

ASP.NET 2.0 : Webparts et personnalisation

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

Nous vous invitons à lire l'article « Sécurité ASP.NET 2.0 : authentification par formulaire », en particulier la mise en place d'un provider personnalisé.

1. Présentation rapide des Webparts

De quoi s'agit-il ?

Une WebPart est une zone de contenu complètement personnalisable. Nous avons la possibilité de la réduire, de la fermer, de la supprimer, de la cacher, ...

Le WebPartManager

Le WebPartManager permet de « manager » l'ensemble des WebParts de la page. Il est indispensable dans la page pour permettre les échanges de données entre WebParts. Il est recommandé de placer ce composant avant les autres genres du Framework des WebParts. Il n'est visible que dans le Designer de Visual Studio.

La WebPartZone

C'est une zone qui contient un certain nombre de WebParts. Il s'agit d'un réservoir de WebParts et même de contrôles serveur.

La CatalogZone

La CatalogZone permet de proposer de nouvelles WebParts à mettre dans une des WebPartZones de la page ou plus simplement, de « récupérer » les WebParts de la page que l'on aurait fermées.

L'EditorZone

L'EditorZone permet d'éditer une WebPart particulière. On pourra y changer tout l'aspect Design mais aussi les propriétés personnalisées de la WebPart.

2. Mise en place de Webparts et de la personnalisation

Notre souhait est de pouvoir personnaliser l'affichage des pages, en particulier l'agencement et le contenu des WebParts. Pour cela, il nous faut stocker quelque part cette information. Nous allons voir comment stocker cette information.

Stockage et personnalisation

La version Beta 1 du « Website Administration » ne permet pas d'indiquer quel « provider » servira pour la personnalisation des WebParts. Donc, par défaut, si vous ne renseignez pas le provider, celui par défaut sera l'AccessPersonalizationProvider qui pointera vers la base Access situé dans le répertoire Data de votre application Web.

Nous invitons le lecteur à lire l'article « Sécurité ASP.NET 2.0 : authentification par formulaire » et à préparer une authentification par formulaire avec un provider

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

personnalisé (On prendra par exemple SQL Server 2000). Vous définirez en particulier le MembershipProvider, le RoleProvider et le ProfileProvider.

http://www.dotnet-tech.com/tutoriels/asp.net_2_secu/

A ce stade, tous les éléments définis dans le web.config ne sont pas suffisants pour stocker la personnalisation de nos WebParts. Il va falloir donc préciser manuellement quel fournisseur nous souhaitons. Comme vous venez d'utiliser une base de données SQL Serveur 2000, nous allons paramétrer le site web pour que la base servant pour les MemberShips, Roles et Profile serve aussi pour la personnalisation des WebParts.

Voici donc l'instruction à placer dans le web.config :

```
<webParts>
  <personalization defaultProvider="sql01">
    <providers><add connectionString="webAdminConnection"
name="sql01"
type="System.Web.UI.WebControls.WebParts.SqlPersonalizationProvider,
System.Web, Version=2.0.3500.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a"/></providers>
  </personalization>
</webParts>
```

Nous indiquons que notre fournisseur par défaut est « sql01 ». Sans ces lignes, la personnalisation des WebParts sera stockée par le provider par défaut : à savoir une base Access pour la beta 1, une base Sql Server 2005 Express Edition pour la version finale des ASP.NET 2.0. Pour vous en convaincre, il vous suffira de mettre en commentaire ces lignes lorsque nous aurons implémenté nos premiers WebParts. Vous vous apercevrez alors que le stockage de cette personnalisation ne se fait plus au même endroit.

Nos premiers WebParts

Rappel : Le support de base de notre démonstration est le code produit dans l'article « Sécurité ASP.NET 2.0 : authentification par formulaire ».

Sous le répertoire « Admin », nous allons placer nos WebParts dans la page « Default.aspx ». Nous allons effectuer notre première prise de contact avec les WebParts via le « designer » de Visual Studio.

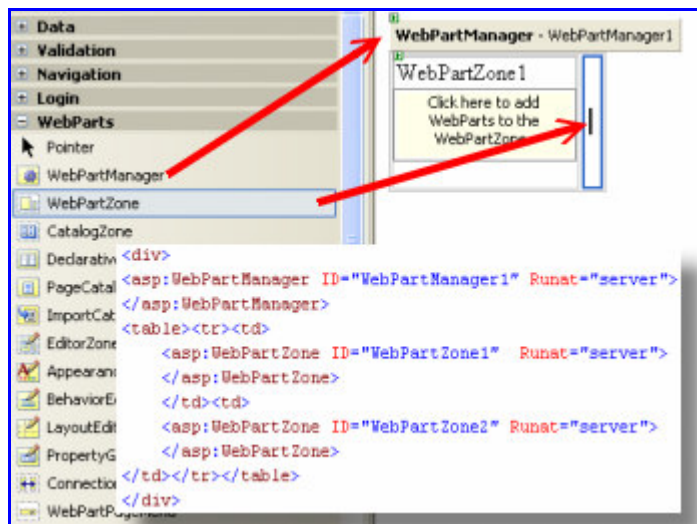
Plaçons tout d'abord un contrôle permettant de nous déconnecter de notre page. Il s'agit du nouveau contrôle LoginStatus :

```
<asp:LoginStatus ID="LoginStatus1" Runat="server"
LogoutPageUrl="~/Admin/Default.aspx" LogoutText="Se déconnecter" />
```

Plaçons ensuite nos WebParts. Nous retrouvons tous les composants définis au début de l'article dans la Boîte à Outils de Visual Studio 2005. Commençons par prendre de bonne habitude : On commence toujours par le composant WebPartManager (un seul par page). Puis plaçons deux WebPartZones dans la page.

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation



Il nous faut placer maintenant des WebParts dans les WebPartZones. Comme nous n'avons pas de WebParts, pour l'instant, à placer dans les WebPartZones, nous allons glisser des WebControls Calendar à partir de la Boîte à outils. En faisant cela, on s'aperçoit que le contrat nécessaire pour qu'un objet fasse partie du réservoir de la WebPartZone est que cet objet soit un control. Affectez des couleurs différentes à la propriété BackColor de vos Calendars.

Personnalisation et mode d'affichage « design »

Placez un bouton nommé « Mode Design » dont la méthode sur l'événement Click est la suivante :

C#

```
void Button1_Click(object sender, EventArgs e)
{
    this.WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode;
}
```

VB

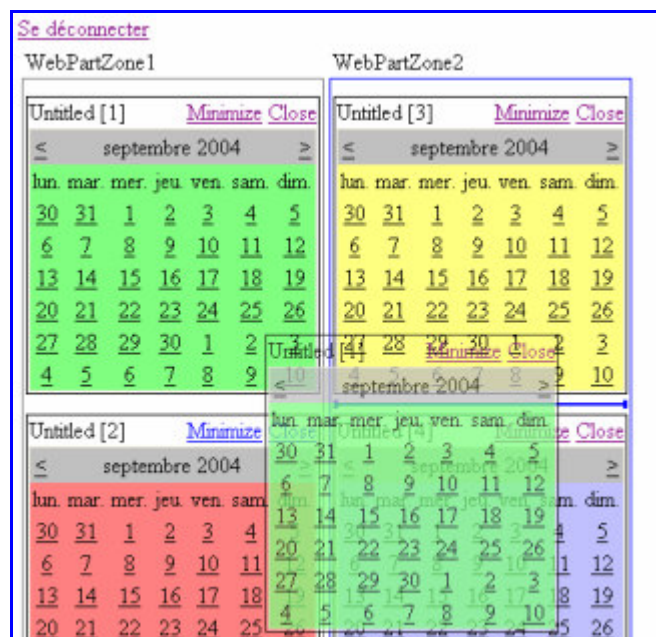
```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode
End Sub
```

Le WebPartManager que nous avons affecté à la page gère les modes d'affichages. Nous choisissons le mode DesignDisplayMode qui va offrir à nos utilisateurs finaux la possibilité de modifier le positionnement des différentes WebParts de la page.

Compilons notre application. Et lançons <http://localhost/SitePerso/Admin/> . Identifiez-vous avec le login et mot de passe d'un des utilisateurs que vous aurez défini auparavant. Une fois identifié, vous parviendrez à la page contenant vos WebParts Calendar. Essayez de déplacer les WebParts (à partir de la zone de titre), vous n'y parvenez pas. Passez en mode Design, en cliquant sur le bouton « Mode Design ». Essayez à nouveau de déplacer les WebParts. Vous y parvenez :

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation



Nous déplaçons le calendrier vert (la WebPart) où l'on souhaite dans les WebPartZone 1 ou 2.

Regardons si cette personnalisation de la page est conservée avec le temps. Cliquez sur le bouton « Se déconnectez » puis reconnectez-vous à nouveau avec le même login/mot de passe que précédemment. Vous retrouvez à nouveau la page comme vous l'aviez quittée.

Assurons-nous qu'il s'agit bien d'une personnalisation : Déconnectez-vous à nouveau et entrez un login/mot de passe différent. Vous retrouvez la page telle qu'elle était initialement.

Une visite dans la base de données confirme que notre page pour nos deux utilisateurs différents a bien été sauvegardée :

Données de la table « aspnet_PagePersonalizationPerUser » dans « SitePerso » sur «					
Id	PathId	UserId	PageSettings	LastUpdatedDate	
{820904F7-A95D-4E	{324C1FC5-0625-427A-97C	{5A79A92C-14DB-	<Binaire>	30/09/2004 19:03:	
{7A944643-AAC1-4	{324C1FC5-0625-427A-97C	{D9BE41AD-D5A3-	<Binaire>	30/09/2004 19:02:	
*					

Telles des fenêtres Windows, vous constatez qu'il est possible de fermer, de réduire, de restaurer une WebPart. Ces changements sont aussi conservés dans la table « aspnet_PagePersonalizationPerUser », en particulier dans le champ « PageSettings ». Une partie des informations y est sérialisée.

La gestion est intelligente puisque si nous supprimons un calendrier (WebPart) physiquement dans la page sous Visual Studio et intégrons un nouveau calendrier avec une nouvelle couleur de fond, la personnalisation des calendriers subsistants n'aura pas été perdue lors de la reconnexion sur la page. Il n'y aura pas non plus de plantage.

En est-il de même d'une WebPart insérée dynamiquement dans une WebPartZone ? Pour répondre à cette question, nous devons trouver la méthode permettant de le faire. La plus rapide à mettre en place est celle du WebPartManager, en particulier la méthode AddWebPart :

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
Visual Basic
Public Function AddWebPart( _
    ByVal webPart As WebPart, _
    ByVal zone As WebPartZoneBase, _
    ByVal zoneIndex As Integer _
) As Void

C#
public void AddWebPart(
    WebPart webPart,
    WebPartZoneBase zone,
    int zoneIndex
);
```

Nous constatons qu'il faut fournir une WebPart. Il n'est donc pas très compliqué de la construire :

C#

```
public class WebPartCalendar : WebPart
{
    private Color _couleurCalendar = Color.Coral;
    public WebPartCalendar(string title, Color couleur)
    {
        this.Title = title;
        _couleurCalendar = couleur;
    }

    protected override void CreateChildControls()
    {
        Calendar cal = new Calendar();
        cal.BackColor = _couleurCalendar;
        Controls.Add(cal);
    }

    public WebPartCalendar()
    {
        this.Title = "WebPart";
    }
}
```

VB

```
Public Class WebPartCalendar
    Inherits WebPart
    Private _couleurCalendar As Color
    Public Sub New(ByVal title As String, ByVal couleur As Color)
        Me._couleurCalendar = Color.Coral
        Me.Title = title
        Me._couleurCalendar = couleur
        Me.ExportMode = WebPartExportMode.All
    End Sub

    Protected Overrides Sub CreateChildControls()
        Dim calendar1 As New Calendar
        calendar1.BackColor = Me._couleurCalendar
        Me.Controls.Add(calendar1)
    End Sub

    Public Sub New()
        Me._couleurCalendar = Color.Coral
    End Sub
End Class
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
Me.Title = "WebPart"  
End Sub  
End Class
```

Ajoutons un bouton à notre page qui va nous permettre d'ajouter notre nouvelle WebPart de façon dynamique. L'ajout de la WebPart se fait comme ceci :

C#

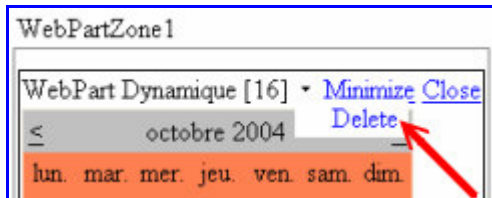
```
void Button2_Click(object sender, EventArgs e)  
{  
    WebPartCalendar wc = new WebPartCalendar("WebPart Dynamique",  
Color.Coral);  
    WebPartManager1.AddWebPart(wc, WebPartZone1, 0);  
}
```

VB

```
Private Sub Button2_Click(ByVal sender As Object, ByVal e As EventArgs)  
    Dim calendar1 As New WebPartCalendar("WebPart Dynamique", Color.Coral)  
    Me.WebPartManager1.AddWebPart(calendar1, Me.WebPartZone1, 0)  
End Sub
```

Compilons à nouveau notre projet. Essayons des connexions et déconnexions successives : nous constatons que les WebPart ajoutées dynamiquement sont aussi prises en compte pour la personnalisation.

Si vous cliquez sur le bouton « Mode Design » que nous avons précédemment défini pour permettre de passer en mode « DesignDisplayMode », vous vous apercevez que toutes les WebParts que vous avez instanciées dynamiquement possèdent un nouveau Verb :



Celui-ci permet de supprimer physiquement la WebPart considérée.

3. La gestion des Webparts

La GenericWebPart pour nos contrôles

Activez la trace de la page. Vous remarquerez que pour chaque control de type Calendar que nous avons placé dans une WebPartZone, une GenericWebPart a été générée.

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

WebPartManager1	System.Web.UI.WebControls.WebParts.WebPartManager
WebPartManager1\$gwpCalendar1	System.Web.UI.WebControls.WebParts.GenericWebPart
WebPartManager1\$gwpCalendar1\$Calendar1	System.Web.UI.WebControls.Calendar
WebPartManager1\$gwpCalendar3	System.Web.UI.WebControls.WebParts.GenericWebPart
WebPartManager1\$gwpCalendar3\$Calendar3	System.Web.UI.WebControls.Calendar
WebPartManager1\$gwpCalendar2	System.Web.UI.WebControls.WebParts.GenericWebPart
WebPartManager1\$gwpCalendar2\$Calendar2	System.Web.UI.WebControls.Calendar
WebPartManager1\$gwpCalendar5	System.Web.UI.WebControls.WebParts.GenericWebPart
WebPartManager1\$gwpCalendar5\$Calendar5	System.Web.UI.WebControls.Calendar

Les identifiants sont bien sûr uniques car la GenericWebPart dérive de la WebPart qui dérive de la Part qui implémente l'interface INamingContainer. Le WebPartManager implémente aussi cette interface.

🚦 Identification unique des WebParts et persistance

On constate que chaque WebPart possède un identifiant unique. Pour la WebPart statique, c'est-à-dire celle que vous avez déposée dans la ZoneTemplate d'une WebPartZone, l'identifiant unique est composé du nom du WebPartManager ainsi que du nom que vous avez donné à la WebPart. Pour ce qui est des WebParts dynamiques, on s'aperçoit, par exemple par la trace, que l'identifiant unique est composé du nom du WebPartManager et d'un nombre aléatoire unique. Y a-t-il persistance du nom de l'identifiant entre deux chargements de pages. La réponse est oui. Si nous nous déconnectons puis reconnectons, les identifiants uniques des WebParts ne changent pas, ce qui laisse penser que ces identifiants sont conservés dans le processus de personnalisation.

🚦 Quelques remarques sur la persistance des données

Il faut bien en comprendre les mécanismes. Placez un control TextBox dans une des WebPartZone que vous aurez défini. Comme nous l'avons vu, une GenericWebPart va être générée prenant en charge notre TextBox. Compilons l'application. Connectons-nous, et entrons une valeur dans la TextBox. Puis cliquons sur le bouton « Mode Design » pour passer en mode « DesignDisplayMode ». Un déplacement de notre WebPart ne fait pas perdre la valeur dans la TextBox. Pas de magie à ce niveau, le mécanisme s'explique très bien, la valeur est « réinjectée » dans la TextBox lors du rendu (il n'y a pas ici de mécanisme de ViewState, pour mieux comprendre, voir un de mes précédents articles « Construire un contrôle serveur Wisiwyg partie 1 » notamment, le chapitre « récupération des données clientes » en page 8). Par contre, si nous réduisons notre WebPart en cliquant sur « Minimize », puis la restaurons via « Restore », nous perdons la valeur de notre TextBox. Ceci est tout à fait normal puisque lorsque la WebPart est « minimisée » ou fermée, il n'y a pas de phase de rendu pour notre control TextBox. Continuons d'explorer la conservation d'état : Faisons la même chose avec un de nos controls calendar. Pour cela, changer de mois. Puis déplacez ou minimisez la GenericWebPart concernée. Le mois que vous aurez choisi est conservé. Pourquoi ? Grâce à la propriété enableViewState à true du control Calendar, les états du control sont conservés entre les diversPostBack de la page. Si nous nous déconnectons puis reconnectons, nous perdons l'état de notre control, on ne retrouve pas le mois que nous avons choisi. Le ViewState n'est pas pris en compte dans la personnalisation. Pour conserver les états de manière permanente, nous

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

devons déclarer les propriétés qui doivent être prises en compte par la personnalisation, ce que nous allons voir un peu plus loin dans cet article.

Le titre d'une WebPart

Vous avez pu remarquer que nos WebParts ont pour titre « Untitled » ce qui n'est pas du plus bel effet. Nous avons placé des controls « calendar » dans nos WebPartZones. Bien que des GenericWebPart aient pu être créées automatiquement comme nous l'avons vu précédemment, nous n'avons renseigné aucune propriété pour le titre. Le control Calendar ne contenant pas de propriété « Title », nous pouvons toujours lui ajouter cet attribut :







C#

```
Calendar1.Attributes["title"] = "Pas beau mon titre ?";
```

VB

```
Calendar1.Attributes.Item("title") = "Pas beau mon titre ?"
```

Pas idéal, me direz-vous. Mais c'est normal, notre control n'est pas prévu pour cela. Si j'utilise mes propres controls (User Controls ou Custom Controls), comment pouvoir bénéficier des propriétés communes des WebParts ? Tout simplement, en implémentant l'interface IWebPart.

	CatalogImageUrl
	Description
	Subtitle
	Title
	TitleImageUrl
	TitleUrl

Pour essayer, construisez un User Control et faites-le implémenter IWebPart. Placez-le ensuite dans une WebPartZone.

Modification du design et du comportement d'une WebPart

Un Verb de WebPart est une fonctionnalité tel que la fermeture (« Close »), la minimisation (« Minimize »), la suppression (« Delete »), l'aide (« Help »), La restauration (« Restore »). Pouvoir modifier le comportement des WebParts que nous avons mis en place se révèle compliqué puisque nous avons utilisé des Controls (Calendar) non prévus pour cela. De même qu'un User Control implémentant l'interface IWebPart, ne serait pas non plus facile à manipuler.

La façon idéale de pouvoir prédéfinir ou modifier les propriétés de design ou de comportements d'une WebPart est que cette WebPart hérite de la classe WebPart. Il n'existe malheureusement pas d'interface que nous pourrions implémenter afin de bénéficier de ces propriétés.

Si nous avons un Control, il va falloir trouver des astuces pour pouvoir manipuler le design et comportement de la WebPart. Comme nous l'avons vu précédemment, l'implémentation de IWebPart nous permet d'accéder seulement à quelques simples propriétés.

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Une première solution est de rechercher la GenericWebPart grâce à l'ID de notre control :

C#

```
WebPartManager1.WebParts["Calendar1"].AllowZoneChange = false;
```

VB

```
WebPartManager1.WebParts.Item("Calendar1").AllowZoneChange = False
```

Ici, il devient impossible pour l'utilisateur en phase de « DesignDisplayMode » de changer sa WebPart de WebPartZone.

Si nous ajoutons des WebParts de façon dynamique, nous pouvons agir sur la WebPart avant qu'elle ne soit ajoutée à la collection du WebPartManager de la page. Nous allons ajouter de façon dynamique un User Control :

C#

```
WebPartUserControlCalendar_ascx uc = new WebPartUserControlCalendar_ascx();  
uc.ID = "MonUserControl"; /*Indispensable de preciser l'ID sinon il y a une  
erreur*/  
GenericWebPart part = WebPartManager1.CreateWebPart(uc);  
part.AllowClose = false;  
WebPartManager1.AddWebPart(part, WebPartZone1, 0);
```

VB

```
Dim uc As New WebPartUserControlCalendar_ascx  
uc.ID = "MonUserControl" 'Preciser l'ID sinon il y a une erreur  
Dim part1 As GenericWebPart = WebPartManager1.CreateWebPart(uc)  
part1.AllowClose = False  
WebPartManager1.AddWebPart(part1, Me.WebPartZone1, 0)
```

Ici, le User Control est ajouté à la collection de WebParts de la WebPartZone. Comme nous l'avons vu, il y a création d'une GenericWebPart que nous récupérons pour la manipuler (empêcher la fermeture de la WebPart). A noter, qu'il est nécessaire d'avoir fourni un ID à notre control car celui-ci est utilisé comme nous l'avons vu pour créer l'ID de la GenericWebPart.

Si on parvient à savoir facilement si une WebPart est fermée par la propriété « IsClosed », comment savoir si notre WebPart est minimisée ou pas? Une propriété vient à notre secours : ChromeState

C#

```
foreach (WebPart part in WebPartManager1.WebParts)  
{  
    if (part.ChromeState == PartChromeState.Minimized)  
    {  
        part.ChromeState = PartChromeState.Normal;  
    }  
}
```

VB

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
For Each part1 In Me.WebPartManager1.WebParts
    If (part1.ChromeState = PartChromeState.Minimized) Then
        part1.ChromeState = PartChromeState.Normal
    End If
Next
```

4. La zone des catalogues de Webparts

La CatalogZone et la PageCatalogPart

En observant la trace de la page, vous allez aussi vous apercevoir que de nombreuses WebParts ont été instanciées dans votre page alors qu'elles ne sont pas visibles. Pour certaines, elles n'ont pas été « rendues ». Il s'agit des WebParts que vous avez fermées (verb Close). Comment les rendre à nouveau visible ?

Nous allons nous servir de la « CatalogZone » qui est comme son nom l'indique une zone contenant des catalogues de WebParts. Ajoutez une CatalogZone dans votre page (en dehors des WebPartZones), et placez dans sa « zonetemplate » une « PageCatalogPart » qui recense tous les WebParts) disponibles de la page, c'est-à-dire les WebParts fermées.

```
<asp:CatalogZone ID="CatalogZone1" Runat="server"
HeaderText="ZoneCatalogue des WebParts"
EmptyZoneText="La Zone Catalogue ne contient pas de WebPart"
SelectTargetZoneText="Ajouter à :">
    <HeaderCloseVerb Text="Fermer"/>
    <AddVerb Description="Ajouter une WebPart dans la Zone"
Text="Ajouter"/>
    <CloseVerb Description="Fermer le Catalogue" Text="Fermer"/>
    <ZoneTemplate>
        <asp:PageCatalogPart Runat="server" ID="PageCatalogPart1" />
    </ZoneTemplate>
</asp:CatalogZone>
```

Ajoutons un bouton « Mode catalogue » à notre page qui va nous permettre de changer de mode d'affichage et de pouvoir afficher notre CatalogZone :

C#

```
void Button3_Click(object sender, EventArgs e)
{
    this.WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode;
}
```

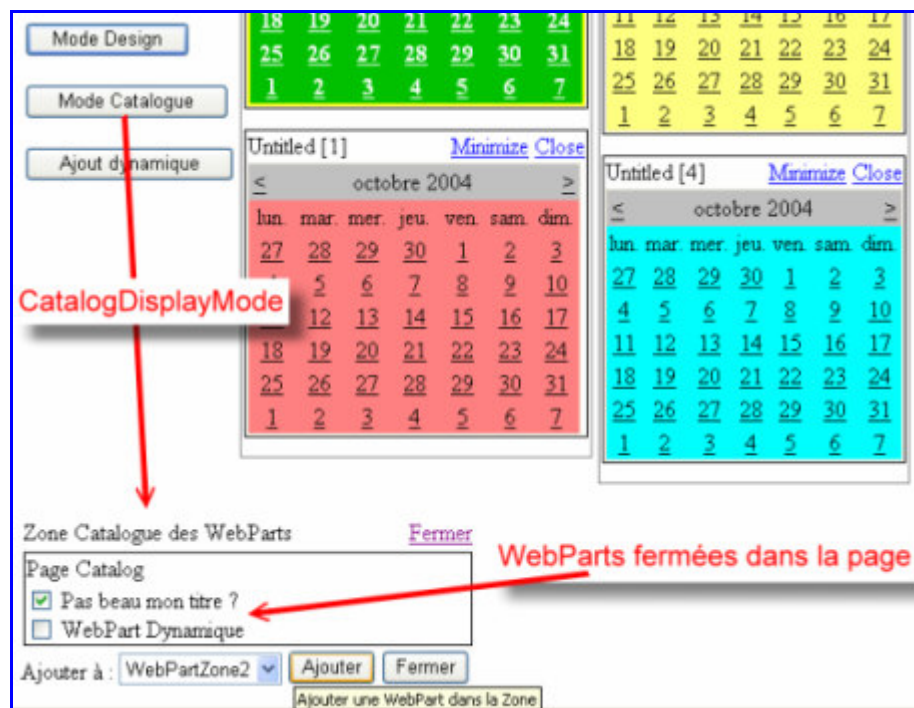
VB

```
Private Sub Button3_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode
End Sub
```

Compilons, identifions-nous et cliquons sur ce bouton, nous avons alors le rendu ci-dessous :

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation



Nous pouvons remarquer qu'il est tout à fait possible de déplacer, minimiser, fermer les WebParts, et supprimer ceux qui sont dynamiques. La CatalogZone est en partie personnalisable. Bien évidemment, si sa structure visuelle ne vous plaît pas, vous pouvez vous-même construire votre propre « CatalogZone » en dérivant de la « CatalogZoneBase ». Voici l'héritage de la CatalogZone.

```
System.Object
System.Web.UI.Control
System.Web.UI.WebControls.WebControl
System.Web.UI.WebControls.WebParts.Zone
System.Web.UI.WebControls.WebParts.ToolZone
System.Web.UI.WebControls.WebParts.CatalogZoneBase
System.Web.UI.WebControls.WebParts.CatalogZone
```

Cette « CatalogZone » comporte une « zonetemplate » qui permet d'accueillir des CatalogParts (Catalogues de WebParts). Le framework en fournit un certain nombre :

```
System.Web.UI.WebControls.WebParts.Part
System.Web.UI.WebControls.WebParts.CatalogPart
System.Web.UI.WebControls.WebParts.DeclarativeCatalogPart
System.Web.UI.WebControls.WebParts.ImportCatalogPart
System.Web.UI.WebControls.WebParts.PageCatalogPart
```

Nous venons de voir la « PageCatalogPart ».

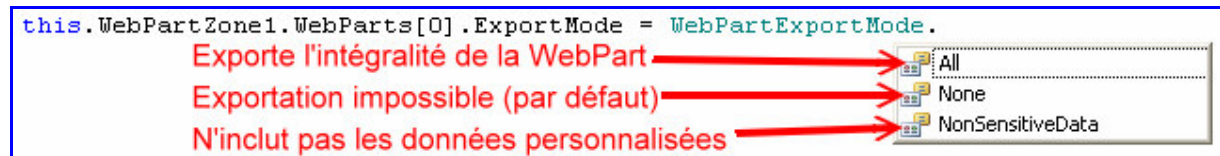
L'ImportCatalogPart

L' ImportCatalogPart permet de proposer à l'utilisateur des WebParts importées. Mais quelles WebParts ? Celles que l'utilisateur aura exportées ! Comment exporte-t-on des WebParts ?

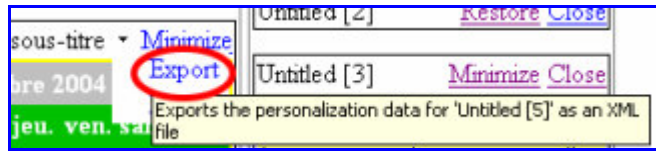
Pour rendre une WebPart exportable, il faut que son mode d'exportation l'autorise de le faire. Par défaut, ce n'est pas le cas. Nous devons modifier ce mode :

.NET passionnément, tout simplement

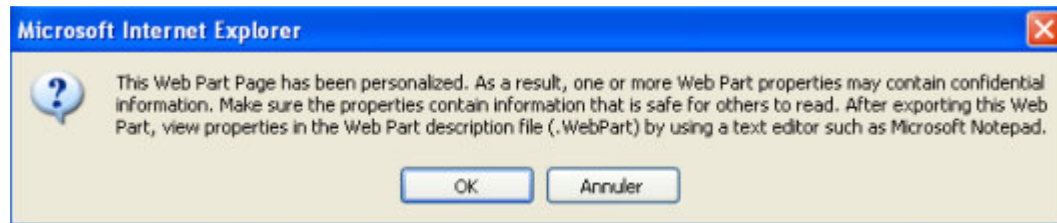
ASP.NET 2.0 : Webparts et personnalisation



Pour l'utilisateur, il devient alors possible d'exporter sa WebPart :



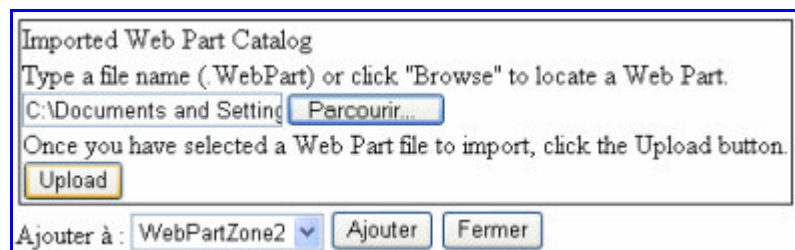
Comme nous avons sélectionné « All » pour le mode d'exportation, l'utilisateur est prévenu qu'il va stocker des données sensibles à ne pas mettre à la portée de tout le monde :



Un fichier xml est généré dont voici un extrait :



Nous allons maintenant mettre en place l'ImportCatalogPart dans la « CatalogZone » à partir de la boîte à outils de Visual Studio. Après compilation, authentification et clic sur notre bouton « Mode Catalogue », nous avons le rendu suivant :



Ce qui va nous permettre d'importer notre WebPart que nous avons sauvegardée sur notre disque dur en tant qu'utilisateur. Je vous laisse imaginer les possibilités qu'auront vos utilisateurs avec cette fonctionnalité !

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

🗑 La DeclarativeCatalogPart

La « DeclarativeCatalogZone » permet de déclarer un ensemble de WebParts à disposition de l'utilisateur. Nous pouvons y placer WebParts, WebControls, User Controls, Custom Controls.

```
<asp:DeclarativeCatalogPart Runat="server" ID="DeclarativeCatalogPart1">
  <WebPartsTemplate>
    <uc3:WebPartUserControlCalendar id="WebPartUserControlCalendar1"
      title="un user control" runat="server">
    </uc3:WebPartUserControlCalendar>
    <asp:TextBox ID="TextBox2" Runat="server" title="une TextBox"/>
  </WebPartsTemplate>
</asp:DeclarativeCatalogPart>
```

Et voici le rendu qui offre à l'utilisateur de nouvelles WebParts pour sa page :



Nous devons définir actuellement (avec la version beta1) en dur les WebParts que nous souhaitons placer dans la « DeclarativeCatalogPart ». Plus idéalement, il devrait être possible à partir de la version beta2 d'indiquer le chemin d'un fichier xml répertoriant les WebParts à y placer.

5. La zone d'édition des WebParts

🗑 L'EditorZone et l'AppearanceEditorPart

L'EditorZone permet d'éditer les WebParts de votre page. Vous pouvez y changer tout l'aspect Design mais aussi les propriétés propres à la personnalisation. Avec la boîte à outils de Visual Studio, l'EditorZone est vite opérationnelle. Glissons dans la « ZoneTemplate » une « AppearanceEditorPart. Celle-ci va nous permettre de changer le titre, de cacher les WebParts,... :

```
<asp:EditorZone ID="EditorZone1" Runat="server">
  <ZoneTemplate>
    <asp:AppearanceEditorPart Runat="server" ID="AppearanceEditorPart1"/>
  </ZoneTemplate>
</asp:EditorZone>
```

Pour pouvoir accéder, en tant qu'utilisateur, il nous faut changer de mode. A cet effet, ajoutons un bouton « Mode Edition » :

C#

```
void Button4_Click(object sender, EventArgs e)
{
    this.WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode;
}
```

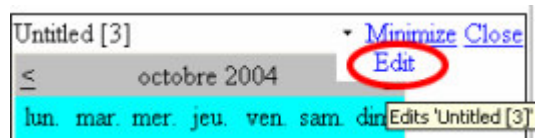
.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

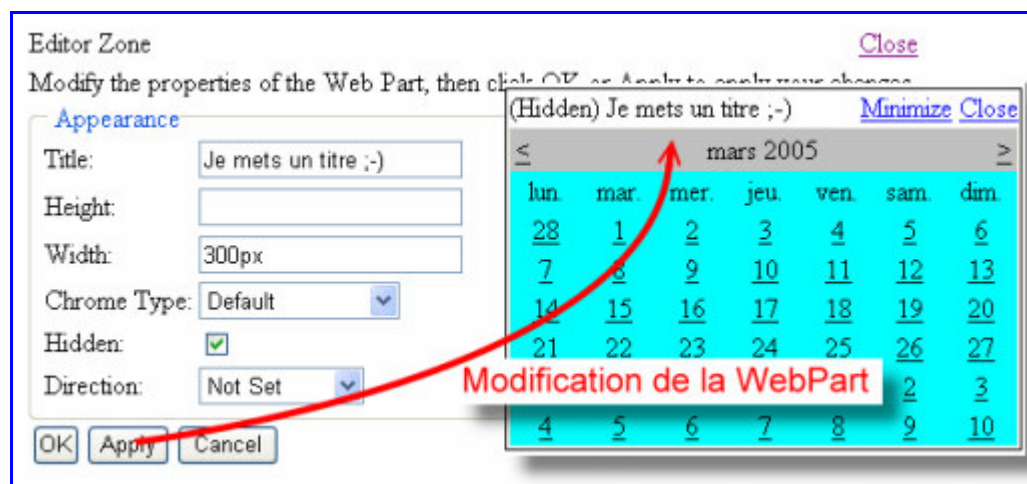
VB

```
Private Sub Button4_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode
End Sub
```

Compilons, authentifions-nous et cliquons sur le bouton « Mode Edition ». Un nouveau menu apparaît pour chaque WebPart : « Edit »



Un clic sur ce menu permet de charger l'AppearanceEditorPart et de changer un certains nombres de propriétés de la WebPart.



Quand on cache une WebPart par la propriété « hidden », comme c'est le cas ici, la WebPart n'apparaît pas quand le mode d'affichage (qui est par défaut) est en « BrowseDisplayMode ».

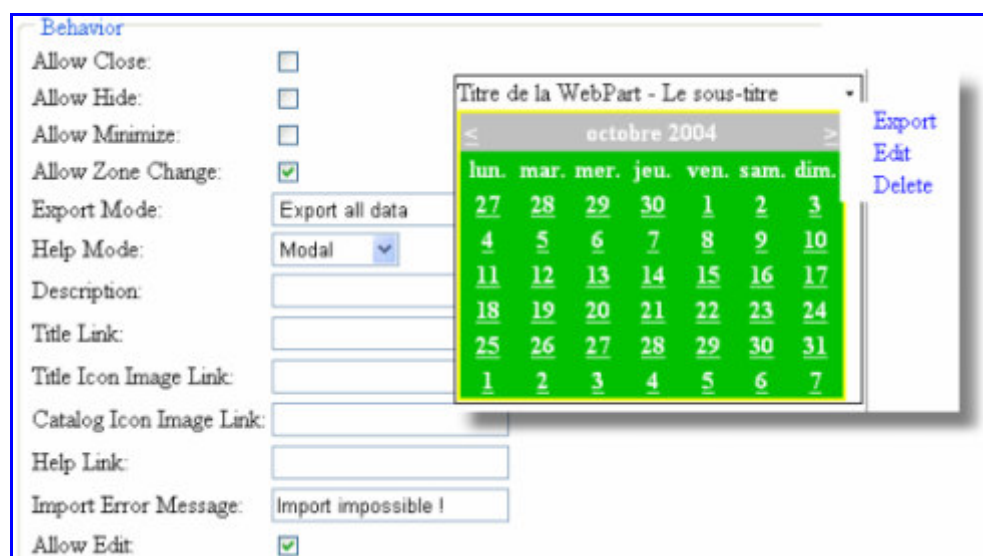
On notera qu'avec la Beta1, il n'est pas possible de changer les labels et différents messages de l'AppearanceEditorPart. Il en est de même pour les autres EditorParts. Rassurez-vous, l'apparition de propriétés avec la beta2 devrait permettre de localiser ces EditorParts comme on le souhaite.

La BehaviorEditorPart

De la boîte à outils de Visual Studio, glissons dans la « ZoneTemplate » de l'EditorZone une « BehaviorEditorPart. Compilons, authentifions-nous et cliquons sur le bouton « Mode Edition ». Le menu « Edit » nous permet d'accéder à notre nouvelle EditorPart :

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation



La LayoutEditorPart

La LayoutEditorPart permet de positionner une WebPart dans n'importe quelle WebPartZone de la page et n'importe où dans la « zonetemplate ».



Vous vous demandez sans doute quelle utilité cette EditorPart peut-elle avoir. Il s'agit d'une alternative pour les utilisateurs non munis d'Internet Explorer 6 pour le « drag and drop » des WebParts. Elle trouve donc parfaitement sa légitimité.

La PropertyGridEditorPart

La PropertyGridEditorPart permet de modifier les propriétés personnalisées des WebParts. Pour bien comprendre de quoi il s'agit, créez un User Control que vous appellerez « WebPartFournisseur.ascx ». Pourquoi ce nom car nous allons nous en servir par la suite dans cet article pour les échanges entre WebPart.

Le User Control implémentera l'interface IWebPart. Une fois l'interface implémentée, occupons nous du design :

```
<table id="Tablo" runat=server width=300>
<tr><td>Chaine</td>
<td><asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox></td>
</tr><tr>
<td>Couleur</td>
<td><asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox></td>
</tr></table>
<asp:Button ID="Button1" Runat="server" Text="Transmettre"
OnClick="Button1_Click" />
```

Un tableau Html manipulable dans le code behind, deux TextBoxs et un bouton qui serviront pour la suite de notre article.

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Définissons notre première propriété personnalisable dans notre User Control. On va définir une propriété qui va permettre de changer la couleur de notre tableau Html. De cette façon, la personnalisation de la propriété sera tout de suite visible.

C#

```
private string _CouleurTableau = "";  
[Personalizable(true), WebBrowsable(true)]  
public string CouleurTableau  
{  
    get  
    { return _CouleurTableau; }  
    set  
    {  
        _CouleurTableau = value;  
        Tableo.BgColor = _CouleurTableau;  
    }  
}
```

Attributs permettant de rendre personnalisable la propriété et la rendre visible dans le PropertyGridEditorPart

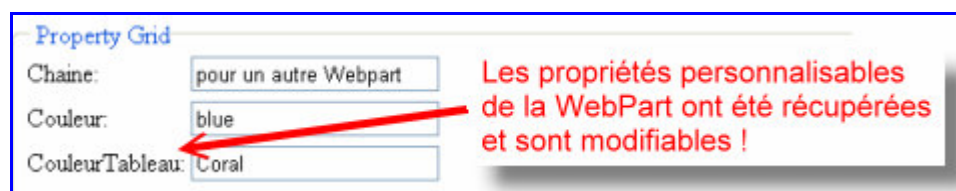
VB

```
Private _CouleurTableau As String  
<WebBrowsable(True), Personalizable(True)> _  
Public Property CouleurTableau As String  
    Get  
        Return Me._CouleurTableau  
    End Get  
    Set(ByVal value As String)  
        Me._CouleurTableau = value  
        Me.Tableo.BgColor = Me._CouleurTableau  
    End Set  
End Property
```

Plaçons notre User Control « WebPartFournisseur.ascx » dans une des WebPartZones. Ajoutons via la boîte à outils de Visual Studio la nouvelle EditorPart « PropertyGridEditorPart » dans l'EditorZone de notre page. Compilons, authentifions-nous, passons en mode « EditDisplayMode ». Voici notre nouvelle WebPart :



Ainsi que la PropertyGridEditorPart :

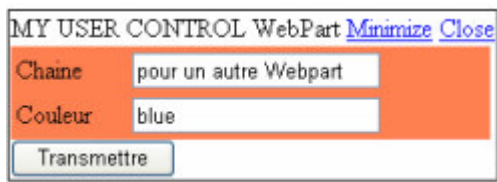


Nous notons que la PropertyGridEditorPart a su récupérer des propriétés personnalisées. Il est prévu que dans la version Beta2 du framework des WebParts que de nouveaux attributs fassent leur apparition afin de pouvoir spécifier un nom et une description de propriété plus compréhensible pour l'utilisateur.

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Une validation des changements dans l'EditorZone permet de modifier visuellement notre WebPart :



Je vous laisse tester effectivement si la personnalisation est effective. Pour cela, reconnectez-vous avec un autre compte.

6. Les échanges entre WebParts

Du fournisseur au consommateur

Nous allons maintenant traiter des échanges de données entre WebParts. Il y a une notion de fournisseur (« provider ») et de consommateur (« consumer »). Nous allons créer nos WebParts en fonction de cela.

Appuyons-nous sur le User Control « WebPartFournisseur.ascx » que nous avons créé au chapitre précédent. Pour transmettre des éléments d'une WebPart vers une autre, il est de bon usage d'utiliser une interface (comme vous le faites par exemple entre deux forms dans le monde des WinForms).

Nous allons faire dans la simplicité pour cette démonstration :

Deux propriétés : une pour passer un texte et une autre pour passer un nom de couleur.

C#

```
public interface IPartageProperties
{
    string Chaine { get;set;}
    string Couleur { get;set;}
}
```

VB

```
Public Interface IPartageProperties
    Property Chaine As String
    Property Couleur As String
End Interface
```

Implémentons l'interface dans notre User Control « WebPartFournisseur.ascx » :

C#

```
private string _Chaine = "";
[Personalizable(true), WebBrowsable(true)]
public string Chaine
{
    get { return _Chaine; }
    set
    {
        _Chaine = value;
        /*Sert quand nous opérons une modification via le
        PropertyGridEditorPart à répercuter le changement dans la TextBox ;-)*
        TextBox1.Text = _Chaine;
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
    }  
}  
  
private string _Couleur = "";  
[Personalizable(true), WebBrowsable(true)]  
public string Couleur  
{  
    get  
        { return _Couleur; }  
    set  
        {  
            _Couleur = value;  
            TextBox2.Text = _Couleur;  
        }  
}
```

VB

```
Private _Chaine As String  
<Personalizable(True), WebBrowsable(True)> _  
Public Property Chaine As String  
    Get  
        Return Me._Chaine  
    End Get  
    Set(ByVal value As String)  
        Me._Chaine = value  
        Me.TextBox1.Text = Me._Chaine  
    End Set  
End Property  
  
Private _Couleur As String  
<Personalizable(True), WebBrowsable(True)> _  
Public Property Couleur As String  
    Get  
        Return Me._Couleur  
    End Get  
    Set(ByVal value As String)  
        Me._Couleur = value  
        Me.TextBox2.Text = Me._Couleur  
    End Set  
End Property
```

Traisons l'action sur le bouton « Transmettre » de notre User Control. Il récupère les valeurs des deux TextBox et affectent les propriétés personnalisables par ces valeurs :

C#

```
void Button1_Click(object sender, EventArgs e)  
{  
    _Chaine = TextBox1.Text;  
    _Couleur = TextBox2.Text;  
}
```

VB

```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)  
    _Chaine = TextBox1.Text  
    _Couleur = TextBox2.Text  
End Sub
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Pour être consommées, nos données doivent être exposées. Elles vont l'être via l'interface que nous avons définie « IPartageProperties ». Mais il va falloir établir un pont (un point de connexion) avec une autre WebPart. Cela se fait grâce à un attribut « ConnectionProviderAttribute » :

C#

```
[ConnectionProvider("FournisseurConsommateur")]
public IPartageProperties PartageProperties()
{
    return this;
}
```

VB

```
<ConnectionProvider("FournisseurConsommateur")> _
Public Function PartageProperties() As IPartageProperties
    Return Me
End Function
```

Il nous faut maintenant construire notre WebPart consommatrice. Créez un User Control appelé « WebPartConsommateur.ascx » implémentant l'interface IWebPart. Nous n'allons y placer qu'un Label :

```
<asp:Label ID="Label1" Runat="server" Text="Label">Pas de connexion entre
les webparts</asp:Label>
```

Et nous faisons en sorte que notre WebPart ne puisse pas être fermée, ni minimisée et que la WebPartZone qui la contient soit de couleur grise foncée. Pourquoi ? Pour se dire que cela est possible ;-)

C#

```
protected override void OnPreRender(EventArgs e)
{
    /*La Modification du comportement d'un WebPart doit se faire juste avant
    le rendu*/
    //on recherche le WebPartManager courant.
    WebPartManager wpm = WebPartManager.GetCurrentWebPartManager(this.Page);
    //on récupère le GenericWebPart prenant en charge notre UserControl
    GenericWebPart genPart = wpm.GetGenericWebPart(this);
    /*on s'assure que notre GenericWebPart, donc notre UserControl a été
    instancié dans la page*/
    if (genPart != null)
    {
        //Modification du comportement de la GenericWebPart
        genPart.AllowClose = false;
        genPart.AllowMinimize = false;
        //Changement de la couleur de la WebPartZone
        genPart.Zone.BackColor = System.Drawing.Color.DarkGrey;
    }
    base.OnPreRender(e);
}
```

VB

```
Protected Overrides Sub OnPreRender(ByVal e As EventArgs)
    Dim manager1 As WebPartManager = WebPartManager.GetCurrentWebPartManager(Me.Page)
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
Dim genPart As GenericWebPart = manager1.GetGenericWebPart (Me)
If (Not genPart Is Nothing) Then
    part1.AllowClose = False
    part1.AllowMinimize = False
    part1.Zone.BackColor = Color.DarkGray
End If
MyBase.OnPreRender (e)
End Sub
```

Après ce divertissement, il nous faut récupérer les données de notre WebPart « fournisseur ». Nous avons juste à indiquer quelle méthode va consommer. Cela se fait via l'attribut « ConnectionConsumerAttribute ». Nous allons récupérer les données via l'interface « IPartageProperties ». Il devient possible de changer le texte de notre Label et la couleur de fond de celui-ci :

C#

```
[ConnectionConsumer("FournisseurConsommateur")]
public void Consommation(IPartageProperties c)
{
    Label1.Text = c.Chaine;
    Label1.BackColor = System.Drawing.Color.FromName(c.Couleur);
}
```

VB

```
<ConnectionConsumer("FournisseurConsommateur")> _
Public Sub Consommation(ByVal c As IPartageProperties)
    Label1.Text = c.Chaine
    Label1.BackColor = Color.FromName(c.Couleur)
End Sub
```

Connexion statique entre deux WebParts

Pour l'instant, nous n'avons défini que la zone d'émission et la zone de réception. Qu'est ce qui va faire la liaison ? Nous avons tout fait pour que les deux WebParts puissent se comprendre. Nous devons maintenant établir une connexion.

Les connexions entre WebParts sont dites statiques si elles sont déjà définies dans la page en cours. Elles sont dites dynamiques si une action dans la page provoque « l'apparition » d'une liaison (connexion) entre deux WebParts.

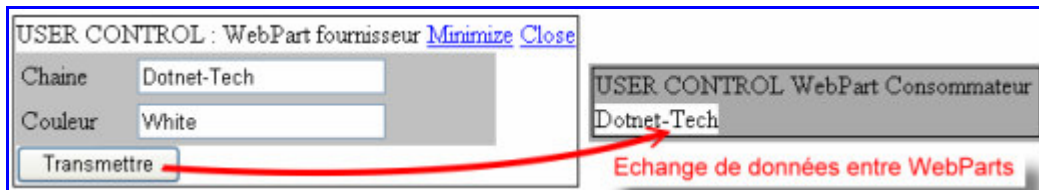
Il est très facile de définir une connexion statique. Il suffit de la déclarer dans le WebPartManager en spécifiant l'ID des WebParts « provider » et « consumer » :

```
<asp:WebPartManager ID="WebPartManager1" Runat="server">
    <StaticConnections>
        <asp:Connection ID="ConnexionStatic"
            ConsumerID="WebPartConsommateur1" ProviderID="WebPartFournisseur1">
        </asp:Connection>
    </StaticConnections>
</asp:WebPartManager>
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Compilons. Authentifions-nous. Il nous suffit maintenant de rentrer une chaîne de caractères, un nom de couleur et d'appuyer sur le bouton « transmettre » de notre WebPart « fournisseur », la WebPart « consommateur » est alors mise à jour.



C'est aussi un très bon moyen de mettre à jour vos WebParts : On peut par exemple cacher la WebPart « fournisseur » qui ne sera plus accessible qu'en mode « EditDisplayMode ». Ce qui peut être intéressant pour la maintenance d'un site. Par exemple, on peut imaginer d'uploader un fichier de contenu Html via le WebPart « fournisseur » et le transmettre au WebPart « consommateur ». De nombreuses perspectives fonctionnelles vous sont donc ouvertes assez facilement.

Connexion dynamique entre deux WebParts

Traitions maintenant des connexions dynamiques. Imaginons que vous importiez des WebParts « fournisseur » et « consommateur », les connexions statiques ne sont donc plus envisageables. Il existe un mode pour permettre la connexion ou la déconnexion entre deux WebParts. Il s'agit du mode « ConnectDisplayMode ». Ajoutons donc un nouveau bouton « Mode Connexion » à notre page afin de pouvoir passer dans ce mode. En cliquant, notre page bascule en « ConnectDisplayMode » :

C#

```
void Button5_Click(object sender, EventArgs e)
{
    this.WebPartManager1.DisplayMode = WebPartManager.ConnectDisplayMode;
}
```

VB

```
Private Sub Button5_Click(ByVal sender As Object, ByVal e As EventArgs)
    WebPartManager1.DisplayMode = WebPartManager.ConnectDisplayMode
End Sub
```

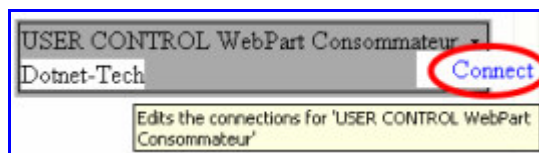
Servons-nous de la boîte à outils de Visual Studio pour insérer un nouveau composant, la « ConnectionsZone », dans notre page. La ConnectionsZone va nous permettre d'établir des connexions entre WebParts partageant les mêmes interfaces de propriétés personnalisées. Elle permet aussi de déconnecter deux WebParts.

Pour la démonstration, je vous invite à supprimer la connexion statique que nous avons mise en place précédemment afin que nous n'ayons pas de problèmes. Logiquement, deux WebParts n'acceptent qu'une connexion de même sens entre elles.

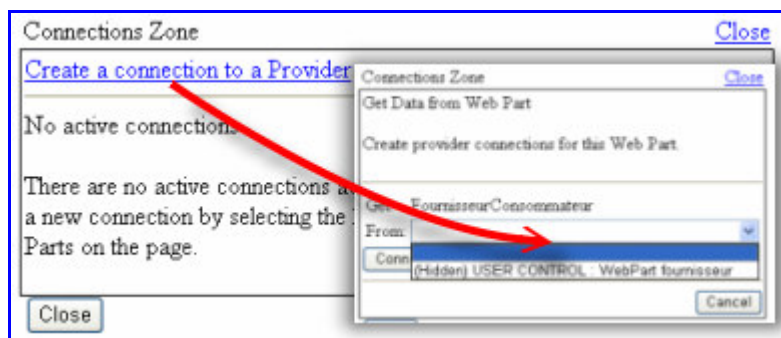
Compilons, authentifions-nous et cliquons sur le bouton « Mode Connexion ». Nous pouvons accéder pour toutes les WebParts disposant d'un « ConnectionProviderAttribute » ou d'un « ConnectionConsumerAttribute » renseigné, à un menu pour se connecter :

.NET passionnément, tout simplement

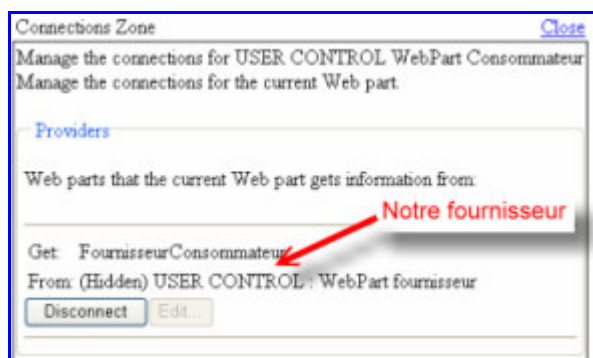
ASP.NET 2.0 : Webparts et personnalisation



Ce qui nous permet d'accéder à la ConnectionsZone. Celle-ci donne la possibilité de choisir une WebPart « fournisseur » et établir une connexion entre elles :



Nous obtenons aussi la liste des « fournisseurs » de notre WebPart :



Nous pouvons à toute moment cesser notre connexion entre les deux WebParts.

7. Les filtres d'autorisation

Pour l'instant, nous n'avons parlé aucunement des rôles de nos utilisateurs.

Suivant le rôle de nos utilisateurs, on donnera accès à certaines WebParts et pas à d'autres.

Mettons en place un filtre d'autorisation. Pour cela, il existe une propriété qui permet de spécifier les rôles autorisés pour la WebPart : « AuthorizationFilter ». Si nous reprenons un de nos précédents User Control, il nous suffit pour mettre en place la propriété d'agir sur le pré-rendu du control.

C#

```
protected override void OnPreRender (EventArgs e)
{
    WebPartManager wpm = WebPartManager.GetCurrentWebPartManager (this.Page);
    GenericWebPart genPart = wpm.GetGenericWebPart (this);
    if (genPart != null)
    {
        genPart.AuthorizationFilter = "administrateurs";
    }
}
```

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
}  
    base.OnPreRender(e);  
}
```

VB

```
Protected Overrides Sub OnPreRender(ByVal e As EventArgs)  
    Dim manager1 As WebPartManager;  
    Manager1 = WebPartManager.GetCurrentWebPartManager(Me.Page)  
    Dim part1 As GenericWebPart = manager1.GetGenericWebPart(Me)  
    If (Not part1 Is Nothing) Then  
        part1.AuthorizationFilter = "administrateurs"  
    End If  
    MyBase.OnPreRender(e)  
End Sub
```

Comme nous le constatons, il suffit de lister les rôles que nous aurons définis au préalable.

Compilons, authentifions nous comme « administrateurs ». Nous disposons bien de la WebPart. Déconnectons-nous pour nous identifier avec un utilisateur ne disposant pas de ce rôle. Vous allez avoir la surprise, si vous utilisez la version Beta1, de pouvoir avoir accès à la WebPart, plus particulièrement à la GenericWebPart. L'implémentation de la propriété n'a pas encore été achevée sous la version beta1. En fait, tout ce qui est relatif aux GenericWebParts n'a pas de propriété « AuthorizationFilter » fonctionnant.

Ceci sera corrigé dans la version beta2 de ASP.NET 2.0.

Mais vous pouvez déjà voir qu'il est facile de restreindre l'accès au WebPart à certains rôles.

8. Brève initiation aux thèmes

De quoi s'agit-il ?

Nous avons pu constater jusqu'à maintenant que l'aspect de nos WebParts était plutôt laid car nous ne sommes pas soucieux de la présentation. Nous allons voir qu'il est très facile de « designer » un site rapidement.

Concernant, l'apparence d'un site, il existe une nouvelle notion sous ASP.NET 2.0 : les thèmes. De quoi s'agit-il ? Il s'agit de modèles d'apparence très faciles à mettre en place.

Mode d'emploi

Créons à la racine de notre site Web, un répertoire que nous nommerons « Themes » (A noter, que dès la version beta2, ce répertoire devra se nommer « Application_Themes »). Intégrons un répertoire que nous nommerons « blues ». A l'intérieur de ce dernier, codons un fichier CSS que nous nommerons « Default.css ». Définissez-y des éléments tel que le « body », les liens, etc... Nous allons créer un fichier « Default.skin » qui va contenir les tags des contrôles les plus courants que nous utilisons dans nos pages. Ces tags ne contiennent pas d'ID mais de nombreuses propriétés de design. Essayons d'intégrer le tag WebPartZone :

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

```
<asp:WebPartZone skinid="skinDotnetTech" runat="server" CloseVerb-ImageUrl="images/bouton_
  <MenuStyle BackImageUrl="images/bouton_fermer.gif"/>
</asp:WebPartZone>

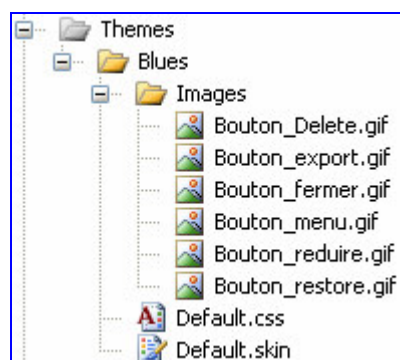
<asp:WebPartZone DeleteVerb-ImageUrl="images/Bouton_delete.gif"
  MenuVerbStyle-ForeColor="#ffffff" MenuStyle-BackColor="#3366ff"
  MenuStyle-BorderColor="#000000" MenuStyle-BorderWidth="1"
  MenuPopupImageUrl="images/bouton_menu.gif"
  RestoreVerb-ImageUrl="images/bouton_restore.gif"
  PartTitleStyle-ForeColor="#ffffff" PartTitleStyle-BackColor="#3333ff"
  runat="server" CloseVerb-ImageUrl="images/bouton_fermer.gif"
  ExportVerb-ImageUrl="images/bouton_export.gif"
  MinimizeVerb-ImageUrl="images/bouton_reduire.gif" />
```

Permet d'avoir plusieurs skins pour un même tag

Images et propriétés propres au thème

Un extrait du fichier « Default.skin ».

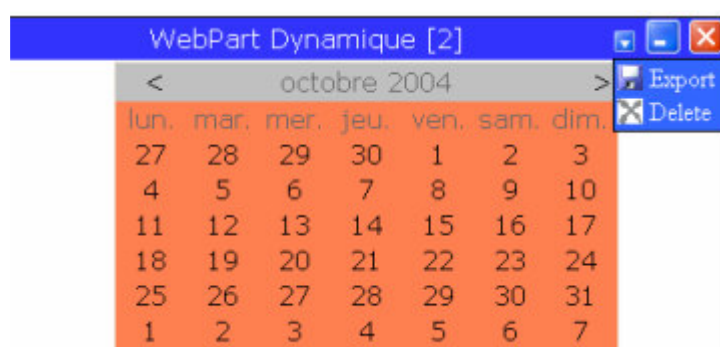
Les images se trouvent dans un répertoire à la racine de notre thème « Blues » :



Appliquez un thème sur une page est un jeu d'enfant :



Compilons, authentifions-nous, voici l'aspect général de nos WebParts :



Il ne s'agit pas d'une fenêtre Windows mais bien d'une WebPart sur lequel s'applique un thème de page !

Il est bien sûr tout à fait possible de personnaliser les thèmes de vos pages. Mais ceci est déjà un autre article... Voir les liens de « En savoir plus ».

.NET passionnément, tout simplement

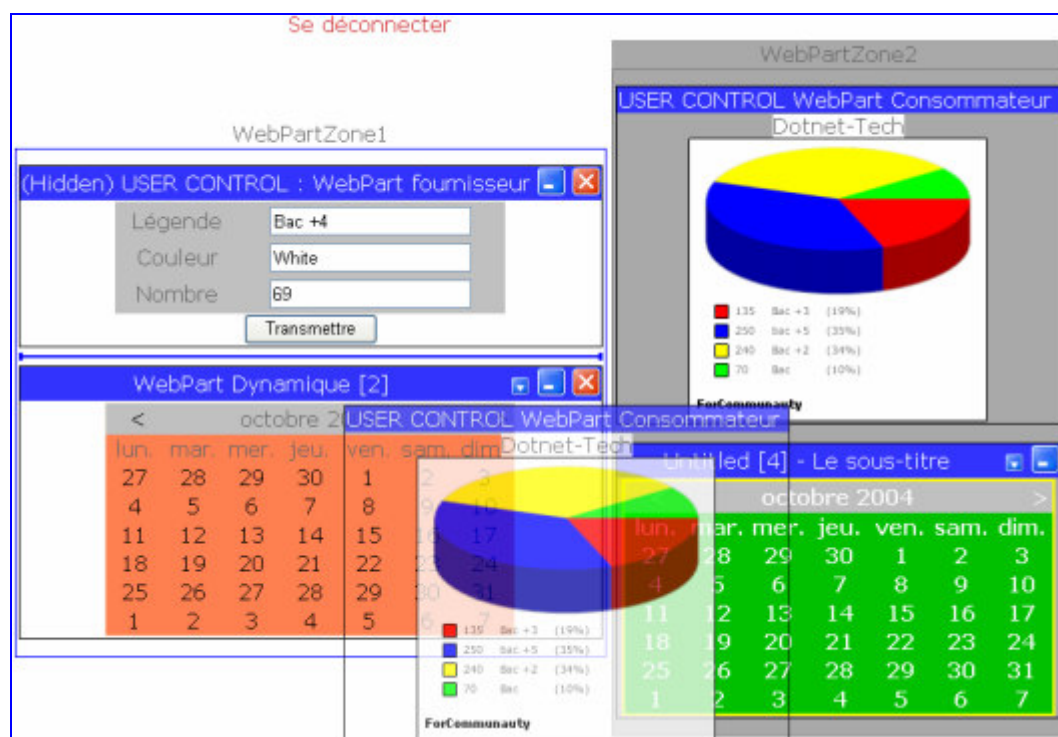
ASP.NET 2.0 : Webparts et personnalisation

9. Conclusion

Le framework des WebParts nous offre de belles perspectives fonctionnelles. Mettre en place un portail évolutif et modulable devient plus facile et plus rapide. Dans cette version beta 1, nous avons déjà l'essentiel.

Nous n'avons pas parlé du composant « WebPartPageMenu » car celui-ci sera abandonné à partir de la version beta 2 pour des questions de délais de développement. Mais, nous avons pu voir que nous pouvions nous en passer et que même son absence, nous permettra de proposer un fonctionnel moins « typé ».

Voici un exemple de page que nous pouvons facilement construire :



10. En savoir plus

Il n'a pas été facile de réaliser cet article car la documentation est pauvre pour l'instant et cela se comprend parfaitement, ASP.NET 2.0 n'en est qu'à la version beta 1. Aussi, le forum du site « asp.net » m'a été d'un très grand secours. Aussi, je vous le recommande vivement. Pour ce qui concerne les WebParts :

<http://www.asp.net/Forums/ShowForum.aspx?tabindex=1&ForumID=145>

Citons Fredrik Normén et Carlos Ag qui font un travail remarquable dans ce forum pour nous éclairer.

Le Blog de Fredrik Normén

<http://fredrik.nsqared2.com/>

.NET passionnément, tout simplement

ASP.NET 2.0 : Webparts et personnalisation

Le site de Carlos Ag de l'ASP.NET Team

<http://www.carlosag.net>

D'autres liens :

Creating Web Application Themes in ASP.NET 2.0

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

Une série de Webcasts sur ASP.NET 2.0 à suivre :

<http://msdn.microsoft.com/training/webcasts/#ASP>

Article nouvellement publié qui vous donnera un matériel supplémentaire pour votre apprentissage : Introducing the ASP.NET 2.0 Web Parts Framework

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

11. Notes

Les 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>

Les images et logos des captures d'écrans utilisés l'ont été avec l'autorisation de Microsoft France. Merci de vous référer à ce site

<http://www.microsoft.com/france/core/copyright.msp> pour une utilisation ultérieure.

N'hésitez pas à me contacter :

Frédéric Mélantois

Email : fmelantois@free.fr