Affichage et thèmes

Sommaire

1 Introduction

Pour que l'utilisateur puisse facilement utiliser votre site, il faudra que celui-ci soit agréable et que la navigation soit facile : pensez qu'un utilisateur qui ne trouve pas facilement l'information ou le service qu'il recherche facilement sur votre site ne passerez pas forcément plus de temps dessus et préférera chercher sur un moteur de recherche un autre site qui le lui fournira. Pour cela une chose importante est la présentation de votre site. Le thème, c'es à dire l'agencement des différents blocs ainsi que leurs couleurs et forment le rendront plus agréable et surtout permettra un meilleur agencement, donc une navigation simplifié. ASP.NET nous aide à créer un thème qui soit facilement applicable et utilisable.

Pour cela nous aurons à notre disposition les Master Pages, qui vont nous permettre de ne pas avoir à refaire sur chaque page la même disposition des blocs (nous pouvons aussi en faire plusieurs pour laisser le choix à l'utilisateur). Les thèmes qui vont nous permettre de proposer à l'utilisateur différentes possibilités au niveau du css (couleur, taille …) suivant les thèmes que vous aurez fait. Enfin nous verrons les Web Parts qui permettent de laisser le choix à l'utilisateur de ce qu'il veux afficher ou non ou encore d'afficher une information différente à un endroit de la page suivant ce que l'utilisateur choisira comme action.

2 Les Master Pages

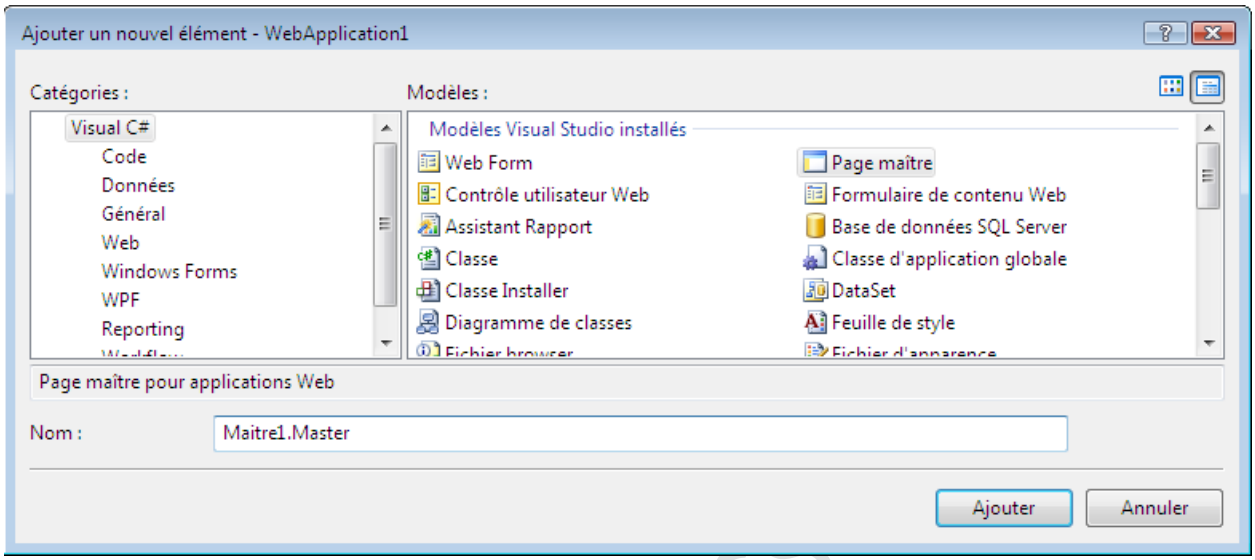
Prenons l'exemple d'un site composé de trois grandes parties : une bannière en haut, un pied de page comportant notamment le copyright et le corps de la page. Seul le corps change quel que soit la partie du site dans laquelle nous nous trouvons. Au lieu de refaire à chaque page tout notre affichage avec un risque d'oublier ou de changer quelque chose au passage, on va tout simplement créer une Master Page. Voyons ce que c'est et en quoi cela va nous simplifier grandement la tâche.

2.1 Introduction

Une Master Page est en fait une page Web qui va servir de modèle pour les autres pages. Plus exactement, on va définir sur une page quel Master Page (page maître en français) elle va devoir utiliser. Pour que cela puisse fonctionner, la Master Page contient des ContentPlaceHolder avec des ID différents. Ce sont ces conteneurs que le page va remplir. En résumé : une page va pointer vers un modèle qu'elle va utiliser. Dans ce modèle se trouvent un ou plusieurs conteneurs. La page qui a été appelé par l'utilisateur ne fera que remplir le ou les conteneurs avec le code que nous avons spécifié (du moins si nous avons défini un contenu pour le conteneur). Voila pour la partie théorique : passons maintenant à la création et à l'utilisation d'une Master Page.

2.2 Création et utilisation d'une page maître

Pour créer une page maître, faites comme pour rajouter n'importe quel page : clic droit sur le nom du projet, Ajouter, Nouvel Elément. Dans la fenêtre qui s'ouvre (voir image ci-dessous) choisissez Page maître.



Vous verrez alors s'ouvrir une page avec ce code ci :

Code par défaut de la page maitre Maitre1.Master

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Maitre1.master.cs"
Inherits="WebApplication1.Maitre1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Page sans titre</title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

Comme vous pouvez le constater, cette page correspond à une page ASPX normale à la différence que la directive en haut ne s'appelle pas Page mais Master. Autre chose : Il contient, par défaut, deux *ContentPlaceHolder*. Un dans le head et un dans le body pour être plus exact. Ce sont des conteneurs qui vont nous permettre de placer le code spécifique à chaque page. On peut donc rajouter depuis n'importe quel page utilisant cette page maître : des en-têtes (dans le head) et du contenu dans le body. Il est bien entendu possible de rajouter ses propres ContentPlaceHolder.

La prochaine étape va être de créer notre propre page maître et de l'utiliser dans la page Default.aspx.

Pour cela voyons d'abord le code de la page maître une fois modifié (les modifications sont soulignées) :

Code de la page maître

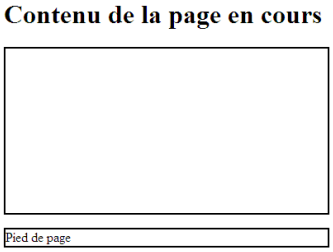
```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Maitre1.master.cs"
Inherits="WebApplication1.Maitre1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">

    <title>Page maître</title>

    <asp:ContentPlaceHolder ID="head" runat="server" />
</head>
<body>
    <form id="form1" runat="server">

    <h1>Contenu de la page en cours</h1>
    <div style="border:solid 2px black;height: 200px;">
        <asp:ContentPlaceHolder ID="Body" runat="server" />
    </div>
    <p style="border:solid 2px black;">Pied de page</p>

    </form>
</body>
</html>
```

On a rajouté du HTML et du CSS tout simple. Maintenant voyons comment utiliser notre page maître comme modèle pour notre page Default.aspx. Sur le plan technique, la page maître possède déjà le Doctype ainsi que les balises html, head et body. Notre page Default.aspx n'en a donc pas besoin : il ne lui reste que la directive page. C'est dans celle-ci que nous allons dire que cette page doit utiliser notre page maître. Pour cela nous utilisons la propriété MasterPageFile. Il y a une autre propriété qui est Title et qui nous permet de modifier le titre de la page même s'il a été défini dans la page maître (comme c'est le cas ici). La directive page doit maintenant ressembler à ceci :

Directive de notre page Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="WebApplication1._Default"
         MasterPageFile="~/Maitre1.Master"
         Title="Test"  %>
```

A partir de maintenant notre page utilise la page maître. On peut d'ailleurs l'afficher. Voici le résultat :



Cet exemple nous montre bien que l'on n'est pas obligé de mettre du contenu dans la page maître. Si vous regardez la barre d'adresse vous verrez que la propriété Title de la page Default.aspx a bien modifié le titre de la page. Maintenant que nous savons utiliser les pages maîtres dans nos pages, nous allons apprendre à y ajouter du contenu.

2.3 Ajouter du contenu

Maintenant que c'est un peu plus clair dans vos esprits nous allons voir comment ajouter du contenu. Pour cela nous avons une page Default.aspx qui va afficher dans le corps de la page un formulaire qui nous permettra de rentrer notre pseudo. Et une seconde page qui va récupérer le pseudo et l'afficher. Pour cela on va utiliser la page maître faites plus haut.

Pour ajouter du contenu au ContentPlaceHolder de la page maître, il va falloir rajouter une balise spéciale : asp:Content.

Page Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="WebApplication1._Default"
         MasterPageFile="~/Maitre1.Master"
         Title="Test"  %>

<asp:Content ID="Content1" ContentPlaceHolderID="Body" Runat="Server">

</asp:Content>
```

Il y a un ID par défaut sur le Content. Celui-ci n'est pas obligatoire. En revanche on est obligé de spécifier quel ContentPlaceHolder on va remplir avec la propriété ContentPlaceHolderId, qui, comme vous le comprenez va prendre comme paramètre l'ID du ContentPlaceHolder. Ensuite le runat="server" car c'est une balise ASP.NET. C'est à l'intérieur de cette balise Content que l'on va placer notre code. Bien entendu on peut mettre plusieurs balise Content pour remplir les différents ContentPlaceHolder qui se trouvent dans la page maître. Remplissons-le avec notre formulaire.

Page Default.aspx

```
<asp:Content ID="Content1" ContentPlaceHolderID="Body" Runat="Server">
    <asp:Label ID="Pseudo" runat="server" Text="Pseudo" />
    <asp:TextBox ID="PseudoBox" runat="server" /><br />

    <asp:Button ID="Valider" runat="server" Text="Envoyer" OnClick="Envoi_Pseudo" />
</asp:Content>
```

On va passer le pseudo par la session de l'utilisateur :

C# - Page Default.aspx.cs

```
protected void Envoi_Pseudo(object sender, EventArgs e)
{
    Session["Pseudo"] = PseudoBox.Text;
    Response.Redirect("GestionFormulaire.aspx");
}
```

VB.NET – Page Default.asp.vb

```
Protected Sub Envoi_Pseudo(ByVal sende As Object, ByVal e As EventArgs)
    Session("Pseudo") = PseudoBox.Text
    Response.Redirect("GestionFormulaire.aspx")
End Sub
```

Ensuite faisons la page qui va récupérer le pseudo :

Page GestionFormulaire.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="GestionFormulaire.aspx.cs"
         Inherits="WebApplication1.GestionFormulaire"
         MasterPageFile="~/Maitre1.Master"
         Title="Résultat formulaire"  %>

<asp:Content ID="Content1" ContentPlaceHolderID="Body" Runat="Server">
    <asp:Label ID="PseudoLabel" runat="server" Text="Aucun pseudo" />
</asp:Content>
```

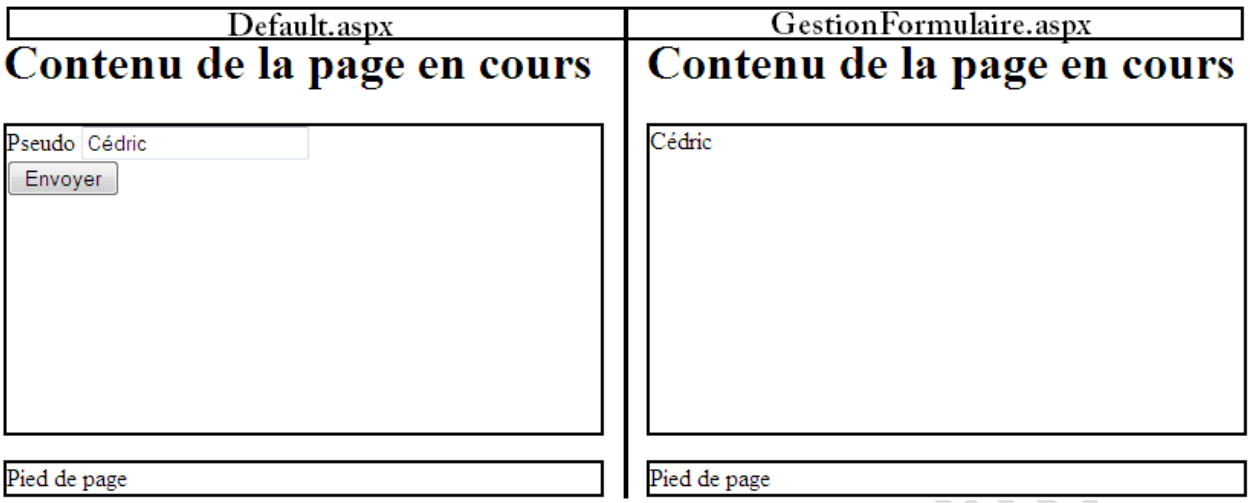
C# - Page GestionFormulaire.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["Pseudo"] != null)
        PseudoLabel.Text = Session["Pseudo"].ToString();
}
```

VB.NET – Page GestionFormulaire.aspx.vb

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    If Session("Pseudo") <> Nothing Then
        PseudoLabel.Text = Session("Pseudo").ToString()
    End If
End Sub
```

Voici un exemple fait avec les deux pages ci-dessus :



Remarque : Si vous définissez du contenu dans un des ContentPlaceHolder, ce contenu sera pris comme contenu par défaut. C'est-à-dire que si une page utilisant cette page maître ne possède pas de contenu pour ce ContentPlaceHolder, le contenu par défaut sera affiché.

2.4 Lier toutes les pages à une page maître

Nous avons vu comment utiliser une page maître sur une page. Il est aussi possible de définir au niveau de l'application que toutes les pages utiliseront une page maître. On va donc utiliser le Web.config qui est, nous le rappelons, le fichier de configuration de notre application Web, pour définir ceci.

On va créer une balise pages dans le system.web. Voici à quoi cela doit ressembler :

Web.config

```
<system.web>
    <pages masterPageFile="~/Maitre1.Master" />
</system.web>
```

Faites le test : enlevez les propriétés MasterPageFile des deux pages ASPX après avoir rajouté la balise page dans le Web.config.

2.5 Modification par code behind

Tout comme une page ASPX on peut utiliser du code behind. Nous allons plus e ndétail comment gérer cela par rapport aux pages ASPX.

2.5.1 Utiliser les propriétés de la page maître dans les autres pages

Pour comprendre cet exemple nous allons faire quelque chose de simple. La page maître contiendra une TextBox et la page Default.aspx va contenir un bouton et un Label. Quand on clic sur le bouton on prend la valeur de la TextBox et la place dans le Label.

AccesMaitre.Master

```
<div>
    <asp:TextBox ID="TextBox" runat="server"></asp:TextBox><br /><br />
    <asp:ContentPlaceHolder ID="Body" runat="server">
    </asp:ContentPlaceHolder>
</div>
```

Avec l'accesseur qui permet de récupérer le texte entré dans la TextBox :

AccesMaitre.Master.cs

```
public string Recuperation
{
    get { return TextBox.Text; }
}
```

AccesMaitre.Master.vb

```
Public ReadOnly Property Recuperation() As String
    Get
        Return TextBox.Text
    End Get
End Property
```

Notre page maître est prête. Il ne reste plus qu'a faire notre page Default.aspx. Pour pouvoir récupérer la propriété, et donc accéder au texte, il va falloir spécifier dans la page Default.aspx le chemin vers la propriété (ou les s'il y en a plusieurs). Pour ce faire nous devons spécifier un chemin virtuel vers la page maître avec la directive MasterType et sa propriété VirtualPath. Une fois ceci fait on peut récupérer les méthodes de la page maître grâce à l'objet Master. Pour comprendre un peu mieux voici le code de la page Default.aspx. Regardez le attentivement pour comprendre son fonctionnement.

Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="WebApplication1._Default"
         MasterPageFile="~/AccesMaitre.Master"
         Title="Test"  %>

<%@ MasterType VirtualPath="~/AccesMaitre.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="Body" Runat="Server">
    <asp:Button runat="server" Text="Charger le Label" OnClick="ChargeLabel" />
    <asp:Label ID="Label1" runat="server" Text="Vide !"></asp:Label>
</asp:Content>
```

Default.aspx.cs

```
protected void ChargeLabel(object sender, EventArgs e)
{
    Label1.Text = Master.Recuperation;
}
```

Default.aspx.vb

```
Protected Sub ChargeLabel(ByVal sender As Object, ByVal e As EventArgs)
    Label1.Text = Master.Recuperation
End Sub
```

Avec cette méthode on peut accéder à n'importe qu'elle propriété de la page maître. Remarquez qu'il faut quand même que la propriété ou méthode en question soit publique.

2.5.2 Accéder aux contrôles de la page maître

Tout comme on peut accéder aux méthodes et propriétés de la page maître depuis le code behind d'une autre page, on peut aussi accéder à ses contrôles.

Reprenons l'autre exemple, au lieu d'appeler une méthode de la page maître on va tout simplement accéder au contrôle TextBox et récupérer son texte. Pour faire cela nous n'avons pas besoin de chemin virtuel comme nous avions eu besoin pour accéder aux méthodes. Voici ce que va contenir notre évènement clic du bouton de Default.aspx.

Default.aspx.cs

```
protected void ChargeLabel(object sender, EventArgs e)
{
    Label1.Text = ((TextBox)Master.FindControl("TextBox")).Text;
}
```

Default.aspx.vb

```
Protected Sub ChargeLabel(ByVal sender As Object, ByVal e As EventArgs)
    Label1.Text = CType(Master.FindControl("TextBox"), TextBox).Text
End Sub
```

Avec cette façon d'écrire et dans ce cas on obtient le même résultat qu'avec un accesseur. Mais c'est utile dans d'autre cas.

2.6 Imbrication des pages maître

Tout comme un conteneur peut contenir un autre conteneur, une page maitre peut en contenir une autre. L'utilité est que lorsque que nous de allons avoir une partie du site légèrement différente du reste (par exemple il faut un menu à gauche), et bien il suffira d'avoir une page maître qui se base sur la première et ne fait que rajouter le menu. Ceci est un exemple parmi tant d'autres possibles bien sur.

Pour ce qui est de la technique, cela n'est pas plus compliqué que d'utiliser une page maitre avec un page aspx. C'est-à-dire que le seconde page maître aura une balise Content, par exemple sur le ContentPlaceHolder nommé Body plus haut. Et que dans ce Content nous aurons un ContentPlaceHolder qui se situera à droite de la page et un menu (par exemple une liste dans un div). Ainsi la partie du site qui a besoin d'un menu va utiliser la seconde page maître (que nous nommerons AffichMenu.Master).

Voici ce que cela donne :

Principale.Master

```
<%@ Master Language="C#"
           AutoEventWireup="true"
           CodeBehind="Principale.master.cs"
           Inherits="WebApplication1.Principale" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Page sans titre</title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <h1>Page Principale</h1>
        <div style="border: solid black 2px; padding: 10px;">ContentPlaceHolder :
Body
            <asp:ContentPlaceHolder ID="Body" runat="server">

            </asp:ContentPlaceHolder>
        </div>
    </div>
    </form>
</body>
</html>
```

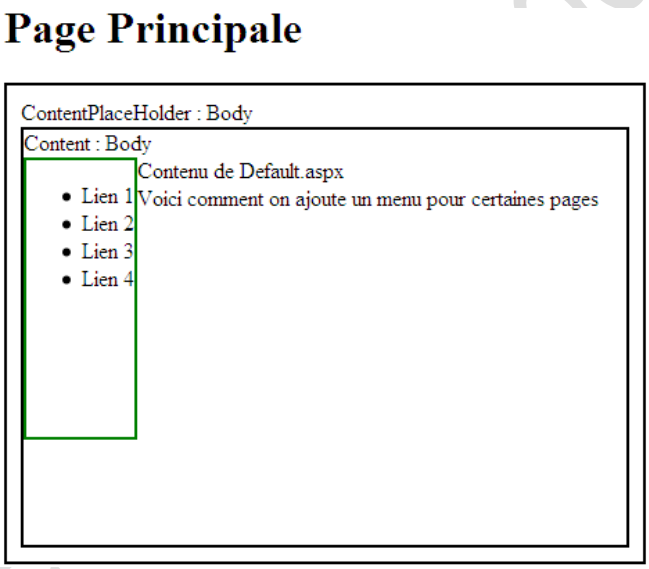
AffichMenu.Master

```
<%@ Master Language="C#"
           AutoEventWireup="true"
           MasterPageFile="~/Principale.Master"
           CodeBehind="AffichMenu.master.cs"
           Inherits="WebApplication1.AffichMenu" %>

<asp:Content runat="server" ContentPlaceHolderID="Body">
    <div style="height: 300px; border: solid black 2px;">Content : Body
        <div style="height: 200px; border: solid 2px green; float: left;">
            <ul>
                <li>Lien 1</li>
                <li>Lien 2</li>
                <li>Lien 3</li>
                <li>Lien 4</li>
            </ul>
        </div>
        <div>
            <asp:ContentPlaceHolder ID="Body2" runat="server" />
        </div>
    </div>
</asp:Content>
```

Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="WebApplication1._Default"
         MasterPageFile="~/AffichMenu.Master"
         Title="Test"  %>

<asp:Content ID="Content1" ContentPlaceHolderID="Body2" Runat="Server">
Contenu de Default.aspx<br />
Voici comment on ajoute un menu pour certaines pages
</asp:Content>
```

Regardez bien le code de la page AffichMenu.Master. C'est lui le plus important. Vous voyez qu'il ressemble beaucoup à une page aspx qui utiliserait une page maître avec juste la directive qui reste Master. Elle comporte aussi un Content pour remplir le Body. Dedans on défini l'architecture de notre page et on remet un ContentPlaceHolder à l'endroit où l'on veut que le contenu de la page se place. Le reste (l'affichage, les bordures, couleurs …) est du CSS simple. Vous pouvez voir si dessous un aperçu de ce que l'on obtient sur le navigateur.



Attention : Les ContentPlaceHolder de la première page maître sont TOUS remplacé par le contenu de la balise Content leur correspondant dans la seconde page maître et cela même s'il n'y a rien. Ainsi si vous faites un page maître avec un ContentPlaceHolder nommé BodyContent et qui contiendra les données spécifiques à chaque page, une fois votre seconde page maître passé ce ContentPlaceHolder n'existera plus. On ne pourra donc pas, depuis une page qui utilise la seconde page maître, ajouter les données principales. Pour contourner cela il faut tout simplement redéfinir ce ContentPlaceHolder :

Première page maître

```
<asp:ContentPlaceHolder runat="server" ID="BodyContent" />
```

Seconde page maître – Redéfinitiion de BodyContent

```
<asp:Content runat="server" ContentPlaceHolderID="BodyContent">
    <asp:ContentPlaceHolder runat="server" ID="BodyContent" />
</asp:Content>
```

Une fois ceci fais, vous pourrez faire appel au ContentPlaceHolder BodyContent depuis votre page ASPX.

2.7 Changer dynamiquement de page maître

Il est intéressant de pouvoir proposer plusieurs affichages différents de notre site aux utilisateurs. Pour cela il suffit tout simplement de changer de page maître. Le contenu restera le même mais l'affichage lui changera. Ainsi on peut facilement changer complètement la façon d'afficher le site mais les données resteront les mêmes. Seul problème : comment savoir quel page maître l'utilisateur veut utiliser sans avoir à lui redemander à chaque page. Et bien pour cela reportez-vous au chapitre sur la Gestion d'état qui traite les sessions, les cookies, les variables dans les url, les champs cachés … . Autant de manières différentes pour stocker cette donnée. Dans notre cas on va simplement utiliser la session.

Une chose importante à savoir pour pouvoir changer dynamiquement la page maître, c'est que l'on peut le faire depuis le code behind avec l'accesseur MasterPageFile de l'objet Page. Plus important encore : quand peut-on modifier la page maître ? ET bien on connait le cycle de la page. En sachant cela on remarque qu'on ne peut modifier la page maître qu'a l'évènement PreInit.

On garde AffichMenu.Master et on ajoute cela :

AffichMenuDroite.Master

```
<%@ Master Language="C#"
           AutoEventWireup="true"
           CodeBehind="AffichMenuDroite.master.cs"
           MasterPageFile="~/Principale.Master"
           Inherits="WebApplication1.AffichMenuDroite" %>

<asp:Content runat="server" ContentPlaceHolderID="Body">
    <div style="height: 300px; border: solid black 2px;">Content : Body
        <div style="height: 200px; border: solid 2px green; float: right;">
            <ul>
                <li>Lien 1</li>
                <li>Lien 2</li>
                <li>Lien 3</li>
                <li>Lien 4</li>
            </ul>
        </div>
        <div>
            <asp:ContentPlaceHolder ID="Body2" runat="server" />
        </div>
    </div>
</asp:Content>
```

Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Inherits="WebApplication1._Default"
         MasterPageFile="~/AffichMenu.Master"
         Title="Test"  %>

<asp:Content ID="Content1" ContentPlaceHolderID="Body2" Runat="Server">
Choisissez votre Master Page : <br />
    <asp:ListBox ID="ListBox1" runat="server" >
        <asp:ListItem Text="AffichMenuGauche.Master" Value="Gauche" />
        <asp:ListItem Text="AffichMenuDroite.Master" Value="Droite" />
    </asp:ListBox><br />
    <asp:Button ID="Button1" runat="server" Text="Button" OnClick="ChangerMaster" />

</asp:Content>
```

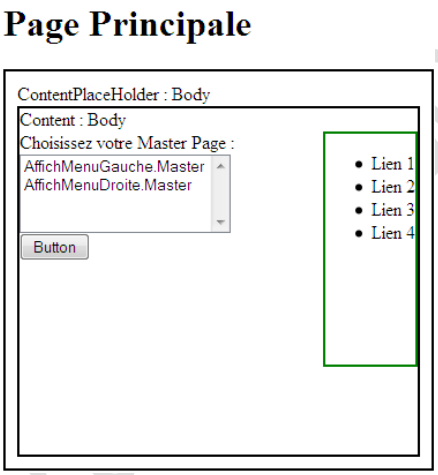
Default.aspx.cs

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_PreInit(object sender, EventArgs e)
    {
        if (Session["Master"] != null)
        {
            if (Session["Master"].ToString() == "Gauche")
                MasterPageFile = "AffichMenu.Master";
            if (Session["Master"].ToString() == "Droite")
                MasterPageFile = "AffichMenuDroite.Master";
        }
    }

    protected void ChangerMaster(object sender, EventArgs e)
    {
        Session["Master"] = ListBox1.SelectedValue;
        Response.Redirect("Default.aspx");
    }
}
```

Default.aspx.vb

```
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.PreInit
    If Session("Master") <> Nothing Then
        If Session("Master").ToString() = "Gauche" Then
            MasterPageFile = "AffichMenu.Master"
        End If
        If Session("Master").ToString() = "Droite" Then
            MasterPageFile = "AffichMenuDroite.Master"
        End If
    End If
End Sub

Protected Sub ChangerMaster(ByVal sender As Object, ByVal e As EventArgs)
    Session("Master") = ListBox1.SelectedValue
    Response.Redirect("Default.aspx")
End Sub
```

Voici ce que cela donne après avoir choisi le menu à droite et avoir appuyé sur le bouton :



3 Thèmes

En ASP.NET on peut créer des thèmes, qui sont un équivalent à une feuille de style CSS. C'est-à-dire qu'un thème va permettre de choisir la façon de s'afficher des éléments de la page comme leurs couleurs, leurs formes, leurs tailles …

3.1 Introduction

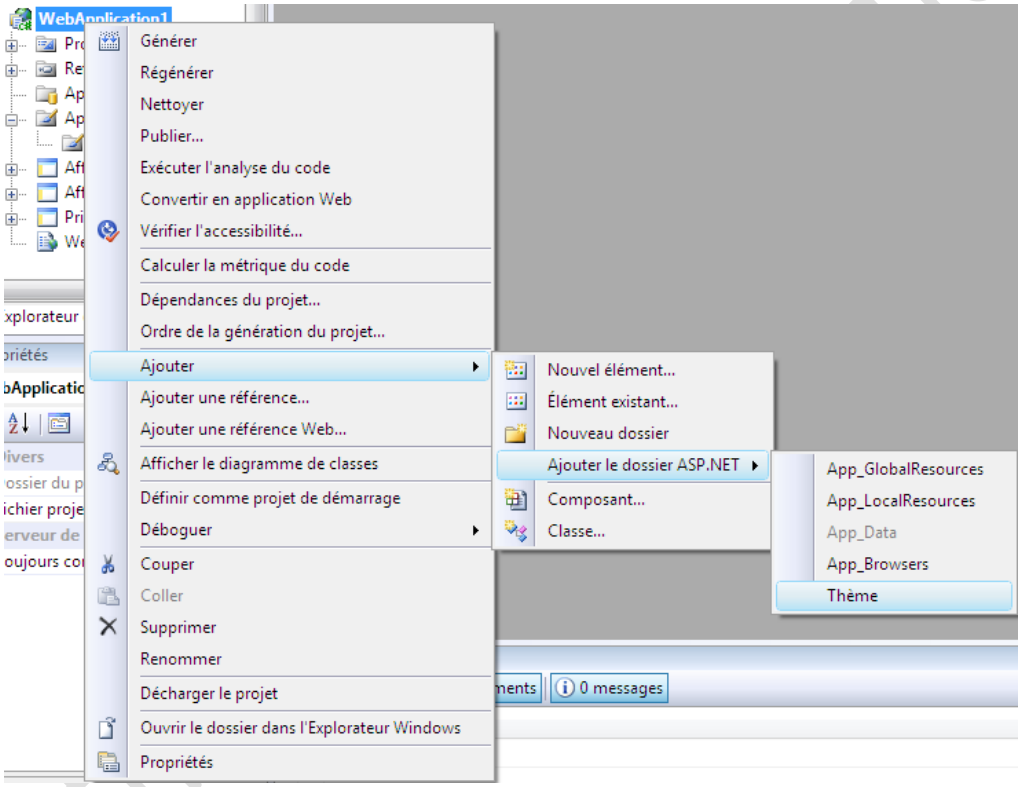
Un thème en ASP.NET est constitué de plusieurs types de fichier : CSS, skin, images et autres ressources de ce types. En quelque sorte, un thème est un « super CSS » qui permet de stocker toutes les ressources pour la configuration des contrôles. Mais alors quelles sont les différences avec un fichier CSS simple ? Est bien tout d'abord il y a une grande différence dans l'ordre d'application. Lors de la génération de la page, sera d'abord appliqué le fichier CSS, puis éventuellement le CSS ce trouvant entre les head, et ensuite le CSS qui se trouve sur la balise. Vient après tout cela l'application du thème. Ce qui revient à dire que quoi que l'on fasse ou spécifie en CSS, le thème sera toujours persistant et sera le dernier appliqué. Néanmoins on peut tout de même le désactiver, soit

pour un contrôle spécifique, soit pour une page défini, soit pour un ensemble de page. Il peut y avoir plusieurs thème existant qui pourront être appliqués dynamiquement.

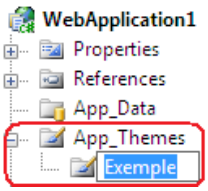
Voyons maintenant où se trouve les thèmes dans notre arborescence de l'explorateur de solution et comment en créer un nouveau.

3.2 Création d'un thème

Tous les thèmes en *ASP.NET* sont regroupés dans un même dossier : *APP_Themes*. Par défaut il n'existe pas dans le projet. Pour l'ajouter faites un clic droit sur votre projet et choisissez *Ajouter*, puis faites "*Ajouter le dossier ASP.NET*" et enfin *Theme* comme vous pouvez le voir sur l'image ci-dessous.



Vous aurez dans votre explorateur de solution un dossier APP_Themes qui devrait s'ajouter, contenant un autre dossier que Visual Studio va vous proposer de nommer. Pour ajouter un autre thème, faites un clic droit sur APP_Themes, et ajouter un Nouveau dossier.



Pour ajouter des ressources dans un thème, il faut faire un clic droit dessus et choisir Ajouter puis Nouvel élément. On peut rajouter un fichier CSS, un skin, un fichier XML … Vous connaissez quasiment tous ce qui est utilisé à l'exception des fichiers skin. Le fichier css sert à définir le style pour du HTML ou des éléments ayant un ID ou une Class défini. Le fichier skin va, lui, permettre de définir le style pour un contrôle précisé. C'est-à-dire que l'on peut définir entièrement l'apparence des boutons (par exemple) et ainsi à chaque fois que l'on ajoutera un bouton qui possédera ce

thème, le bouton aura déjà une apparence de défini. Ce système est très pratique puisque grâce à lui, une fois le style de nos contrôles défini, on ne s'occupe quasiment plus de l'apparence que chaque contrôle va prendre.

Voyons un peu à quoi un fichier skin ressemble par défaut lorsqu'on le créer :

Fichier skin : Skin1.skin

```
<%--
Modèle d'apparence par défaut. Les apparences sont fournies uniquement à titre
d'exemple.

1. Apparence de contrôle nommée. Le SkinId doit être défini de manière unique, car
 un type de contrôle d'un même thème ne peut pas compter de SkinId dupliqués.

<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >
   <AlternatingRowStyle BackColor="Blue" />
</asp:GridView>

2. Apparence par défaut. Le SkinId n'est pas défini. Une seule apparence
de contrôle par défaut par type de contrôle est autorisée dans le même thème.

<asp:Image runat="server" ImageUrl="~/images/image1.jpg" />
--%>
```

Le fichier skin par défaut contient déjà des explications sur comment créer un skin. Il suffit donc de mettre le contrôle quasiment comme si c'était une page ASPX. Quasiment, parceque vous n'avez pas besoin de mettre les propriétés Text ect … . La seule propriété obligatoire est le runat="server". Pour commencer ce skin nous allons y placer le skin du button que nous utiliserons dans la partie suivante :

Fichier skin : Skin1.skin

```
<asp:Button runat="server" Font-Bold="true" ForeColor="Blue" />
```

C'était un exemple de ce que l'on va avoir dans notre fichier skin. Il faut comprendre qu'on aura les skins de plusieurs contrôles sur chaque fichier skin (on ne fait pas un skin par contrôle).

Attention : Il ne faut pas qu'il y ait d'ID dans le skin.

3.3 Appliquer un thème

Maintenant que l'on a appris à créer un thème, nous allons voir comment l'appliquer sur les pages qui nous intérèsse.

3.3.1 Dans le Web.config

Dans le *Web.config* on va ajouter une propriété à la balise *pages* : *theme* comme dans l'exemple ci-dessous. "Exemple" correspond au nom du dossier contenant le thème dans *APP_Themes*.

Web.config

```
<system.web>
  <pages theme="Exemple"/>
</system.web>
```

3.3.2 Dans une page

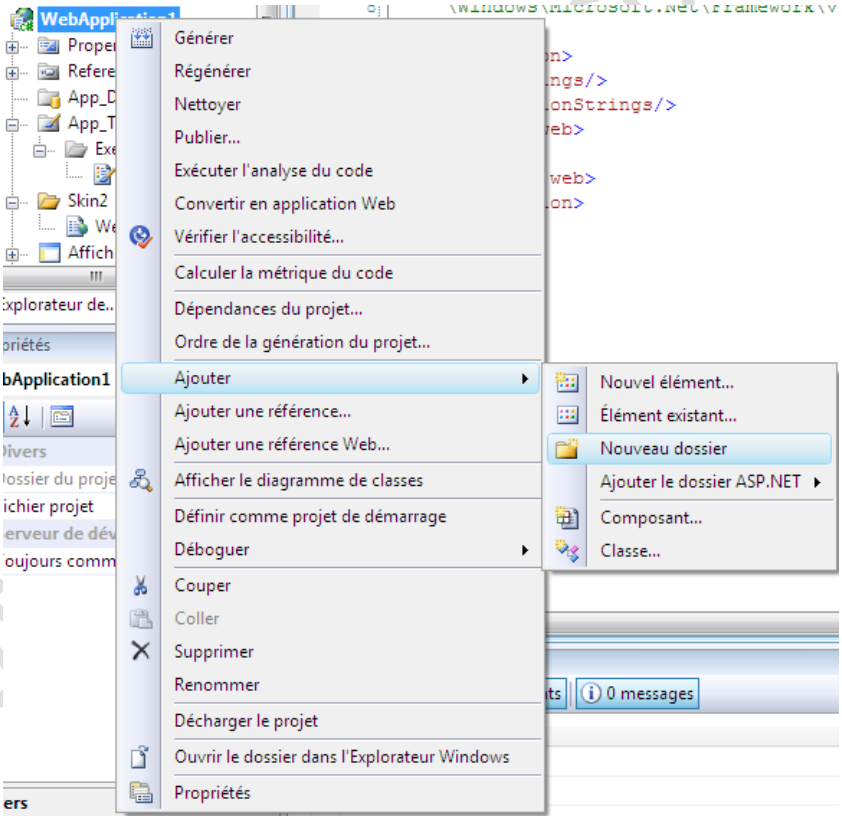
On peut aussi spécifier le thème dans la directive *Page* d'une page *ASPX*.

Web.config

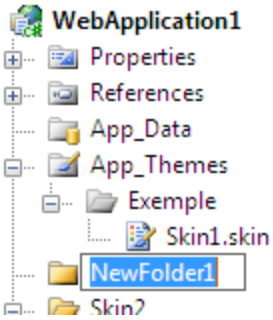
```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Theme="Exemple"
         Inherits="WebApplication1.Default" %>
```

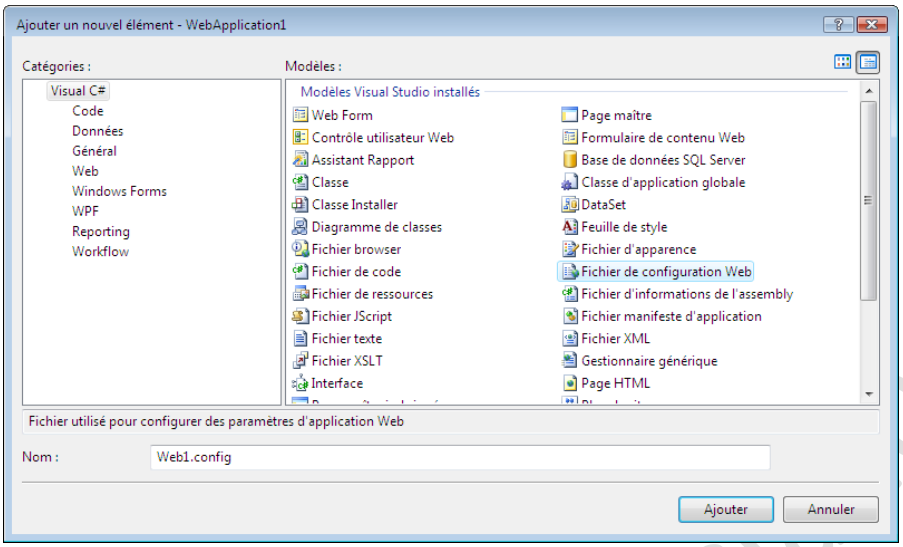
3.3.3 Pour un groupe de pages

Pour cela on va regrouper les pages qui doivent avoir un thème différent du reste du site dans un dossier. Clic droit sur le nom du projet, *Ajouter*, *Nouveau dossier*.



Vous aurez alors un dossier qu'il vous sera proposé de nommer. Dans ce dossier mettez les pages qui vous intéressent. Ensuite il va falloir rajouter un *Web.config* dans ce dossier. Ce *Web.config* ne s'appliquera que dans ce dossier. Pour cela ajouter un fichier de configuration web.





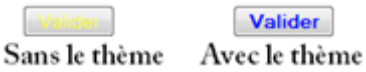
3.4 Exemple

On utilise le thème et le skin créé plus tôt. Voici ce que contient notre page Default.aspx :

Web.config

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Theme="Exemple"
         Inherits="WebApplication1.Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Page sans titre</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Valider" ForeColor="Yellow" />
    </div>
    </form>
</body>
</html>
```

Vous remarquez que la couleur du texte du bouton est bleu comme défini dans le thème malgrès la spécification du jaune dans la balise. Maintenant enlevez la propriété Theme dans la directive *Page*. Vous remarquerez alors que le texte du bouton est maintenant jaune (voir l'image ci-contre).



Sans le thème Avec le thème

3.5 Utiliser les SkinID

Les *SkinID* servent à la même chose que les *ID* et *Class* en CSS. C'est-à-dire qu'il va nous permettre de définir plusieurs skin différents pour les boutons (dont le bouton par défaut et les spéciaux) et de pouvoir choisir parmis eux lequel sera utilisé pour un bouton particulier. Cette méthode se retrouve

sur beaucoup de contrôle serveur Web (mais pas sur les Literal puisqu'ils n'ajoutent aucun code et ne font que placer du texte "litéralement »). En revanche on ne peut pas configurer les skins pour les contrôles HTML quelqu'il soit, on n'y trouvera donc pas cette méthode.

Nous allons créer deux Labels, le premier aura le skin par défaut que nous allons définir. Le second quant à lui possèdera le second skin qui sera un skin nommé "Important".

Skin1.skin

```
<asp:Label runat="server" ForeColor="Blue" />
<asp:Label runat="server" SkinID="Important" ForeColor="Red" />
```

Default.aspx

```
<asp:Label ID="Table1" runat="server" Text="Ceci est un texte par défaut" /><br />
<asp:Label ID="Table2" runat="server" SkinID="Important" Text="Ceci est un texte
avec le SkinID Important" />
```

Vous pouvez voir sur l'image ci-contre le résultat de ce code. Maintenant une explication de celui-ci : le skin pour le Label qui ne possède pas la propriété SkinID sera le skin par défaut pour tous les Label. Celui possédant un SkinID ne va être spécifique qu'a un Label possédant le même SkinID. Il est important de préciser que le skin par défaut ne sera pas hérité par les skins spécifiques. Ainsi si les skins spécifiques reprennent un peu de l'affichage du skin par défaut, il faudra le refaire. Toujours est-il que l'on aura besoin de le faire qu'une fois au lieu de refaire l'affichage à chaque contrôle.

Ceci est un texte par défaut
Ceci est un texte avec le SkinID Important

3.6 Désactiver le thème

Il est possible de désactiver le thème juste pour un contrôle ou pour une page entière. Pour cela on utilise la propriété EnableTheming que l'on retrouve sur les contrôles serveur Web et sur la directive page. Pour désactiver le thème, on le met à False. En revanche il n'est pas possible de n'activer le thème que sur un contrôle.

```
<%@ Page Language="C#"
        AutoEventWireup="true"
        CodeBehind="Default.aspx.cs"
        EnableTheming="false"
        Inherits="WebApplication1.Default" %>
```

```
<asp:Label ID="Label1" runat="server"
   EnableTheming="false" Text="Ceci est un texte par défaut" />
```

3.7 Changer le thème dynamiquement

Comme pour changer de page maître, pour changer dynamiquement de thème il faut utiliser le *Page_PreInit* car après cet évènement l'environnement de travail est déjà défini et le thème n'est plus modifiable.

Dans l'exemple qui suit nous allons passer la variable contenant le thème choisit par l'utilisateur par l'url. Si on avait du retenir le choix de l'utilisateur on aurait, par exemple, pu stocker cette variable

dans une base de données. Ici on garde le premier thème qu'on a fait (soit Exemple) et on en ajoute un second nommé TestChangement.

Fichier skin de TestChangement

```
<asp:Label runat="server" ForeColor="Yellow" />
<asp:Label  runat="server"  SkinID="Important"  ForeColor="Green"  BorderColor="Aqua"
BorderWidth="2px" />
```

Default.aspx

```
<%@ Page Language="C#"
         AutoEventWireup="true"
         CodeBehind="Default.aspx.cs"
         Theme="Exemple"
         Inherits="WebApplication1.Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Page sans titre</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Ceci est un texte par défaut"
/><br />
        <asp:Label ID="Label2" runat="server" SkinID="Important" Text="Ceci est un
texte avec le SkinID Important" /><br /><br />
        <asp:HyperLink runat="server"
NavigateUrl="~/Default.aspx?Theme=Exemple">Exemple</asp:HyperLink><br />
        <asp:HyperLink runat="server"
NavigateUrl="~/Default.aspx?Theme=TestChangement">TestChangement</asp:HyperLink>
    </div>
    </form>
</body>
</html>
```

Dans ce fichier on remarque que par défaut on utilise le thème Exemple. Mais comme après on change le thème par le code behind et que ce dernier est prioritaire, cela ne posera aucun problème.

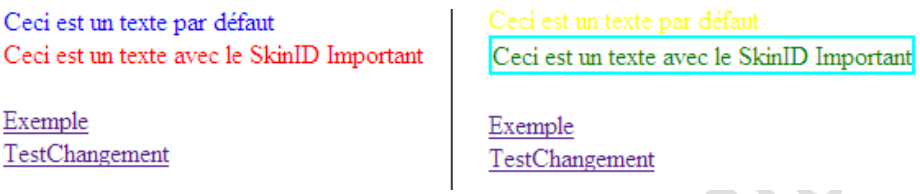
C# - Default.aspx.cs

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (Request.QueryString["Theme"] != null)
    {
        Page.Theme = Request.QueryString["Theme"].ToString();
        // Ici nous avons choisi l'option de facilité. Il est évident qu'il
        // faudrait vérifier ce que contient ce paramètre en utilisant par exemple
        // un if ou un switch pour éviter au maximum une faille.
    }
}
```

VB.NET - Default.aspx.vb

```
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.PreInit
    If Request.QueryString("Theme") <> Nothing Then
        Page.Theme = Request.QueryString("Theme").ToString()
        ' Ici nous avons choisi l'option de facilité. Il est évident qu'il
        ' faudrait vérifier ce que contient ce paramètre en utilisant par exemple
        ' un if ou un switch pour éviter au maximum une faille.
    End If
End Sub
```

Et le résultat que l'on obtient avec le premier et le second lien :



4 Les profils utilisateurs

De plus en plus actuellement, les applications se doivent d'être personnalisables par le client sans avoir des notions en informatiques avancées ; et les applications web ne font pas exception à cette règle!

Pour cela, l'ASP.NET utilise les profils utilisateurs qui permettent la sauvegarde des informations dans le contexte http de l'application et, à termes, dans une base de données. Pour cela, vous avez accès à deux types de profils :

- Les profils anonymes. Ils peuvent être utilisés pour des internautes itinérants ou sur des sites qui n'utilisent aucune identification. Ce type de profil est basé sur un identifiant unique qui est généré au démarrage de la session de navigation et qui est échangé via un cookie (dont la durée de vie est de 70 jours après la dernière visite de votre site web) ou via l'adresse URL. Dans tous les cas, l'inconvénient de cette méthode est que l'identifiant peut être perdu si les cookies sont effacés ou si le navigateur est fermé.

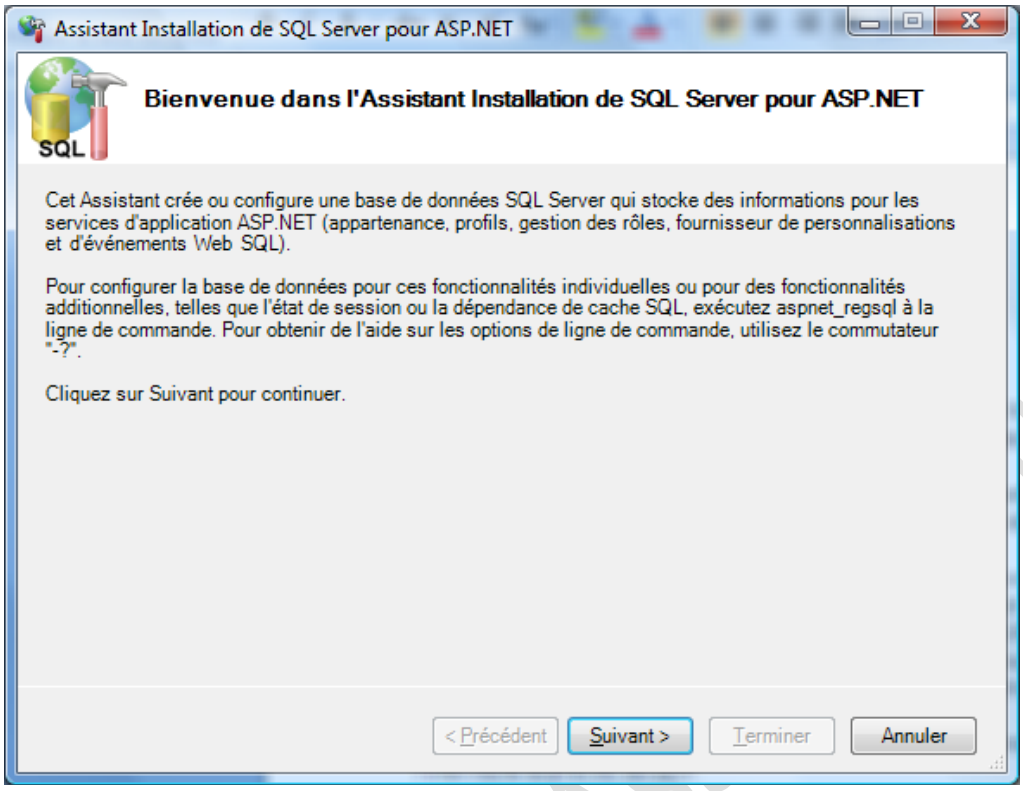
- Les profils authentifiés qui nécessitent obligatoirement une identification de l'internaute auprès du serveur.

Il sera tout à fait possible de passer d'un statut anonyme au statut authentifié. Si vous souhaitez en savoir d'avantage à ce sujet, je vous recommande ce lien.

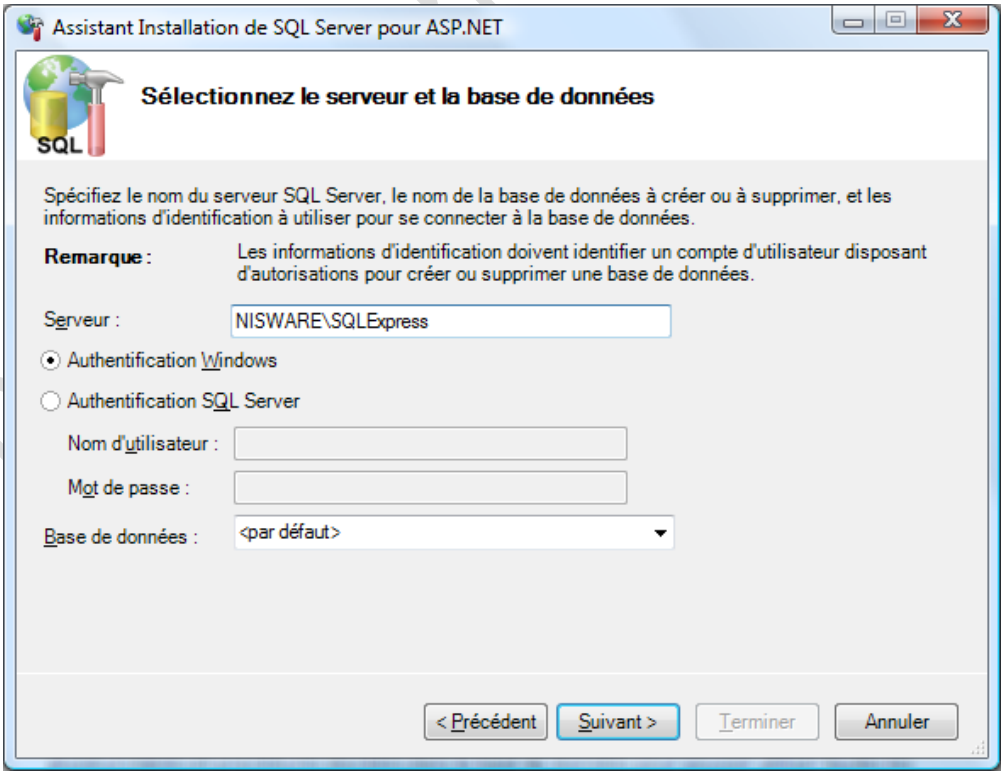
4.1 Configuration des profils

Avant de pouvoir utiliser l'un ou l'autre type de profil, vous devez le configurer dans le fichier de configuration web de votre application et il vous faudra également une base de données correctement initialisée.

Pour la base de données, nous utiliserons une base SQL Server 2005 Express. Une fois la base de données installée et accessible, nous l'initialiserons à l'aide de l'outil aspnet_regsql.exe se trouvant dans C:\Windows\Microsoft.NET\Framework\v2.0.50727. Cet outil va permettre de créer plusieurs tables et procédures stockées dans la base de données pour pouvoir utiliser toutes les fonctionnalités proposées par ASP.NET :



A l'écran suivant, laissez coché l'option "Configurer SQL Server pour les services et applications". Lorsque vous arrivez sur le choix du serveur, entrez tous les paramètres nécessaire à l'accès à votre serveur de données. Dans notre exemple, la base de données est locale et son instance se nomme "SQLExpress" :



Une fois votre base de données initialisée, il vous faudra créer la chaine de connexion permettant d'accéder à la base de données :

```
<!—Web.config-->
<connectionStrings>
    <clear/>
    <add connectionString="Data Source=Nisware\SQLExpress;Initial
Catalog=Profile;Integrated Security=True;Pooling=False"
providerName="SqlServer" name="Connexion"/>
</connectionStrings>
```

Dans cet exemple, la chaine de connexion nommée "Connexion" permet d'accéder à la base de données dont l'instance a été nommée "SQLExpress" sur l'ordinateur "Nisware".

Entre les balises System.web, il faudra ajouter la balise Profile pour activer les profils en spécifiant quelle base de données sera utilisée lors de l'enregistrement des informations. Le nom du fournisseur de données à utiliser par défaut sera le nom attribué à l'un des fournisseurs placé dans les balises Providers :

```
<!—Web.config-->
<system.web>
   <profile enabled="true" defaultProvider="ProfileProvider">
      <providers>
        <clear/>
        <add name="ProfileProvider"
type="System.Web.Profile.SqlProfileProvider"
connectionStringName="Connexion"/>
      </providers>
   </profile>
</system.web>
```

4.2 Les profils anonymes

Pour activer les profils anonymes, il faudra ajouter une ligne dans le fichier de configuration web :

```
<!—Web.config-->
<system.web>
    <anonymousIdentification enabled="true"/>
</system.web>
```

Cette ligne permet d'indiquer que, par défaut, les internautes auront un profil anonyme attribué.

Ensuite, vous devrez spécifier, dans la balise profile, la balise properties contenant le nom des champs disponible :

```
<!—Web.config-->
<profile enabled="true" defaultProvider="ProfileProvider"
automaticSaveEnabled="true">
   <providers>
      <clear/>
      <add name="ProfileProvider"
type="System.Web.Profile.SqlProfileProvider"
connectionStringName="Connexion"/>
   </providers>
   <properties>
      <clear />
      <add name="Couleur" allowAnonymous="true"/>
      <add name="Nom" allowAnonymous="true"/>
   </properties>
</profile>
```

Le type par défaut de chaque champs est System.String. Si vous avez besoin d'un autre type, vous pouvez l'indiquer sur chaque balise add grâce à l'attribut type. Ici, nous avons activé les profils

anonymes et nous indiquons que les champs Nom et Couleur peuvent être utilisés sur les profils anonymes.

Dans le code-behind, nous pouvons désormais accéder à chacun des profils grâce à la propriété `Profile` de l'objet HttpContext associé à chaque session de navigation de notre site :

C#

```
public partial class _Default : System.Web.UI.Page
{
       protected void Page_Load(object sender, EventArgs e)
       {
              base.Context.Profile["Couleur"] = "Black";
             base.Context.Profile["Nom"] = "Inconnu";
       }
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    MyBase.Context.Profile("Couleur") = "Black"
    MyBase.Context.Profile("Nom") = "Inconnu"
End Sub
```

Ici, nous nous contentons d'initialiser les deux champs du profil anonyme à des valeurs par défaut lorsque la page s'est chargée. Par la suite, on pourra récupérer ces paramètres pour modifier les couleurs de différents contrôles par exemple.

Un peu plus haut dans cette partie, nous avions vu qu'il était nécessaire d'avoir une base de données disponible pour pouvoir utiliser les profils. Cette base de données va permettre de stocker les paramètres contenus dans le profil actuel juste en appelant la méthode Save de la propriété Profile.

4.3 Les profils authentifiés

Ce type de profil ne diffère en rien des profils anonymes au niveau de leur utilisation. Pour créer un profil authentifié, il suffira de supprimer l'attribut allowAnonymous (ou de le passer à False) dans la configuration des champs de profils :

```
<!—Web.config-->
<properties>
      <clear />
      <add name="Couleur" allowAnonymous="true"/>
      <add name="Nom" allowAnonymous="true"/>
      <add name="Argent" type="System.Int32"/>
</properties>
```

Ici, nous avons simplement ajouté à la liste des champs disponible dans les profils, le champ "Argent" qui ne peut être accessible sans s'être authentifié auparavant. Si nous tentons d'y accéder en écriture alors que l'internaute courant n'est pas authentifié, une exception sera levée.

Note : Les processus d'authentification seront abordés dans un chapitre ultérieur.

4.4 Les groupes de paramètres

Pour des raisons de clartés, il est possible de créer des groupes de propriétés. Pour cela, il vous suffira d'ajouter une ou plusieurs balises Group dans la balise Properties, d'y spécifier l'attribut name et d'ajouter des propriétés à l'aide des balises add :

```
<!—Web.config-->
<properties>
      <clear />
      <group name="Couleurs">
          <add allowAnonymous="true" name="BackColor"/>
          <add allowAnonymous="true" name="FontColor"/>
      </group>
      <add name="Nom" allowAnonymous="true" type="System.String"/>
      <add name="Argent" type="System.Int32"/>
</properties>
```

Dans cet exemple, nous avons passé la propriété Couleur en groupe de propriété du même nom. A l'intérieur, nous avons deux propriétés nommées BackColor et FrontColor.

Pour modifier ces propriétés, nous passerons par la méthode GetProfileGroup et nous accédons aux propriétés de la même façon que si elles n'étaient pas contenues dans un groupe :

C#

```
protected void Page_Load(object sender, EventArgs e)
{
      base.Context.Profile.GetProfileGroup("Couleurs")["FontColor"] = "Vert";
     base.Context.Profile["Nom"] = "Gaspard";
}
```

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
Me.Load
    MyBase.Context.Profile.GetProfileGroup("Couleurs")("FontColor") = "Vert"
    MyBase.Context.Profile("Nom") = "Gaspard"
End Sub
```

5 Web Parts

Lors de l'affichage d'une page Web, on peut cliquer/selectionner tous les éléments apparents ou naviguer facilement entre les pages. Ceci étant, on peut se poser la question de savoir comment un utilisateur authentifié pourrait créer un site d'accueil de type MySpace. C'est-à-dire que sans aucune connaissance en développement, on voudrait que l'utilisateur puisse ajouter des éléménts dans sa page d'accueil : un texte d'accueil, un diaporama, un formulaire pour livre d'or, etc.

5.1 Introduction

Les Web Parts sont une solution qui va nous permettre de créer des zones sur notre page dans lesquelles on va pouvoir ajouter des contrôles, les supprimer, les réduire, les déplacer, les modifier, sans avoir à manipuler du code.

5.2 Créer une Web Part

La première étape pour créer une Web Part est d'ajouter au début de notre page un *WebPartManager*, comme son nom l'indique, c'est cet élément qui va permettre de gérer tous les éléments de notre Web Part. Pour ajouter un *WebPartManager*, il suffit soit de l'ajouter via la boite à outil dans la section WebParts soit l'ajouter manuellement a l'aide de l'instruction suivante :

ASPX

```
<form id="form1" runat="server">
    <asp:WebPartManager runat="server" ID="WebPartManager1">
    </asp:WebPartManager>
</form>
```

Le *WebPartManager* dispose de propriétés qui vont nous permettre entre autre de switcher entre les différents modes d'utilisation des Web Parts que nous allons créer.

Une fois que nous avons ajouté notre *WebPartManager*, nous pouvons ajouter des *WebPartZone* n'importe où dans notre page. C'est dans ces *WebPartZone* que nous allons pouvoir ajouter des contrôles. Pour créer des *WebPartZone* deux solutions s'offrent à nous.

La première est la plus simple et se fait en mode Design dans Visual Studio, il suffit d'effectuer un Drag&Drop d'un *WebPartZone* de la boite à outil (section WebParts).

La seconde se fait via le code ASPX. Ainsi après notre *WebPartManager* on va créer notre *WebPartZone* comme montré dans l'exemple ci-dessous :

ASPX

```
<form id="form1" runat="server">
    <asp:WebPartManager runat="server" ID="WebPartManager1">
    </asp:WebPartManager>
    <asp:WebPartZone ID="Gauche" runat="server">
    </asp:WebPartZone>
</form>
```

Astuce :

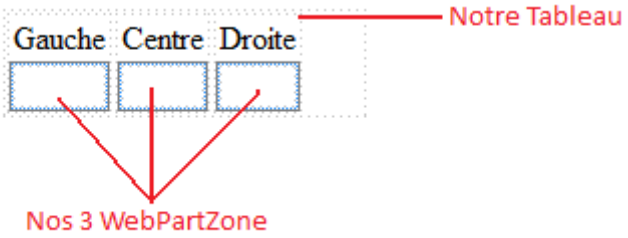
> Afin de mettre en page vos différents *WebPartZone* vous pouvez les mettre dans un tableau et créer un *WebPartZone* dans chaque cellule.

Pour les besoins du cours nous allons créer trois *WebPartZone* dans un tableau.

ASPX

```
<asp:WebPartManager ID="WebPartManager1" runat="server">
</asp:WebPartManager>
<table>
    <tr>
        <td>
            <asp:WebPartZone ID="Gauche" runat="server">
            </asp:WebPartZone>
        </td>
        <td>
            <asp:WebPartZone ID="Centre" runat="server">
            </asp:WebPartZone>
        </td>
        <td>
            <asp:WebPartZone ID="Droite" runat="server">
            </asp:WebPartZone>
        </td>
    </tr>
</table>
```

Voici un aperçu de ce que vous devriez voir en mode Design :



Pour ajouter des contrôles dans nos *WebPartZone* en mode design, il suffira d'effectuer un Drag&Drop des contrôles sur la *WebPartZone* de notre choix, le code ASPX supplémentaire sera généré pour nous.

Si vous désirer malgré tout ajouter vos contrôles en mode Source il vous faudra rajouter la balise `ZoneTemplate` dans laquelle vous ajouterez vos contrôles.

Exemple :

ASPX

```
<asp:WebPartZone ID="Gauche" runat="server">
    <ZoneTemplate>
       <asp:Button ID="Button1" runat="server" Text="Button" />
    </ZoneTemplate>
</asp:WebPartZone>
```

Comme vous pouvez le constater avec cet exemple on peut parfaitement ajouter des contrôles prédéfinis dans notre *WebPartZone* tel que des *Label*, des *TextBox*, ou encore des *Button*.

5.3 Utilisation des Web Parts

Nous avons vu comment créer des Web Parts, cela dit les possibilités de cet outil ne s'arrêtent pas là. En effet l'une des particularités des Web Parts c'est que les contrôles contenues dans celle-ci sont déplaçable et modifiable facilement par l'utilisateur.

De plus il existe différent mode d'affichage que va gérer le *WebPartManager* dont voici un tableau.

BrowserDiplayMode	Affichage classique de la page dans le navigateur Web.
DesignDisplayMode	Permet de déplacer les contrôles en Drag&Drop d'une *WebPartZone* à une autre (Fonctionne surtout avec Internet Explorer, pour les autres il faudra de l'AJAX).
CatalogDisplayMode	Permet d'ajouter des contrôles à nos WebPartZone.
EditDisplayMode	Permet d'accéder aux modifications de nos contrôles : apparences, propriétés, position, comportement.
ConnectDisplayMode	Permet de connecter un contrôle avec un autre afin d'afficher des informations croisées (Ex : Une ListBox affiche des départements, une GridView va afficher la liste des employés correspondant à ce département).

Nous allons voir en détail chacun de ces modes. Cependant si vous désirez revenir à l'affichage premier de votre page vous allez devoir effectuer un Reset de vos personnalisations. La chose est simple, il vous suffira d'appeler la méthode `ResetPersonalizationState()` dans la partie `Personalization` de votre *WebPartManager*. Exemple sur l'évènement *onClick* d'un *Button* :

C#

```
protected void WebPartReset_Click(object sender, EventArgs e)
{
   WebPartManager1.Personalization.ResetPersonalizationState();
}
```

VB.NET

```
Protected Sub WebPartReset_Click(ByVal sender As Object, ByVal e As EventArgs)
    WebPartManager1.Personalization.ResetPersonalizationState()
End Sub
```

5.3.1 BrowserDisplayMode

Le *BrowserDisplayMode* est le mode par défaut et affiche simplement notre WebPart en ne laissant comme option que la réduction de nos contrôles ou leurs fermetures. On peut cependant forcer ce mode à être activé, cela est utile quand on a switché sur un autre mode.

Pour activer ce mode il suffit de mettre à jour la propriété *DisplayMode* de notre *WebPartManager* de notre page courante que l'on pourrait gérer en la liant à l'évènement *OnClick* d'un *Button* par exemple.

C#

```
protected void WebPartNormal_Click(object sender, EventArgs e)
{
    WebPartManager1.DisplayMode = WebPartManager.BrowseDisplayMode;
}
```

```
VB.NET

Protected Sub WebPartReset_Click(ByVal sender As Object, ByVal e As EventArgs)
    WebPartManager1.Personalization.ResetPersonalizationState()
End Sub
```

5.3.2 DesignDisplayMode

Ce mode ne trouve son intérêt que sous Internet Explorer version 7.0 ou plus. En effet, il nous permet de déplacer nos contrôles en Drag&Drop (glisser/déposer) ce que les autres navigateurs ne peuvent faire (néanmoins pas sans de l'AJAX ce que nous n'utilisons pas ici). Par ailleurs il nous permet aussi de supprimer les contrôles ajoutés via le catalogue (que nous allons voir juste après).

De façon similaire au *BrowserDisplayMode*, il faut mettre à jour la propriété *DisplayMode* pour pouvoir accéder à ce mode (toujours via une méthode *OnClick*) :

```
C#

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

```
VB.NET

Protected Sub WebPartDesign_Click(ByVal sender As Object, ByVal e As EventArgs)
    WebPartManager1.DisplayMode = WebPartManager.DesignDisplayMode
End Sub
```

5.3.3 CatalogDisplayMode

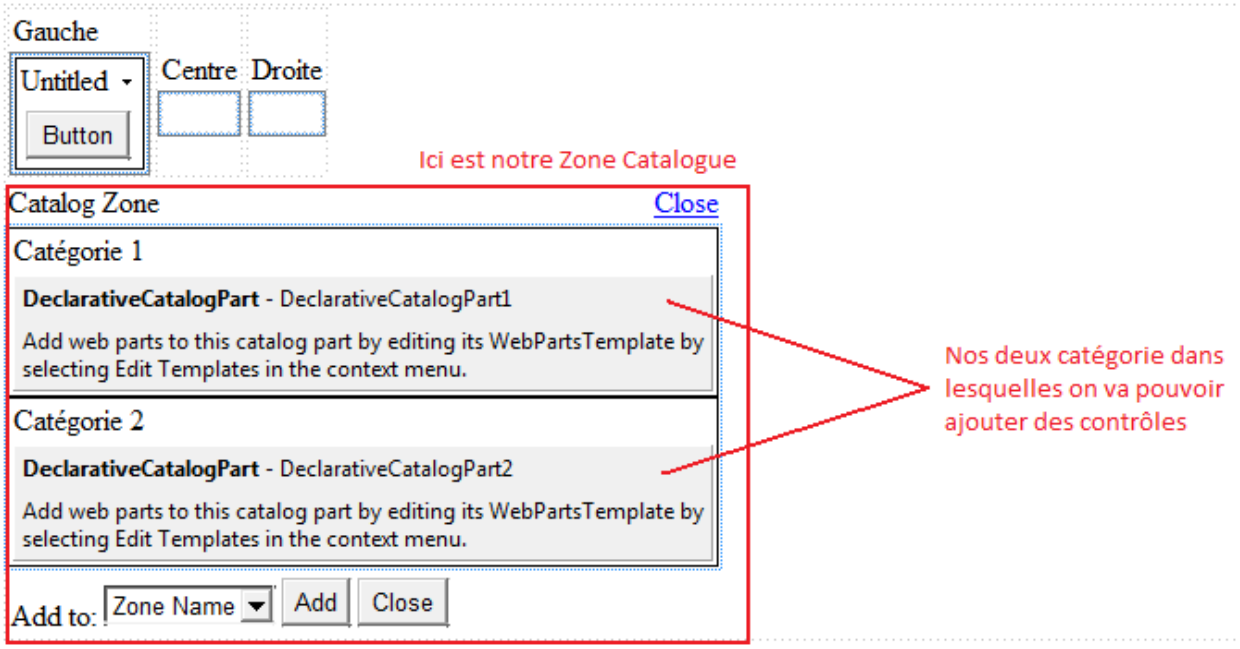
Ce mode nous permet d'ajouter des contrôles à nos *WebPartZone* que nous avons prédéfinis. Pour activer ce mode, il suffit de procéder comme précédemment pour les autres modes :

```
C#

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

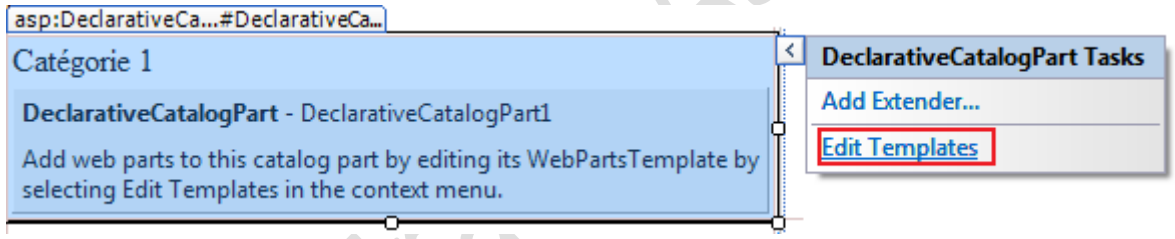
```
VB.NET

Protected Sub WebPartCatalog_Click(ByVal sender As Object, ByVal e As EventArgs)
     WebPartManager1.DisplayMode = WebPartManager.CatalogDisplayMode
 End Sub
```

Nous allons voir comment créer un catalogue et y ajouter des éléments.

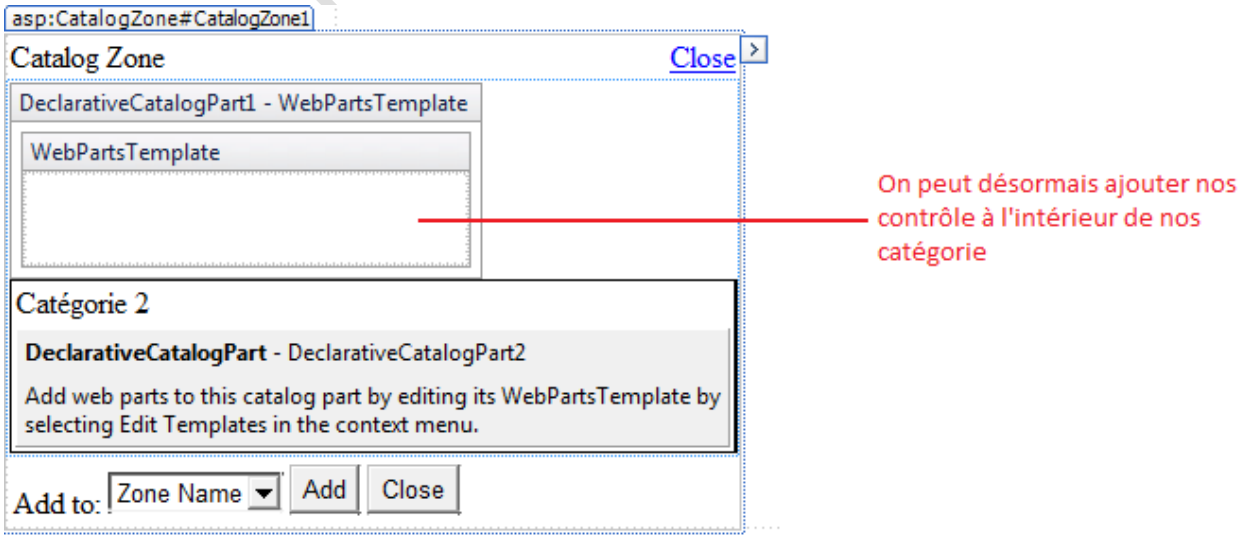
Pour ajouter un catalogue en mode design (de Visual Studio), il suffit de faire un Drag&Drop d'un *CatalogZone* à l'endroit souhaité sur notre page web (de préférence en dehors de nos *WebPartZone*). La zone catalogue ne s'affichera dans le navigateur Web que si le *CatalogDiplayMode* est activé. Puis il faut ajouter un *DeclarativeCatalogPart* dans notre *CatalogZone*, qui va nous permettre de créer une catégorie dans laquelle on pourra stocker des contrôles. Pour classifier nos contrôles que l'on va ajouter, il faut créer autant de *DeclarativeCatalogPart* que l'on veut créer de catégorie.



Comme il est expliqué dans les cadres, si vous voulez ajouter des contrôles vous devez sélectionnez l'Edition de Template comme le montre l'image ci-dessous.



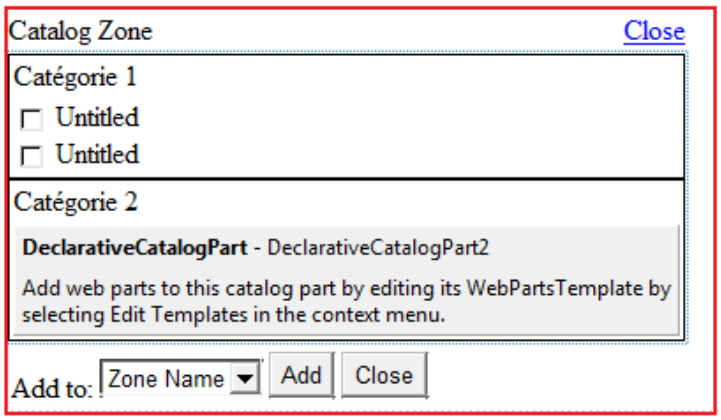
Une fois que nous avons activé l'édition de Template on peut ajouter nos contrôles en Drag&Drop à l'intérieur de notre catégorie.



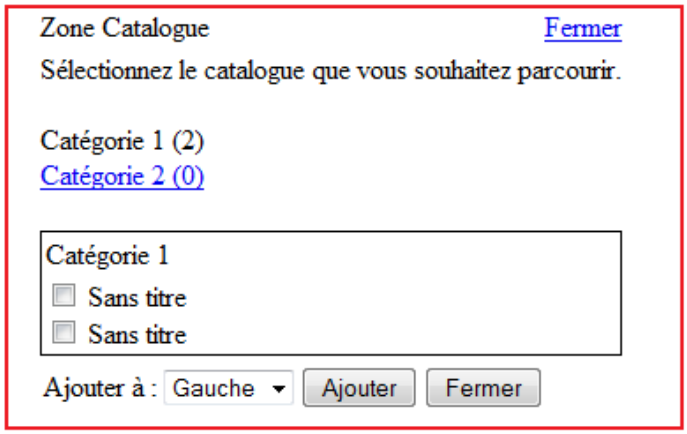
Testons avec une *TextBox* et un *Label*, rappelons que l'on préfèrera ajouter des Web User Control mais pour les besoins de l'exemple nous nous efforcerons de faire au plus simple.

Voici l'aperçu une fois ces opérations terminées :

Dans Visual Studio



Dans un navigateur



Astuce :

> Pour personnaliser le nom de vos catégories ou/et de vos contrôles, ajoutez une propriété `Title="Le nom que vous souhaitez"` dans chaque éléments (*DeclarativeCatalogPart, TextBox, Label, User Control* etc.).
>
> Le catalogue dispose d'une propriété `HeaderText="Mon Catalogue"` qui servira à donner un nom à votre catalogue.

Vous pouvez désormais donner la possibilité à l'utilisateur d'ajouter les contrôles que vous avez définis dans les *WebPartZone* que vous avez créés.

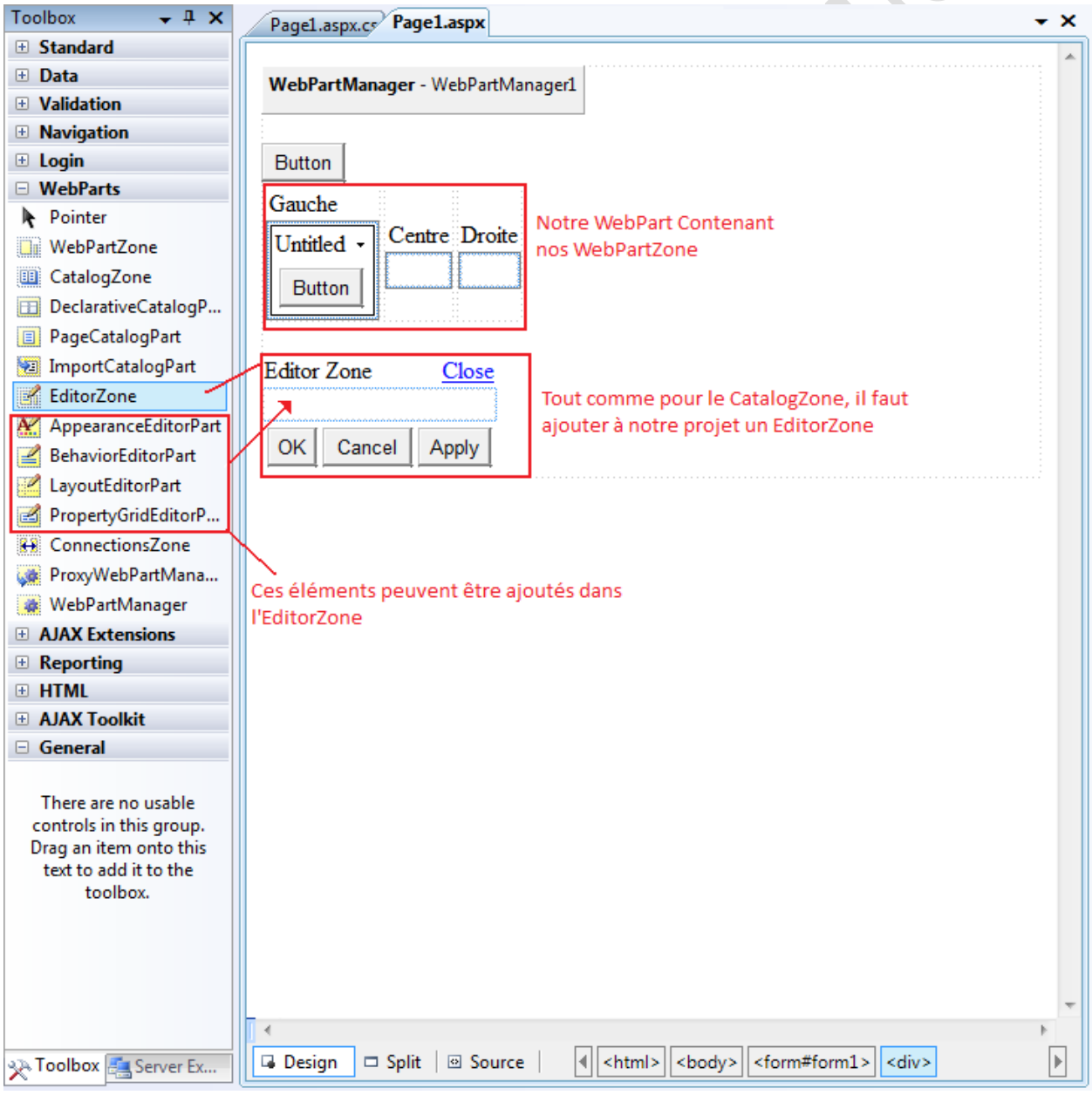
5.3.4 EditDisplayMode

Le mode *EditDisplayMode* va nous permettre de modifier les propriétés, le comportement et l'emplacement d'un contrôle choisi. Il permet entre autre de pallier au mode Design uniquement fonctionnel sous Internet Explorer. Le mode *EditDisplayMode* s'active de la même manière que pour les autres modes :

```
C#

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

```
VB.NET

Protected Sub WebPartEdit_Click(ByVal sender As Object, ByVal e As EventArgs)
    WebPartManager1.DisplayMode = WebPartManager.EditDisplayMode
End Sub
```

Tout comme pour le *CatalogDisplayMode* il va falloir ajouter un *EditorZone* à notre projet. L'image ci-dessous nous montre comment l'ajouter à notre projet et ce que l'on peut y mettre.



Ajouter des propriétés à la grille de propriété

Dans le mode édit, il est possible d'ajouter des propriétés personnalisées qui vont nous permettre de créer des accesseurs pour modifier des champs supplémentaires. Par exemple on souhaite créer une propriété qui va nous permettre de modifier un Label et qui apparaîtra dans la grille de propriétés du *EditorDisplayMode*. Pour ce faire il va nous falloir modifier le code behind de notre contrôle afin d'ajouter des propriétés. Notons maintenant que cela aurait été impossible si nous utilisions un contrôle prédéfini dans la boite à outil.

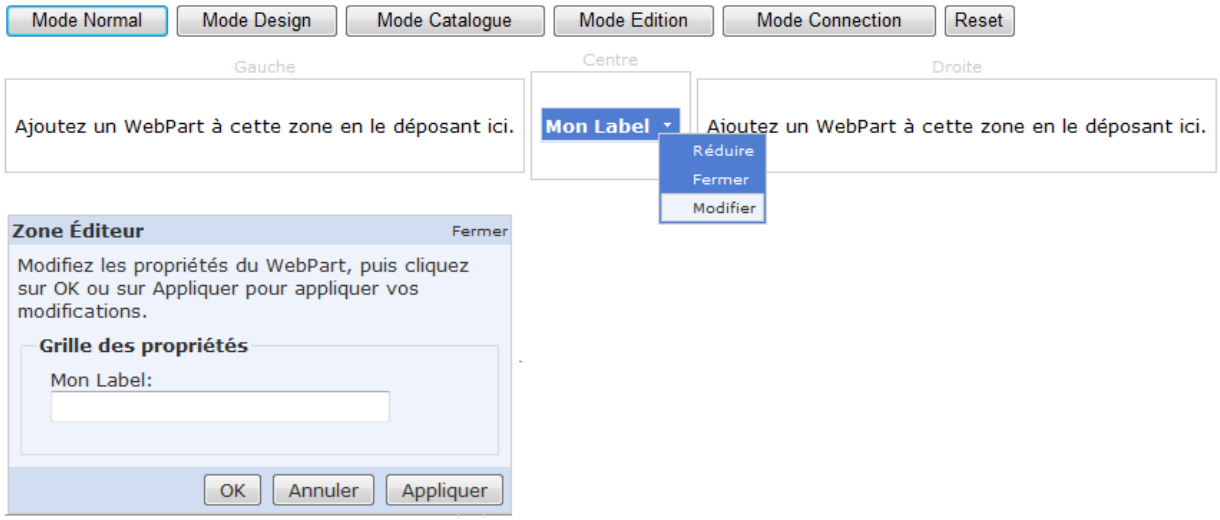
C# de notre contrôle

```
 [WebBrowsable(true)]
 [Personalizable(true)]
//Les deux paramètres ci-dessus sont
// indispensable pour créer la propriété
 [WebDescription("Parametrez le label")]
// -> Affiche un texte d'aide
 [WebDisplayName("Mon Label")]
// -> Ce qui sera affiché dans la grille
//    des propriétés
public string Texte2
// -> On déclare notre champs à modifier
//    ainsi que les accesseurs
{
    get
    {
        return Label1.Text;
    }
    set
    {
        if (value != null && value != "")
                    Label1.Text = value;
    }
}
```

VB.NET

```
<WebBrowsable(True), Personalizable(True), WebDescription("Parametrez le label"),
WebDisplayName("Mon Label")> _
Public Property Texte2() As String
    Get
        Return Label1.Text
    End Get
    Set(ByVal value As String)
        If value <> Nothing And value <> "" Then
            Label1.Text = value
        End If
    End Set
End Property
```

Il est évident que *Label1* est en fait le label sur ma page ASPX. Désormais, le contenu de notre label sera modifiable via le *PropertyGridEditorPart*.



5.3.5 ConnectDisplayMode

Le mode *ConnectDisplayMode* va nous permettre de lier des contrôles entre eux pour afficher des données croisées. Par exemple nous allons pouvoir créer une recherche filtrée sur une base de données en liant deux contrôles. De la même manière que pour l'*EditorZone* il vous faudra ajouter une *ConnectionsZone* a votre page contenant vos *WebParts*.

Ce mode est utilisable par le biais d'un contrat (une interface) qu'il va falloir faire passer entre deux contrôles : un *provider* (fournisseur) et un *consumer* (consommateur).

Le fournisseur sera le contrôle qui donnera la donnée au consommateur afin d'affiner la recherche par exemple.

Pour les besoins de l'exemple qui va suivre nous utiliserons une base de données contenant deux tables. Une table *members* qui contiendra des personnes membres de l'association **.NET France** :

Column Name	Data Type	Allow Nulls
id	int	☐
nom	varchar(50)	☑
prenom	varchar(50)	☑
adresse	nvarchar(MAX)	☑
poste	varchar(50)	☑

Puis une table *postes* qui répertoriera les différents postes existant de l'association :

Column Name	Data Type	Allow Nulls
id	int	☐
poste	varchar(50)	☑

Remplissez comme bon vous semble vos tables afin qu'il y ait des données sur lesquelles on va pouvoir requêter. N'oublier pas, en remplissant la table *postes,* qu'il vous faudra lister les différents postes que vous avez crées pour les membres de .NET France (ex : Fondateur, Manageur, Développeur, Stagiaire).

Grace à ces deux tables nous allons vous montrer comment deux contrôles affichant des données de façon indépendante, vont pouvoir dialoguer, être connecté.

Création des contrôles

Nous allons commencer par créer deux *WebUserControl*.

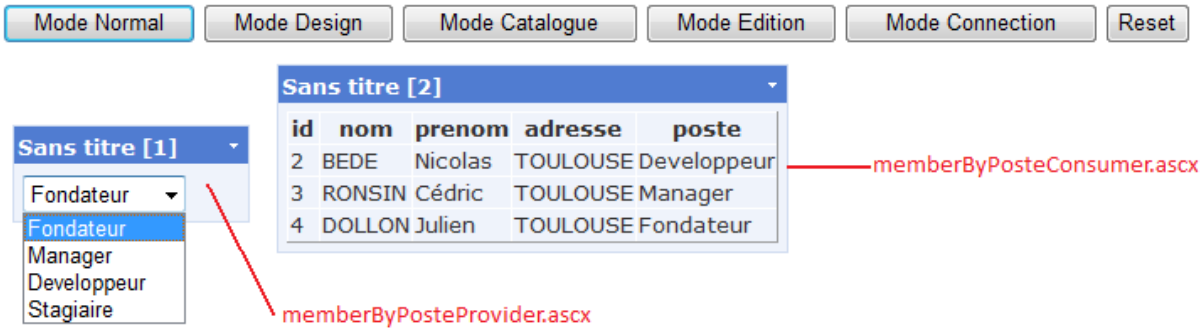
Le premier se nommera *MemberByPosteProvider.ascx* et contiendra une *DropDownList* qui sera liée à notre base de données (*bind*) sur la table *postes*, *value* et *texte* contiendrons tous deux l'intitulé du poste.

Le second se nommera *MemberByPosteConsumer.ascx* contiendra une *GridView* qui sera liée dynamiquement (dans le Page_Load du CodeBehind) à notre table *members*.

La liaison d'un contrôle à une base de données ayant été vu dans les chapitres précédents, nous ne reviendront pas dessus.

Puis, mettez chacun de ces deux *WebUserControl* dans une des *WebPartZone* que nous avons crées auparavant.

Voici un aperçu de ce que vous devriez obtenir :



Comme vous pouvez le constater nous avons prévu le boutton pour activer le mode *ConnectDysplayMode* et l'activation de ce mode ne change pas des autres :

C#

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

VB.NET

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

Cependant à ce stade, même si vous activez le mode *ConnectDisplayMode* l'option *Connecter* n'apparait pas dans la liste des possibilités. En effet, pour qu'un consommateur et un fournisseur

puissent dialoguer, il faut avant tout qu'il passent un contrat. Ce contrat nous allons le matérialiser par une interface que nous allons créer.

L'interface est une notion de programmation objet que vous avez du voir en étudiant le C#, il ne faut pas la confondre avec l'interface graphique !

Création de l'interface

Pour créer une interface faites clic droit sur votre solution, ajouter un nouvel élément, et choisissez Interface. Par convention, tous les noms d'interface commencent par un I, nous allons donc l'appeler *IPosteMemberContract*.

Le but de notre manœuvre va être de permettre une recherche de nos membres en fonction du poste qu'ils occupent. Pour cela nous allons ajouter à notre interface le prototype d'une méthode adéquate :

C#

```
public interface IPosteMemberContract
{
    string GetPoste();
}
```

VB.NET

```
Public Interface IPosteMemberContract
    Function GetPoste() As String
End Interface
```

Ne pas oublier de mettre `public` sinon vous ne pourrez pas accéder à votre interface (`private` par défaut).

Nous allons maintenant voir comment implémenter cette interface de façon à ce que nous puissions faire dialoguer nos contrôle.

Implémenter l'interface dans nos contrôles

Voyons tout d'abords avec notre fournisseur :

C#

```
public partial class MemberByPosteProvider : System.Web.UI.UserControl,
IPosteMemberContract
    // Ici nous implémentons notre interface, nous aurons donc à redéfinir toutes
les méthodes
    // qu'elle contient
    {
        [ConnectionProvider("Poste", "PosteProvider")]
        // Ce champs nous permet d'identifier notre contrôle
        // en tant que fournisseur
        public IPosteMemberContract GetPosteInterface()
        // Cette méthode retourne un objet qui a implémenté
        // notre interface
        {
            return (IPosteMemberContract)this;
            // -> Notre méthode retourne notre objet entier
            //    cela dit nous n'auront pas à nous préocuper
            //    de la gestion de ce retour, le mode
            //    ConnectDisplayMode le fera pour nous grace au
            //    ConnectionProvider juste avant notre méthode
        }
        public string GetPoste()
        //Voici la méthode de l'interface que nous devons définir
        {
            if (DropDownList1.SelectedIndex >= 0)
            //Si nous ne sortons pas de l'index de notre DropDownList
            {
                return DropDownList1.Items[DropDownList1.SelectedIndex].Value;
                // -> On retourne la valeur de l'index selectionné
            }
            else
            {
                return "";
                // -> Sinon on renvoit une chaine vide.
            }
        }
    }
```

VB.NET

```
Public Partial Class MemberByPosteProvider
    Inherits System.Web.UI.UserControl
    Implements IPosteMemberContract
    ' Ici nous implémentons notre interface, nous aurons donc à redéfinir toutes les
méthodes
    ' qu'elle contient

    ' Ce champs nous permet d'identifier notre contrôle
    ' en tant que fournisseur
    <ConnectionProvider("Poste", "PosteProvider")> _
    Function GetPosteInterface() As IPosteMemberContract
        ' Cette méthode retourne un objet qui a implémenté
        ' notre interface
        Return CType(Me, IPosteMemberContract)
        ' -> Notre méthode retourne notre objet entier
        '    cela dit nous n'auront pas à nous préocuper
        '    de la gestion de ce retour, le mode
        '    ConnectDisplayMode le fera pour nous grace au
        '    ConnectionProvider juste avant notre méthode
    End Function

    Public Function GetPoste() As String Implements IPosteMemberContract.GetPoste
        'Voici la méthode de l'interface que nous devons définir
        If DropDownList1.SelectedIndex >= 0 Then
            'Si nous ne sortons pas de l'index de notre DropDownList
            Return DropDownList1.Items(DropDownList1.SelectedIndex).Value
            ' -> On retourne la valeur de l'index selectionné
        Else
            Return ""
            ' -> Sinon on renvoit une chaine vide.
        End If
    End Function
End Class
```

Puis avec notre consommateur :

C#

```
public partial class MemberByPosteConsumer : System.Web.UI.UserControl
// Nottons que nous n'avons pas implémenter l'interface dans notre contrôle
{
    protected void Page_Load(object sender, EventArgs e)
    // Comme dit précédement nous lions notre GridView à
    // notre base de données de façon dynamique
    {
        string sqlConnectLoad = @"Data Source=CHOUPI2\SQLEXPRESS;
                                  Initial Catalog=dbTest;Integrated
Security=True;Pooling=False";
        string requeteLoad = "Select * from members ";
        SqlDataSource myConnectionLoad= new SqlDataSource(sqlConnectLoad,
requeteLoad);
        GridView1.DataSource = myConnectionLoad;
        GridView1.DataBind();
    }

    [ConnectionConsumer("Poste", "GetPoste")]
    // Ce champs défini notre contrôle comme étant un consommateur
    // la méthode qui suivra permettra la communication avec le fournisseur
    public void GetPosteInterface(IPosteMemberContract provider)
    // Cette méthode prend en paramêtre objet ayant implémenté notre interface,
    // or nous avons vu que dans notre fournisseur, une méthode retourné exactement
    // ce type d'objet, le mode ConnectDysplayMode fera le liens tout seul entre nos
    // deux contrôles grace à cela
    {
        string poste = "";
        if (provider.GetPoste() != "")
        {
            poste = "WHERE poste='" + provider.GetPoste() + "'";
            // -> provider.GetPoste() nous permet de récupérer le poste
            //    envoyé pas le fournisseur
        }
        string sqlConnect = @"Data Source=CHOUPI2\SQLEXPRESS;Initial Catalog=dbTest;
                              Integrated Security=True;Pooling=False";
        string requete = "Select * from members " + poste;
        SqlDataSource myConnection = new SqlDataSource(sqlConnect, requete);
        GridView1.DataSource = myConnection;
        GridView1.DataBind();
    }
}
```

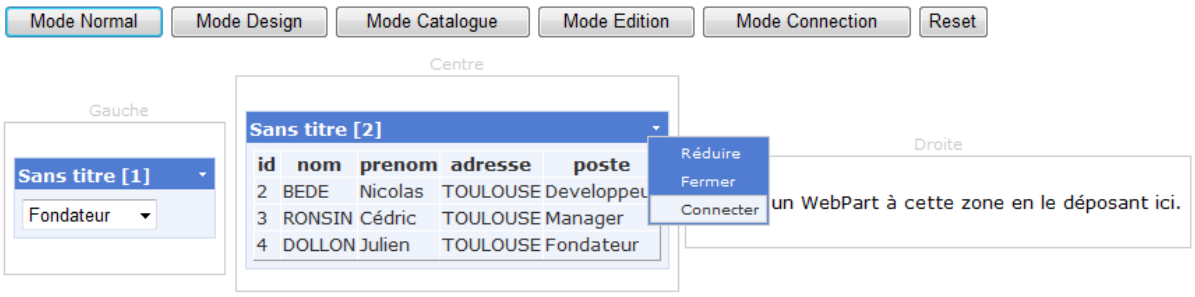
VB.NET

```
Public Partial Class MemberByPosteConsumer1
    Inherits System.Web.UI.UserControl
    ' Nottons que nous n'avons pas implémenter l'interface dans notre contrôle

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
        ' Comme dit précédement nous lions notre GridView à
        ' notre base de données de façon dynamique
        Dim sqlConnectLoad As String = "Data Source=CHOUPI2\SQLEXPRESS;Initial
Catalog=dbTest;Integrated Security=True;Pooling=False"
        Dim requeteLoad As String = "Select * from members "
        Dim myConnectionLoad As SqlDataSource = New SqlDataSource(sqlConnectLoad, requeteLoad)
        GridView1.DataSource = myConnectionLoad
        GridView1.DataBind()
    End Sub

    ' Ce champs défini notre contrôle comme étant un consommateur
    ' la méthode qui suivra permettra la communication avec le fournisseur
    <ConnectionConsumer("Poste", "GetPoste")> _
    Public Sub GetPosteInterface(ByVal provider As IPosteMemberContract)
        ' Cette méthode prend en paramètre objet ayant implémenté notre interface,
        ' or nous avons vu que dans notre fournisseur, une méthode retourné exactement
        ' ce type d'objet, le mode ConnectDysplayMode fera le liens tout seul entre nos
        ' deux contrôles grace à cela
        Dim poste As String = ""

        If Provider.GetPoste() <> "" Then
            poste = "WHERE poste='" + provider.GetPoste() + "'"
            ' -> provider.GetPoste() nous permet de récupérer le poste
            '    envoyé pas le fournisseur
        End If

        Dim sqlConnect As String = "Data Source=CHOUPI2\SQLEXPRESS;Initial
Catalog=dbTest;Integrated Security=True;Pooling=False"
        Dim requete As String = "Select * from members " + poste
        Dim myConnection As SqlDataSource = New SqlDataSource(sqlConnect, requete)
        GridView1.DataSource = myConnection
        GridView1.DataBind()
    End Sub
End Class
```

Testons à présent

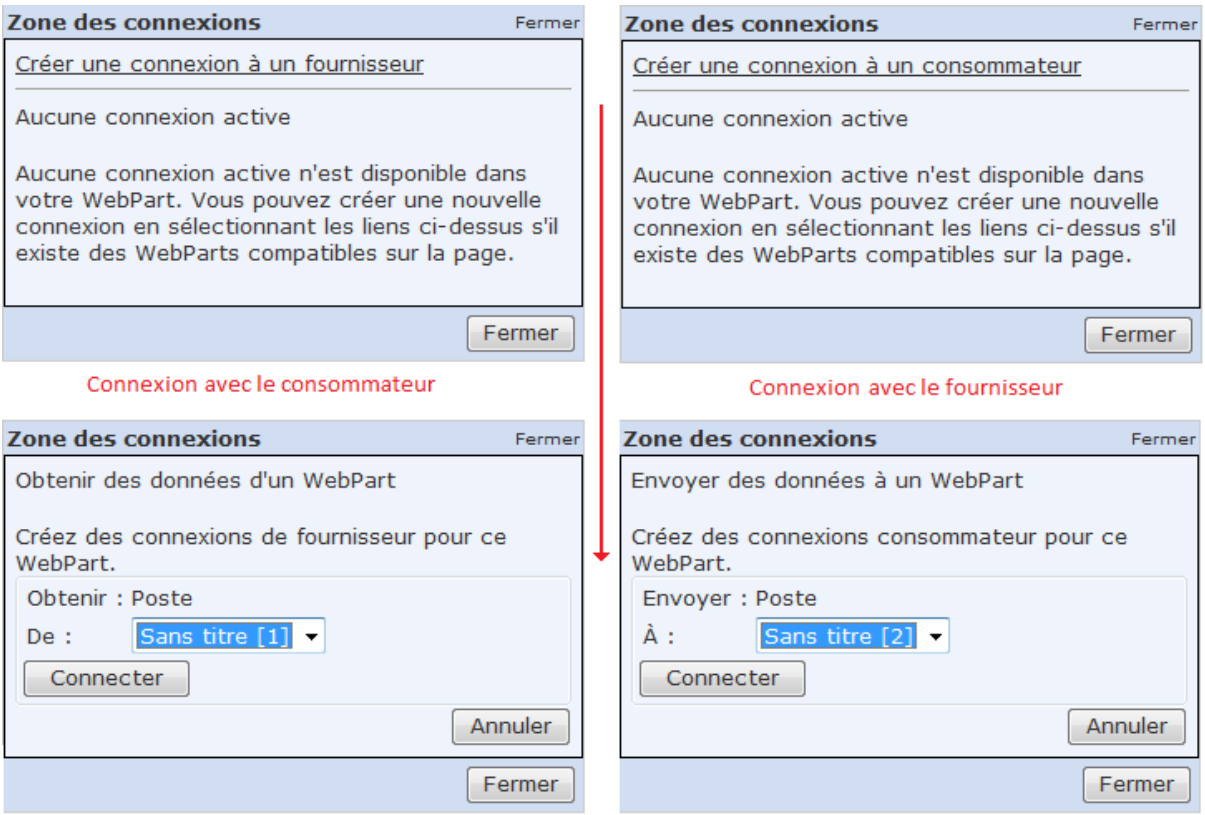
Il ne reste plus qu'à vérifier que nos contrôles fonctionnent.

Si tout s'est bien passé, lorsque vous activez le mode *ConnectDisplayMode* via votre *Button* l'option *Connecter* devrait apparaitre sur vos deux *WebUserControl*.



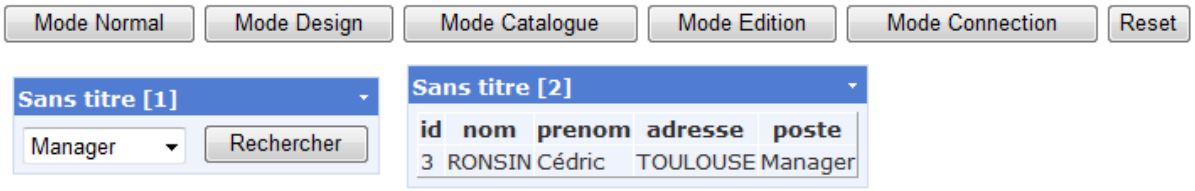
En fonction du contrôle que vous choisirez, il sera identifié soit en tant que fournisseur, soit en tant que consommateur conformément à ce que vous avez défini dans votre code.

Il vous sera alors demandé de choisir un fournisseur ou un consommateur avec lequel vous désirez créer une connexion.



Une fois que vous avez cliquez sur *Connecter* vos contrôles sont enfin lié. Il est possible de les déconnecter à tout moment. Mais désormais la recherche de membres sur la table *members* est filtrée grâce à la modification de la requête SQL que nous avons déterminé dans le *CodeBehind* de notre consommateur.

Afin d'effectuer plusieurs test, ajoutez un *Button* à votre contrôle *MemberByPosteProvider.ascx*. Inutile d'y ajouter un évènement, par défaut le *Button* effectue un *PostBack* et crée des *ViewState* de notre page courante et nous permettra de constater la mise à jour de notre contrôle consommateur *MemberByPosteConsumer.ascx*.



6 Conclusion

Comme nous avons pu le voir, ASP.NET permet de gérer facilement l'affichage à travers différentes méthodes que sont les thèmes, les pages maîtres, les profils utilisateurs et les Web parts. Ces techniques d'affichage et d'organisation sont des méthodes relativement avancées qui nécessite un bonne pratique avant de pouvoir prétendre à leurs maitrises. A la fin de la lecture de ce chapitre n'hésiter pas à effectuer vos propres recherches pour approfondir vos connaissances dans ce domaine.