Cols,"1");
writer.AddAttribute("style","PADDING-RIGHT: 0px; PADDING-LEFT: 0px; VISIBILITY: hidden; PADDING-BOTTOM: 0px");
writer.RenderBeginTag(HtmlTextWriterTag.Textarea);
writer.Write(prop.Value);
writer.RenderEndTag();
```

#### VB

```
writer.AddAttribute(HtmlTextWriterAttribute.Id, (prop.ClientID & "_id"))
writer.AddAttribute(HtmlTextWriterAttribute.Name, (prop.ClientID & "_name"))
writer.AddAttribute(HtmlTextWriterAttribute.Rows, "1")
writer.AddAttribute(HtmlTextWriterAttribute.Cols, "1")
writer.AddAttribute("style", "PADDING-RIGHT: 0px; PADDING-LEFT: 0px; VISIBILITY: hidden; PADDING-BOTTOM: 0px")
writer.RenderBeginTag(HtmlTextWriterTag.Textarea)
writer.Write(prop.Value)
writer.RenderEndTag()
```

#### Ajout du javascript nécessaire au contrôle

Nous allons devoir opérer le passage d'une valeur via du javascript côté client, de la balise « object » vers le « textarea ». Nous nous servons d'une fonction javascript que nous écrirons dans « WriteGeneralJavascriptBlock » de la classe « HtmlTextWriterWYSIWYG » :

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

### C#

```
public string WriteGeneralJavascriptBlock(IProperties prop)
{
    string s = @"
<script language=""javascript"">
function CopyHtmlToHidden(FormHidden, FormHtml)
{
    FormHidden.value = FormHtml.DOM.body.innerHTML;
}
</script>";
    return s;
}
```

### VB

```
Public Function WriteGeneralJavascriptBlock(ByVal prop As IProperties) As String Implements _
IHtmlTextWriterStrategy.WriteGeneralJavascriptBlock
    Dim s As String = ChrW(13) & ChrW(10) & "<script language=""javascript"">" & _
ChrW(13) & ChrW(10) & "function CopyHtmlToHidden(FormHidden, FormHtml)" & _
ChrW(13) & ChrW(10) & "{" & ChrW(13) & ChrW(10) & ChrW(9) & _
"FormHidden.value = FormHtml.DOM.body.innerHTML;" & _
ChrW(13) & ChrW(10) & "}" & ChrW(13) & ChrW(10) & "</script>"
    Return s
End Function
```

Il nous faut maintenant écrire le javascript sur la sortie cliente. En fait, ceci se fait exactement dans le pré-rendu du contrôle. Nous devons à cet effet surcharger la méthode « OnPreRender » du WebControl :

### C#

```
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
    Rendu r = new Rendu(this.Page.Request);
    //En identifiant le bloc par un même nom ,
    //permet de ne pas écrire 10 fois les mêmes scripts
    //si on a 10 composants dans la page.
    this.Page.RegisterClientScriptBlock("GeneralScript", r.WriteGeneralJavascriptBlock(this));
    //place la fonction javascript sur le OnSubmit du formulaire client
    this.Page.RegisterOnSubmitStatement(this.ClientID + "_OnSubmit",
        "CopyHtmlToHidden(document.forms[0]." + this.ClientID + @"_name,document.forms[0]." + this
```

### VB

```
Protected Overrides Sub OnPreRender(ByVal e As EventArgs)
    MyBase.OnPreRender(e)
    Me.Page.RegisterRequiresPostBack(Me)
    Dim rendu As New Rendu(Me.Page.Request)
    Me.Page.RegisterClientScriptBlock("GeneralScript", rendu.WriteGeneralJavascriptBlock(Me))
    Me.Page.RegisterOnSubmitStatement(Me.ClientID & "_OnSubmit", "CopyHtmlToHidden(document.forms[0]." & _
        Me.ClientID & "_name,document.forms[0]." & Me.ClientID & "_nameHTML);")
End Sub
```

« RegisterClientScriptBlock » permet de placer un bloc javascript dans la page. Le fait de lui donner un même nom assure que l'utilisation de plusieurs instances de notre contrôle génère le même javascript dans la page cliente.

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

Si le javascript que l'on souhaite implémenter est spécifique à l'instance de notre contrôle, on se servira par exemple du ClientID unique du contrôle.

Si l'on souhaite, exécuter un javascript à l'événement OnSubmit du formulaire client, on le fera grâce à la méthode « RegisterOnSubmitStatement ».

Pour l'initialisation de notre balise « Object » (en clair l'initialisation de notre contrôle côté client), il nous faut passer la valeur du textarea, que nous initialisons (voir le code de celui-ci), vers la balise « Object ». Cela se fait aussi par javascript. Comme nous devons être certains que le textarea et la balise Object soient définis avant l'exécution du javascript, nous plaçons sa définition donc son écriture après celles des balises « Object » et « Textarea ». Comme ceci :

### C#

```
writer.RenderBeginTag(HtmlTextWriterTag.Textarea);
writer.Write(prop.Value);
writer.RenderEndTag();

//javascript la balise OBJECT
//pour faire passer la valeur du textarea vers l' OBJECT
writer.RenderBeginTag(HtmlTextWriterTag.Script);
string sjavascript = @"
document.forms[0]."+prop.ClientID+"_nameHTML.DocumentHTML = window.document.forms[0]."+prop.ClientID+"_name.value;";
writer.Write(sjavascript);
writer.RenderEndTag();
```

### VB

```
writer.RenderBeginTag(HtmlTextWriterTag.Textarea)
writer.Write(prop.Value)
writer.RenderEndTag()

writer.RenderBeginTag(HtmlTextWriterTag.Script)
Dim sjavascript As String = "document.forms[0]." & prop.ClientID & _
"_nameHTML.DocumentHTML = window.document.forms[0]." & prop.ClientID & "_name.value;"
writer.Write(sjavascript)
writer.RenderEndTag()
```

Maintenant que nous avons réglé la première problématique, réglons la seconde, qui est de récupérer la valeur du « textarea » côté serveur.

### Récupération des données clientes

Il nous suffit d'implémenter l'interface « IPostBackDataHandler » sur notre webcontrol.

« LoadPostData » va nous permettre de récupérer les données postées :

### C#

```
public bool LoadPostData(string postDataKey, NameValueCollection postCollection)
{
    string chaine = postCollection[this.ClientID+"_name"].ToString();
    this.Value = chaine;
    return true;
}
```

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie

### VB

```
Public Overridable Shadows Function LoadPostData(ByVal postDataKey As String, _  
ByVal postCollection As NameValueCollection) As Boolean _  
Implements IPostBackDataHandler.LoadPostData  
    Dim chaine As String = postCollection.Item((Me.ClientID & "_name")).ToString  
    Me.Value = chaine  
    Return True  
End Function
```

Ici on réaffecte la propriété « Value » de notre contrôle avec la valeur du « textarea » postée (grâce à la collection de valeurs récupérés dont nous identifions l'élément grâce au « name » que nous avons attribué au « textarea » caché).

Pourtant ce n'est pas suffisant, car il est nécessaire « d'inscrire » notre contrôle dans la liste des contrôles de la page, gérants le « Post Back ». Ceci se fait grâce à la méthode « RegisterRequiresPostBack » :

### C#

```
protected override void OnPreRender (EventArgs e)  
{  
    base.OnPreRender (e);  
    Page.RegisterRequiresPostBack (this);  
}
```

### VB

```
Protected Overrides Sub OnPreRender (ByVal e As EventArgs)  
    MyBase.OnPreRender (e)  
    Me.Page.RegisterRequiresPostBack (Me)
```

Nous allons enfin pouvoir tester. A cet effet, n'oubliez pas de désactiver la sécurité de votre page aspx en mettant l'attribut de la directive Page :

***ValidateRequest="false"***

Si vous oubliez de placer cet attribut à false, vous obtiendrez une exception, car nous véhiculons en effet du contenu Html et que ceci n'est plus permis par défaut depuis la version 1.1 du Framework.

Je considère que vous n'affectez pas vraiment la sécurité de votre système si vous placez votre contrôle dans un back-office à accès très restreint. Il appartient à vous de prendre la décision...

Sinon, une alternative serait de traiter ce contenu Html juste avant le post (submit) via du javascript (fonction escape). Mais il faudrait à nouveau traiter côté serveur pour obtenir le code HTML.

Voici ce que nous obtenons après un postback (Ici nous plaçons dans une TextBox Multiligne la valeur de la propriété « Value » de notre contrôle :



The screenshot displays a web control with two main sections. The top section shows the rendered output of HTML code, and the bottom section shows the raw HTML code. A 'PostBack' button is located between the two sections.

**Introduction aux Applications Blocks**

18/08/2004  
Par [Elise Dupont](#)

[Version PDF à télécharger \(256 Ko\)](#)

**Droit de diffusion:**  
L'ensemble ou partie de ce document ainsi que le code mis à disposition, ne peut être diffusé sur d'autres sites Web sans l'autorisation au préalable de son créateur.

PostBack

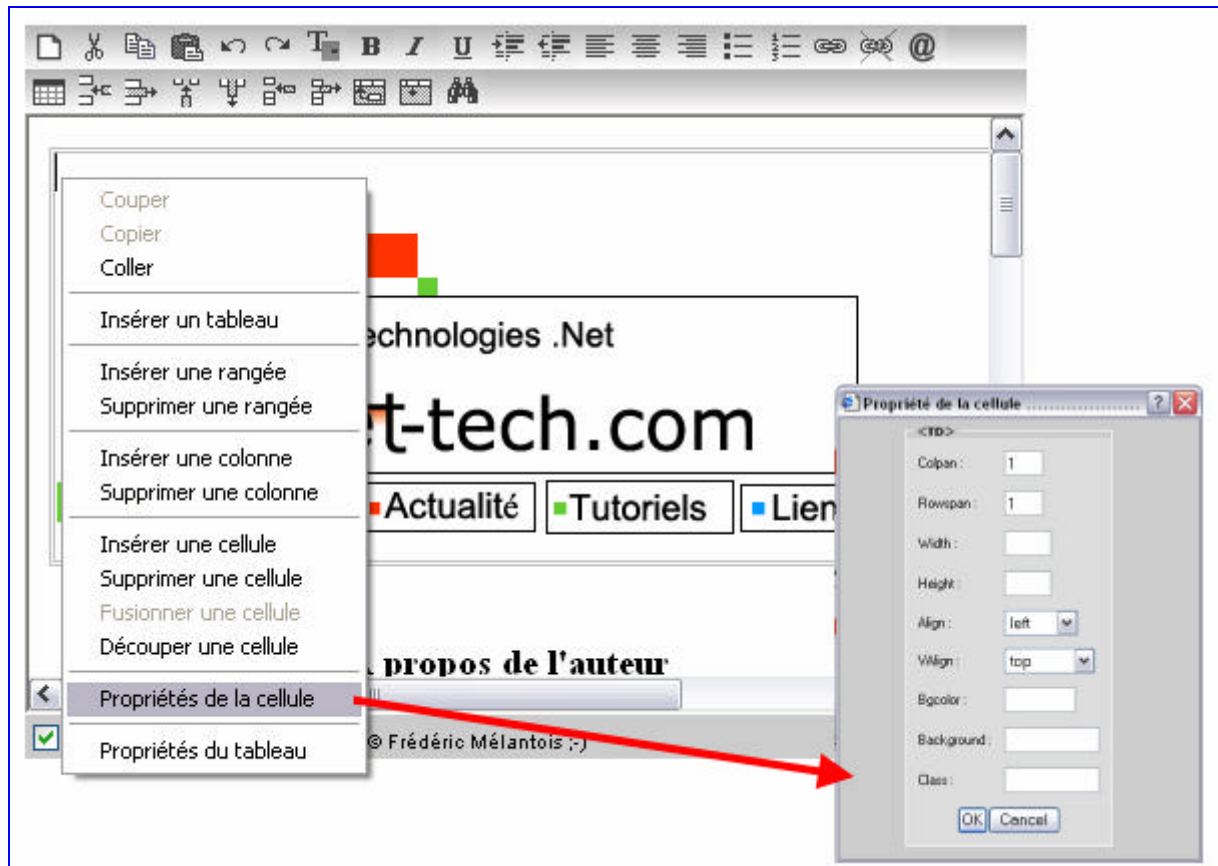
```
<BR>
<H3>
<DIV align=center>Introduction aux Applications
Blocks</DIV></H3><BR>
<P align=center>18/08/2004<BR>Par <A
href="mailto:elise@dotnet-fr.org">Elise Dupont</A></P><BR><A
href="application-blocks.pdf">Version PDF à télécharger (256
Ko)</A> <BR>
<P><FONT color=#ff0000><B>Droit de
diffusion:</B></FONT><BR>L'ensemble ou partie de ce document
ainsi que le code mis à disposition, ne peut être diffusé sur
d'autres sites Web sans l'autorisation au préalable de son
créateur.</P>
```

Nous poursuivrons dans le prochain article la construction de notre contrôlé. Nous personnalisons l'affichage de celui-ci dans le designer de Visual Studio. Nous traiterons aussi le code HTML renvoyé (nettoyage). Nous réaliserons une fonction de récupération du texte seul à partir du HTML (grand intérêt lors des recherches de mots en base de données). Enfin, nous mettrons en place tout la plomberie javascript (c'est là où je peine le plus ;-)

Et nous devrions avoir un rendu comme celui-ci :

## .NET passionnément, tout simplement

Construire un contrôle serveur Wysiwyg, 1<sup>ère</sup> partie



### 4. En savoir plus :

Design Patterns (remplacer les classes abstraites par des interfaces quand cela est possible)

<http://www.dofactory.com/Patterns/Patterns.aspx>

Working with Client-Side Script

<http://msdn.microsoft.com/asp.net/using/building/web/default.aspx?pull=/library/en-us/dnasp/html/ClientSideScript.asp>

Construisez vos premiers contrôles serveur ASP.NET

<http://www.dotnet-tech.com/tutoriels/>

Contactez l'auteur :

Frédéric Mélandois

Email : fmelantois[arobase]free.fr