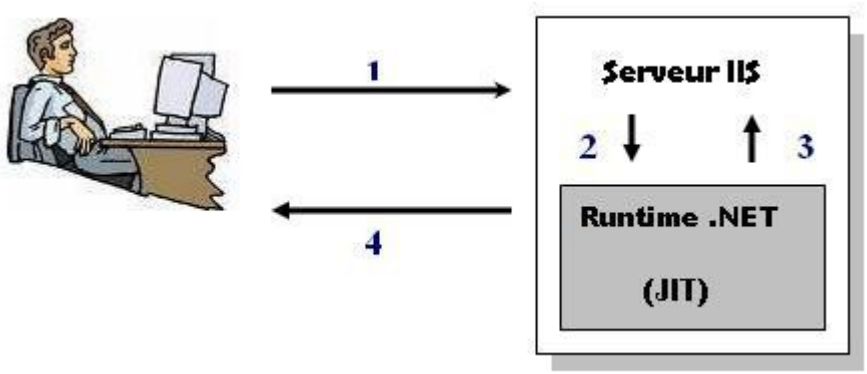
Sommaire

1. Introduction ASP.NET 2.0

1.1. *Principes*

L'interaction Client / Serveur est la base principale des applications web. Il est donc très important de bien comprendre le principe de fonctionnement d'ASP.NET dans l'environnement DotNet avec le serveur IIS.
Un petit schéma très simplifié vous aidera peut être à y voir plus clair :



Voici donc ce qui se passe lorsque vous, utilisateur désirant naviguer sur une page web, générez comme action si l'application que vous désirez atteindre est développée en ASP.NET 2.0 :

- 1 = vous tapez une url dans votre navigateur et donc, envoyez une requête pour une page aspx d'un client web vers le serveur IIS
- 2 = la requête est analysée et le traitement est transféré au runtime, un processus est créé pour exécuter l'application
- --> S'il s'agit de la première exécution du code de cette page, le compilateur JIT (Just In Time) compile le code en binaire natif et le stoque en mémoire.
- --> Si ce n'est pas la première exécution, le code binaire est chargé depuis le cache.
- 3 = ce code binaire est exécuté puis renvoyé vers le serveur IIS
- 4 = IIS renvoie la réponse sous la forme de code HTML strict vers l'utilisateur. Ce code HTML est affiché dans votre navigateur.

1.2. *Environnement de développement*

1.2.1. Un serveur Web

Puisque nous allons créer du code utilisant une liaison Client / Serveur, il est bien entendu nécessaire d'avoir un serveur à disposition dans notre cas, Nous allons utiliser le **serveur IIS**.
IIS est disponible avec windows XP professionnel et windows 2003 server Sous XP Home, il n'est pas aisé d'installer IIS, bien que cela soit possible.

1.2.2. Framework 2.0

Si framework .NET n'a pas été installé **après** le serveur IIS, vous aurez des problèmes d'exécution des pages aspx.
Pour remédier à cet inconvénient à postériori, vous pouvez exécuter une commande du type :
C:\Windows\Microsoft.Net\Framework\v2.0.xx\aspnet_regiis.exe -i ou xx est la version du Framework 2.0 présente sur votre ordinateur.

1.2.3. Un EDI, c'est nécessaire ?

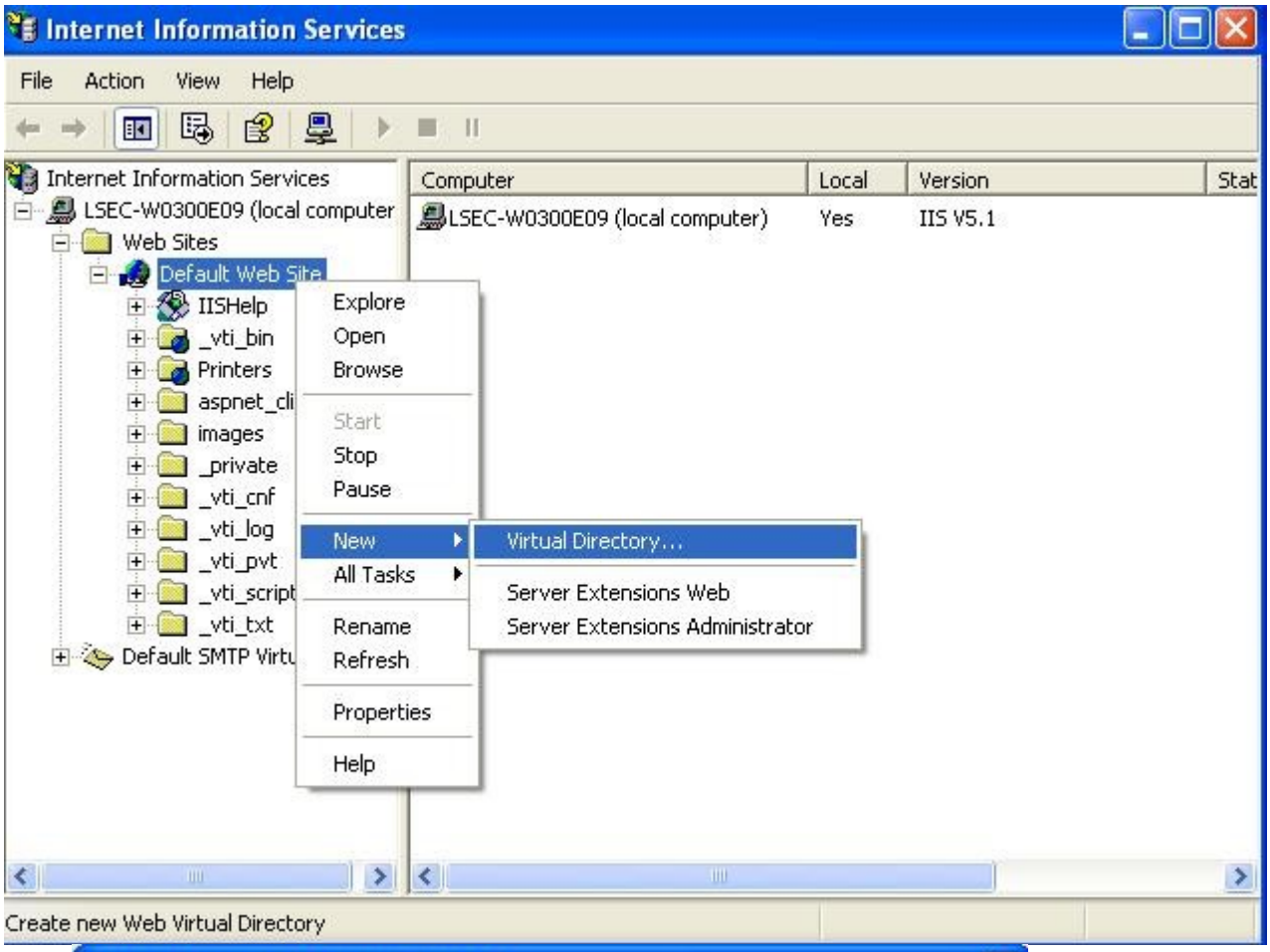
Nous avons tous l'habitude de travailler dans un environnement de développement intégré bien que cela ne soit pas toujours nécessaire mais plutôt bien pratique. Il en est de même avec le développement ASP.NET. Vous pouvez, comme pour des applications Winforms, écrire du code dans un éditeur de texte. Voici, en quelques étapes, la réalisation et l'exécution d'une page aspx créée avec le bloc-note :

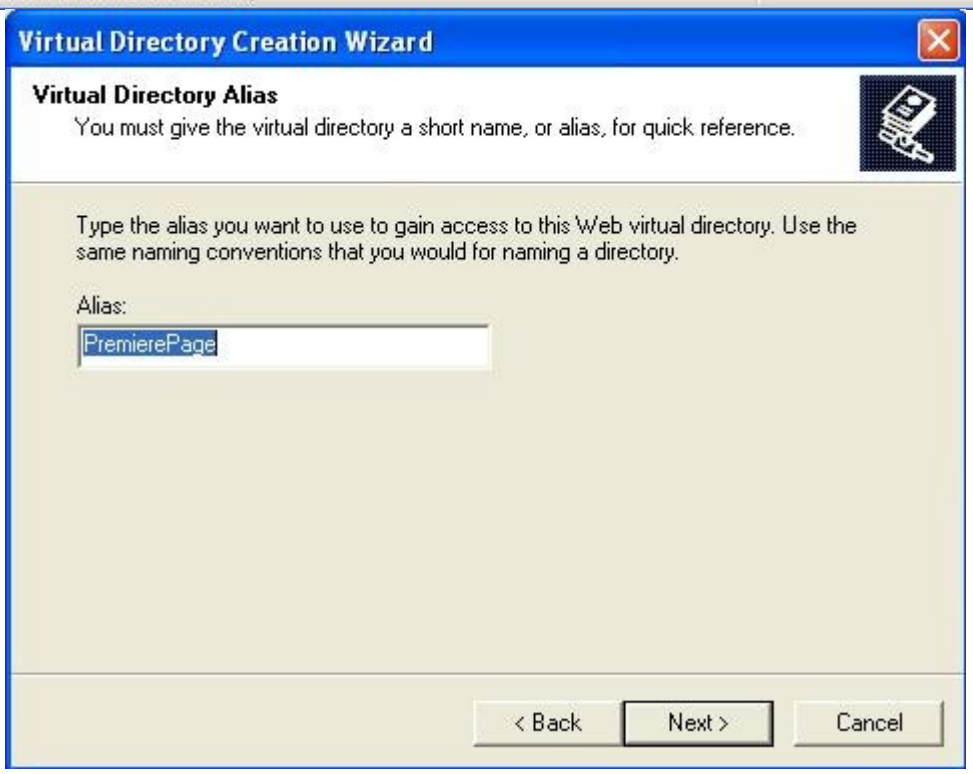
1.2.3.1. Etape 1

Créez un site virtuel sur votre IIS et nommez-le, par exemple, *"PremierePage"*. Si vous n'avez jamais réalisé cette opération, voici comment procéder :

a. Allez dans le panneau de contrôle de Services Internet (IIS) : Outils d'administration dans le panneau de configuration puis choisissez Internet Informations Services
b. Déroulez les options jusqu'à trouver Site Web par défaut et faites un clic droit
c. Choisissez Nouveau -> Répertoire virtuel ...
d. Créez votre répertoire

Voici en images et sur XP Pro en anglais les étapes décrites ci-dessus :


Internet Information Services
File Action View Help
Internet Information Services
LSEC-W0300E09 (local computer
Web Sites
Default Web Site
IISHelp
_vti_bin
Printers
aspnet_cli
images
_private
_vti_cnf
_vti_log
_vti_pvt
_vti_script
_vti_txt
Default SMTP Virtu
Computer
Local
Version
Stat
LSEC-W0300E09 (local computer)
Yes
IIS V5.1
Explore
Open
Browse
Start
Stop
Pause
New
All Tasks
Rename
Refresh
Properties
Help
Virtual Directory...
Server Extensions Web
Server Extensions Administrator
Create new Web Virtual Directory


Virtual Directory Creation Wizard
Virtual Directory Alias
You must give the virtual directory a short name, or alias, for quick reference.
Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.
Alias:
PremierePage
< Back
Next >
Cancel

Terminer votre création en laissant les paramètres par défaut.

1.2.3.2. Etape 2

Ouvrez le bloc-notes et créez un fichier avec ce code :

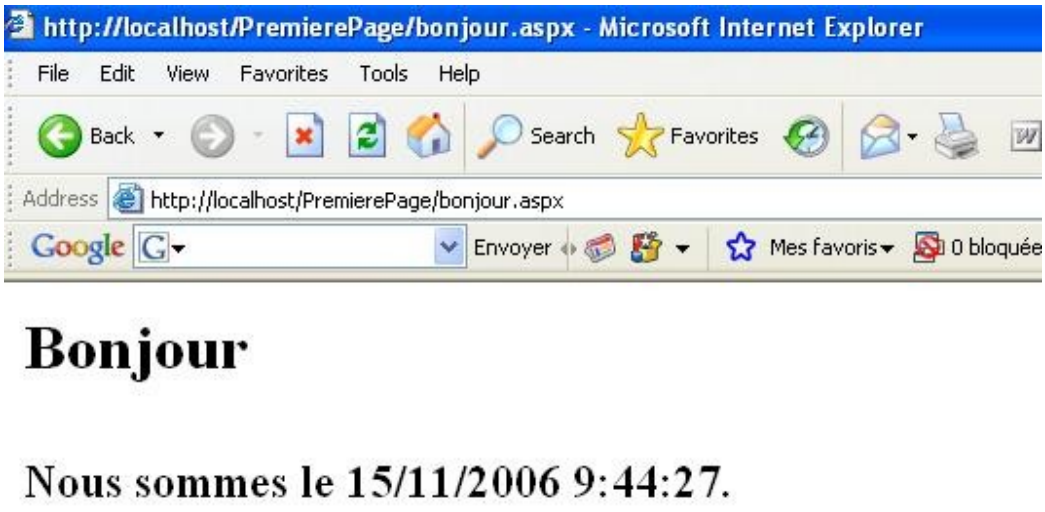
```
<%@ Page Language="VB" %>
<html>
        <body>
                <h1>Bonjour</h1>
                <br />
                <h2>Nous sommes le <%= DateTime.Now.ToString()
%>.</h2>
</body>
</html>
```

Sauvegardez-le à la racine du site que vous avez créé en le nommant par exemple *"bonjour.aspx"*.

1.2.3.3. Etape 3

Exécutez cette page aspx dans votre navigateur en tapant son adresse dans la barre de navigation :
http://localhost/PremierePage/bonjour.aspx

et vous devez avoir une page web comme suit :

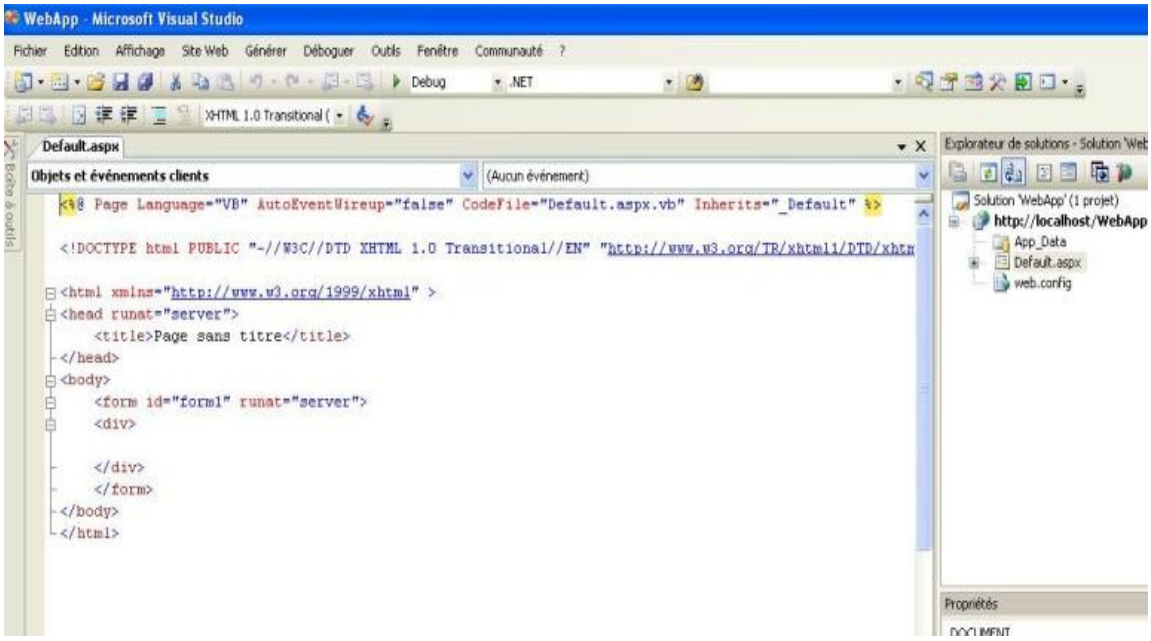


Vous venez donc de créer votre première page ASP.NET s'exécutant sur un serveur IIS sans avoir ouvert Visual Studio comme support de programmation.

1.3. *La gestion d'Etat*

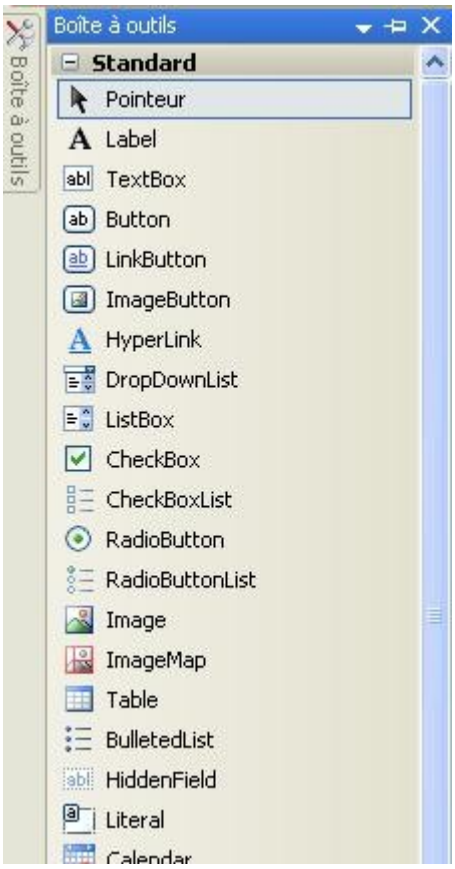
1.3.1. Première page

Enfin ! Un petit exemple en utilisant Visual Studio ou Visual Web Développer pour se familiariser avec l'environnement ASP.NET. Si vous êtes familier avec le "designer" des applications Visual Studio ou Visual Express Edition, ceci vous paraîtra très simple mais on se permet tout de même de détailler un peu l'interface pour ceux qui abordent ce thème pour la première fois.
Reprenons notre EDI et, après avoir bien configuré les données au niveau du serveur et du nom de l'application, l'interface doit maintenant ressembler à ceci :



Vous pouvez remarquer que l'interface des applications ASP.NET diffère des applications Winforms mais nous y retrouvons quand même pas mal de points communs, notamment :

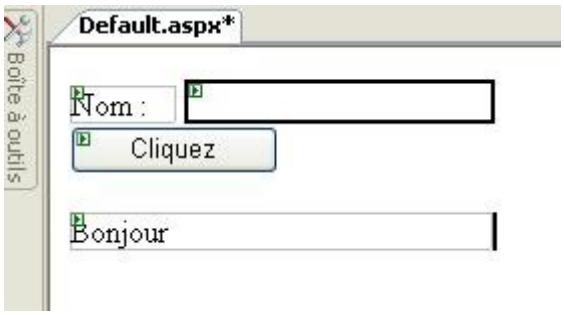
l'explorateur de solution contenant notre projet "WebApp", sa localisation "http://localhost/WebApp" et la page par défaut "Default.aspx", que nous pouvons bien évidemment renommer.
les propriétés des contrôles et pages grâce auxquelles nous allons pouvoir définir des comportements graphiques ou autres.
la page de code où une partie de codage est générée automatiquement par l'environnement de développement.
deux boutons "design" et "source" nous permettant de passer aisément d'un mode à l'autre dans notre page aspx. Remarquez aussi que, si vous déplacez votre curseur dans la partie code, à droite du bouton "source", vous apercevez l'endroit exact où se situe le curseur dans l'arborescence des balises HTML.
la boite à outils, ancrée ou non, contenant les contrôles utilisables pour votre application web :



Passons maintenant en mode "design". Faites glisser sur la page les contrôles suivant et changez leurs propriétés en suivant le tableau ci-après :

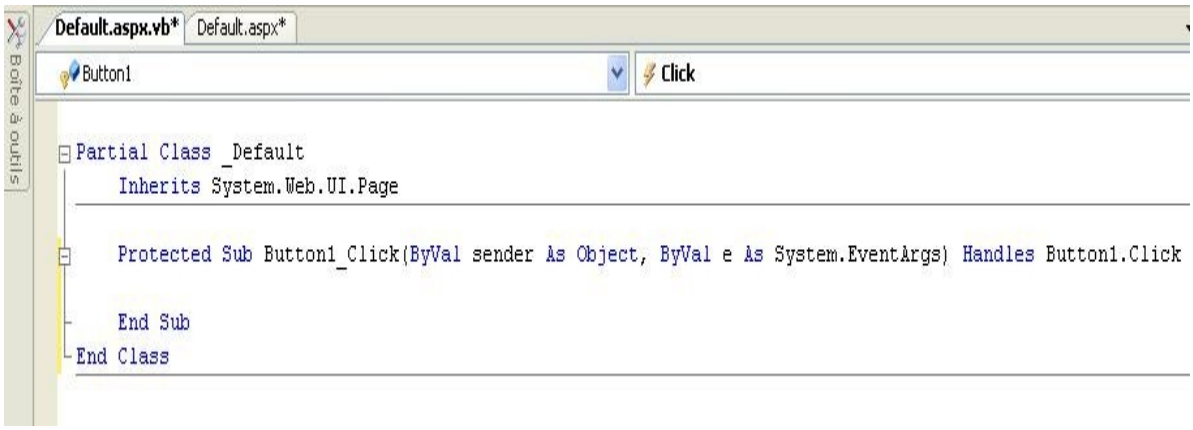
Contrôle	Propriété	Contenu
Un "label" : Label1	Text	"Nom :"
Un "textbox" à droite de Label1 : TextBox1	BorderWidth	2
Un "button" sous Label1 : Button1	Text	"Cliquez"
Un "label" sous le bouton : Label2	Text	"Bonjour"

Remarque : dans la propriété BorderWidth, par défaut, l'unité de mesure est en "px" (pixel). Cela correspond bien aux normes HTML.
Votre page doit ressembler à ceci :



Si vous retournez en mode "source", vous constatez que le code HTML s'est enrichi automatiquement des contrôles que vous avez intégrés à votre page ainsi que des propriétés modifiées via la page de propriétés. Rien ne vous empêche, au fil de l'expérience acquise dans le développement ASP.NET, de taper immédiatement le code de vos contrôles dans la page HTML, vous verrez que le "design" se met aussi à jour de la même manière. L'avantage de coder directement dans l'HTML se trouve dans le libre choix que vous avez du type de contrôle placé. Par exemple, vous voyez dans notre application que le TextBox1 est considéré comme un "asp:textbox" ce qui, niveau exécution du code prend plus de place et de temps qu'un simple "asp:inputbox" alors que le résultat, ici, est exactement le même. Pour les utilisateurs avertis ayant déjà réalisé des sites web en HTML, il peut aussi être plus aisé de coder directement dans la page source.

A ce point, nous avons des contrôles placés sur une page aspx, mais encore aucune action n'est définie. Vous avez beau taper un nom dans "TextBox1" et cliquer sur le "Button1", rien ne se passe. En effet, il faut associer un événement au bouton "Cliquez". Pour ce faire, double-cliquez sur le bouton en mode design et l'environnement de développement va créer une méthode associée à l'événement "Click" du bouton :



Remarquez qu'une nouvelle page est apparue "Default.aspx.vb" qui contient le code associé aux méthodes et événements. Dans votre événement "Button1_Click", tapez cette ligne :

```
label2.text=label2.text & " " & textbox1.text
```

Vous verrez en cours de frappe que l'aide à la complétion existe aussi, exactement comme dans les applications winforms. Maintenant, vous pouvez exécuter votre page aspx (F5). Lors d'une première exécution vous allez certainement obtenir ce message :

Par défaut, le débogage n'est pas activé au niveau des applications web. On vous conseille fortement de l'activer en répondant OK avec la première option cochée.
Ensuite, testez en tapant un nom et en cliquant sur votre bouton. Prenons un peu le temps de voir le code HTML de la page exécutée :

```
<!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><title>
        Page sans titre
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
      value="/wEPDwUJODUwMjI0NzE3D2QWAgIDD2QWAgIHDw8WAh4EVGV4dAUNQm9uam91ciBLaGFueWRkZLdHzDIpDIJkqx1ADxwXHexql11a" />
</div>

    <div>
        <span id="Label1" style="display:inline-block;width:53px;">Nom :</span>
        <input name="TextBox1" type="text" value="khany" id="TextBox1" style="border-width:2px;border-style:solid;" />
        <br />
        <input type="submit" name="Button1" value="Cliquez" id="Button1" style="width:104px;" /><br />
        <br />
        <span id="Label2" style="display:inline-block;width:211px;">Bonjour Khany</span></div>

<div>

        <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
      value="/wEWAwKC2u62DALs0bLrBgKM54rGBtivPH2tbxgafhki+swEj4o+245c" />
</div></form>
</body>
</html>
```

Vous constatez que des champs cachés ont été générés. Le champ nommé **_VIEWSTATE** contient toutes les informations d'état des contrôles de la page. Il est intéressant car est accessible par le programmeur et peut contenir des données internes aux pages. Cette notion de conservation de données sera développée plus loin dans ce tutoriel.

1.3.2. Des événements particuliers

En tant que programmeur, nous sommes pratiquement tous habitués à entendre parler d'événements, nous venons encore de nous en servir

dans notre première petite application. Nous connaissons ceux liés à une action comme "click", répondant à une action sur la souris. ASP.NET possède le même genre d'événements mais, certains sont assez particuliers et très importants pour le déroulement et le contrôle de ce genre d'application.

1.3.2.1. Application

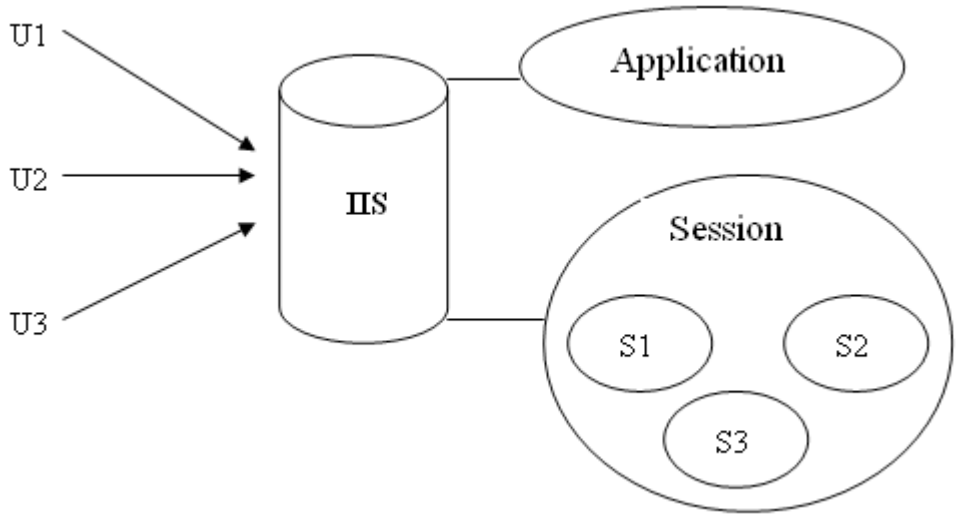
Evénement	Description
Application_Start	Exécuté lors du premier appel à une page du site depuis le démarrage de IIS
Application_End	Appelé lorsque l'application se termine, cela ne signifie pas que IIS s'arrête mais est d'office appelé si, pour une raison quelconque IIS est arrêté

1.3.2.2. Session

Evénement	Description
Session_Start	appelé lors de chaque nouvelle session d'un navigateur client
Session_End	fin de session : lors d'un timeout ou lors d'une destruction explicite (Session.Abandon()) via un lien "Log Out" par exemple

Il faut aussi savoir qu'une session peut stocker ses données en mode "InProc" (dans le process en mémoire) ou en mode "Sql..." (dans une BD SqlServer) via la base de données "AspNetState". Application et Session sont des notions très importantes en ASP.NET. Elles jouent en effet un rôle très actif au niveau de la vie d'un site et, notamment, au niveau de la pérénité des données véhiculées dans le site lui-même.

Un petit schéma pour mieux visualiser la différence entre "Application" et "Session" :



Soit trois utilisateurs U1, U2 et U3 qui envoient une requête vers le serveur IIS. Il y aura un seul objet "Application" commun à tous les

utilisateurs du site mais trois objets "Session" correspondant chacun à un utilisateur précis.
Si U2 quitte son poste de travail sans couper son navigateur :

- s'il n'y a pas de timeout, les autres utilisateurs peuvent accéder à S2
- S'il y a timeout et que U2 revient visiter le site, une nouvelle session S4 sera créée .

Par contre, si U2 coupe son navigateur, S2, persiste jusqu'à un éventuel timeout ou jusqu'à la fin de l'application

1.3.2.3. PostBack

Cet événement génère un appel au serveur. Dans ASP.NET 2.0, la page se rappelle continuellement en déclenchant cet événement. C'est au programmeur de créer les conditions de passage d'une page à l'autre.

- **IsPostBack** est une propriété de la page booléenne (read-only) qui permet justement d'effectuer ce genre de test.
  Par exemple, on l'utilise dans l'événement Page_Load pour éviter de recharger des données persistantes.
- **AutoPostBack** est une propriété des contrôles qui active le déclenchement d'un aller retour sur le serveur.

Surtout, ne pas s'affoler, ces notions seront reprises maintes fois dans le développement du tutoriel mais sont nécessaires niveau vocabulaire pour évoluer dans la manipulation du code proprement dit.

1.3.3. Les Server Controls

Un petit mot sur les types de contrôles présents dans ASP.NET. Il existe deux jeux de contrôles s'exécutant côté serveur :
Les **Web Controls**, gérés par des événements, ils ressemblent plus aux objets utilisés dans du développement winforms c'est-à-dire qu'ils possèdent des propriétés ("font", "backcolor", ...) facilitant la mise en forme. Ils dépendent de "***System.Web.UI.WebControls***".
Les **HTML Controls** qui correspondent directement aux balises HTML. Les attributs des balises correspondantes sont accessibles via les propriétés de ces contrôles. Pour faire une analogie avec les "WebControls", ceux-ci ne possèdent qu'une balise "Style" pour la mise en forme, cela est plutôt limitatif.
Ces derniers dépendent eux de "***System.Web.UI.HtmlControls***".

1.3.4. ViewState

Comme dit lors du premier exemple de page aspx, le ViewState, nouveau concept introduit par Microsoft avec ASP.NET, représente l'état de l'ensemble des contrôles d'un page. Les informations sont sauvées sous forme d'un flux sérialisé dans la page HTML et le champ caché **_VIEWSTATE** permet le transit de ces informations entre le client et le serveur.
Il peut être désactivé au niveau d'un contrôle, au niveau d'une page ou au niveau d'une application en plaçant la propriété *EnabledViewState* à *False*.

Le plus intéressant est que le programmeur peut y ajouter ses propres informations sous forme d'objets indexés par une clé de type *String*. Pour sauvegarder et lire une information, voici comment utiliser le ViewState, par exemple pour modifier un argument dans une requête de sélection :

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        If Not IsPostBack Then
                ViewState("tri") = "nom"
                ConstruireRequete()
        End If
End Sub

Private Sub ConstruireRequete
    Dim tri As String = CType(ViewState("tri"), String)
    Dim rq As String = "SELECT * FROM UneTable ORDER BY " & tri
End Sub

Protected Sub cmdLocalite_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles cmdLocalite.Click
        ViewState("tri") = "localite"
        ConstruireRequete()
End Sub
```

1.3.5. Cookies

Les cookies permettent aux applications Web de stocker des informations spécifiques à l'utilisateur. Par exemple, lorsqu'un utilisateur visite votre site, les cookies peuvent vous servir à stocker ses préférences, ou d'autres informations. Lorsque cet u tilisateur revient visiter votre site Web, l'application peut récupérer les informations stockées précédemment.

- **Exemple de Création de cookie**

```
Dim cookie As HttpCookie
Dim UserID As String
User = "neo"
cookie = New HttpCookie("User")
cookie.Values.Add("User", User)
Response.Cookies.Add(cookie)
```

- **Exemple de Lecture de cookie**

```
Dim cookie As HttpCookie
cookie = Request.Cookies("User")
Dim User As String
User= cookie.Value()
```

- **Détecter si le navigateur supporte les cookies**

```
Dim CoookiesSupported As Boolean = Request.Browser.Cookies
```

- **Supprimer un cookie**

Vous ne pouvez pas supprimer directement un cookie sur l'ordinateur d'un utilisateur. Mais vous pouvez donner au navigateur de l'utilisateur l'ordre de supprimer le cookie en réglant la date d'expiration de ce cookie sur une date révolue. La prochaine fois que l'utilisateur soumettra une demande à une page dans le domaine ou le chemin d'accès où se trouve le cookie, le navigateur jugera que le cookie a expiré et le supprimera.

```
myCookie.Expires = DateTime.Now.AddDays(1D)
```

1.3.6. Variable de session

"Session" est un objet qui s'utilise un peu comme le ViewState, c'est-à-dire avec une clé mais se comporte plutôt comme une table de hachage. Prenons deux pages aspx :

---

page1.aspx : page dans laquelle nous encodons, par l'intermédiaire d'une TextBox, un nom de société.

page2.aspx : page dans laquelle nous affichons le nom de la société (vous comprenez que le but est d'avoir une page d'affichage de données de société se trouvant par exemple dans une base de données)

```
Protected Sub cmdAfficheSoc (Byval sender As Object, ByVal e
As System.EventArgs) Handles cmdAfficheSoc.Click

       Session("NomSoc") = txtNomSoc.Text
       Response.Redirect("page2.aspx")

 End Sub
```

Code de la page1.aspx : L'utilisateur introduit un nom de société dans la TextBox nommée "txtNomSoc". Cette information est sauvée en Session avant de passer à la page2.aspx

```
Protected Sub Page_Load (Byval sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

  If Session("NomSoc") IsNot Nothing Then
       lblNomSoc.Text = CType(Session("NomSoc"), String)
  Else
       Response.Write("Aucune société n'a été choisie !")
  End If

End Sub
```

Code de la page2.aspx : Un test est effectué pour savoir si la variable de session contient bien une donnée. Celle-ci est affichée en passant par un transtypage.
Il est évident que cet exemple est très simpliste et que l'objet Session permet bien d'autres utilisations. Voici quelques points liés à l'objet Session (liste non exhaustive) :

- Initialisation de l'objet Session : événements Session_Start et Session_End déclenchés par le serveur et accessibles via le fichier Global.asax
- Expiration de la session
- Session avec ou sans cookies
- Session sécurisée

1.3.7. Variable d'application

La grande différence avec l'objet Session se situe dans le fait qu'un objet Application conserve des données pour **l'ensemble des utilisateurs** d'un même site web. Il s'utilise de la même manière que l'objet Session.

```
Protected Sub Page_Load (Byval sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
 Dim cpt As Integer = 0
 Application.Lock()
 If Application("Compteur") IsNot Nothing Then
        cpt = CType(Application("Compteur"), Integer)
 End If
 cpt = cpt + 1



 Application("Compteur") = cpt
 Application.UnLock()
 lblVisite.Text = "Page vue : " & cpt & " fois."
End Sub
```

L'objet Application étant commun à tous les utilisateurs du site, il est préférable de bloquer l'accès lors de l'écriture et, bien entendu, de ne pas oublier l'action inverse.

1.3.8. L'objet Cache

Comme l'objet Application, il conserve aussi des données accessibles à tous les utilisateurs mais il possède quelques avantages non négligeables:

- Gestion interne de locking
- Plus rapide
- Gestion des dépendances

En ce qui concerne les dépendances, on peut en citer quelques-unes très succinctement car ce genre de programmation demanderait presque un tutoriel à elle toute seule !

- Dépendances de temps : permet de faire expirer automatiquement une donnée à une date/heure absolue

- Dépendances fichiers : le serveur d'application peut mettre à jour des données lorsque cells-ci sont modifiées dans le fichier associé

- Dépendances SQL : sous SqlServer 2000 et 2005. Agit de la même manière avec une base de données grâce au "poling" (interrogation du serveur vers la BD).
- le callback : association d'une procédure qui est rappelée, non pas dès que la donnée est supprimée mais à la prochaine exécution de la page qui contient la procédure

1.3.9. Caching (ou cache HTML)

Un autre aspect de la mise en cache des données suivant diverses méthodes. Ici aussi, il serait trop long d'étendre leur mode d'utilisation.

- Cache de sortie (output cache) : prend un "copie" instantanée du flux HTML puis supplante toute action de requête en imposant sa "copie" gardée en cache
- substitution : ce contrôle permet de ne pas mettre en cache une partie de la page même si le cache est activé
- profils de cache : peuvent être créés dans le Web.Config et associé par leur nom aux pages qui en ont besoin
- fragments de cache : fonctionne comme le cache de sortie mais donne la possibilité au programmeur de ne mettre en cache qu'une partie de la page HTML. Le frament caching peut se faire grâce aux usercontrols qui disposent eux-mêmes d'une directive Output

1.3.10. QueryString

QueryString permet de faire passer des informations via l'URI d'une page à une autre.
En reprenant l'exemple d'un ID de société sélectionné dans une page dont les données sont présentées dans une autre page, on aurait très bien pu indiquer cet ID via l'URI lors de l'appel à la deuxième page.

Vous avez choisi la société ayant un ID = 1235, voici comment passer l'identifiant à la page suivante :

```
<A href="page2.aspx?idsoc=1235"></A>
```

Pour récupérer l'ID dans la seconde page, il vous suffira de coder comme suit :

```
<p>Vous avez choisi la société : &
Request.QueryString("idsoc")</p>
```

Vous comprenez maintenant le pourquoi de certaines url complexes du genre :
http://www.monsite.com/repertoire/liste.asp?
id=1257&lng=fr&action=del&email=abc@prov.fr

1.4. *Contrôles utilisateur ASP.NET*

Il peut arriver que vous ayez besoin dans un contrôle de fonctionnalités dont les contrôles serveur Web ASP.NET intégrés ne disposent pas. Vous pouvez alors créer vos propres contrôles. Pour ce faire, vous disposez de deux options : Vous pouvez créer :

- Des contrôles utilisateur. Les contrôles utilisateur sont des conteneurs dans lesquels vous pouvez placer des balises et des contrôles serveur Web. Vous pouvez ensuite traiter le contrôle utilisateur comme une unité et lui assigner des propriétés et des méthodes.
- Des contrôles personnalisés. Un contrôle personnalisé est une classe que vous écrivez et qui dérive de Control ou de WebControl.

Les contrôles utilisateur sont beaucoup plus faciles à créer que les contrôles personnalisés, dans la mesure où vous pouvez réutiliser des contrôles existants. Il est donc particulièrement facile de créer des contrôles comportant des éléments d'interface utilisateur complexes. Cette rubrique fournit une vue d'ensemble de l'utilisation des contrôles utilisateur ASP.NET.

1.4.1. Structure de contrôle utilisateur

Un contrôle Web ASP.NET ressemble à une page ASP.NET complète (fichier .aspx), avec à la fois une page d'interface utilisateur et du code. Un contrôle utilisateur se crée de façon très semblable à une page ASP.NET. On lui ajoute par la suite le balisage et les contrôles enfants nécessaires. Tout comme une page, un contrôle utilisateur peut inclure du code servant à manipuler son contenu, et notamment à effectuer des tâches telles que des liaisons de données.
Un contrôle utilisateur présente les différences suivantes par rapport à une page Web ASP.NET :

- L'extension du nom de fichier du contrôle utilisateur est .ascx.
- Au lieu d'une directive @ Page, le contrôle utilisateur contient une directive @ Control qui définit la configuration et d'autres propriétés.
- Les contrôles utilisateur ne peuvent pas s'exécuter comme des fichiers autonomes. Vous devez au lieu de cela les ajouter à des pages ASP.NET, comme vous le feriez pour n'importe quel contrôle.
- Le contrôle utilisateur ne contient pas d'élément **html body** ou **form**. Ces éléments doivent se trouver dans la page d'hébergement.

Vous pouvez utiliser sur un contrôle utilisateur les mêmes éléments HTML (sauf les éléments **html**, **body** ou **form**) et les mêmes contrôles Web que dans une page Web ASP.NET. Par exemple, si vous créez un contrôle utilisateur afin de l'utiliser comme barre d'outils, vous pouvez placer dessus une série de contrôles serveur Web Button et créer des gestionnaires d'événements pour les boutons.

L'exemple suivant montre un contrôle utilisateur qui implémente un contrôle Spinner dans lequel les utilisateurs peuvent cliquer à leur guise sur des boutons pour naviguer dans une série de choix au sein d'une zone de texte.

```
<%@ Control Language="VB"  ClassName="UserControl1" %>
<script runat="server">
   Protectedcolors As String() = {"Red", "Green", "Blue", "Yellow"}
   Protected currentColorIndex As Integer = 0
   Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs)
      If IsPostBack Then
         currentColorIndex = CInt(ViewState("currentColorIndex"))
      Else
         currentColorIndex = 0
         DisplayColor()
      End If
   End Sub

   Protected Sub DisplayColor()
      textColor.Text = colors(currentColorIndex)
      ViewState("currentColorIndex") = currentColorIndex.ToString()
   End Sub

   Protected Sub buttonUp_Click(ByVal sender As Object, _
         ByVal e As System.EventArgs)
      If currentColorIndex = 0 Then
         currentColorIndex = colors.Length - 1
      Else
         currentColorIndex -= 1
      End If
      DisplayColor()
   End Sub

   Protected Sub buttonDown_Click(ByVal sender As Object, _
         ByVal e As System.EventArgs)
      If currentColorIndex = colors.Length - 1 Then
         currentColorIndex = 0
      Else
         currentColorIndex += 1
      End If
      DisplayColor()
   End Sub
</script>
```

```
<asp:TextBox ID="textColor" runat="server"
  ReadOnly="True" />
<asp:Button Font-Bold="True" ID="buttonUp" runat="server"
  Text="^" OnClick="buttonUp_Click" />
<asp:Button Font-Bold="True" ID="buttonDown" runat="server"
  Text="v" OnClick="buttonDown_Click" />
```

1.4.2. Ajout d'un contrôle utilisateur à une page

Pour utiliser un contrôle utilisateur, vous devez l'inclure dans une page Web ASP.NET. Lorsqu'une demande est soumise concernant une page et que cette page contient un contrôle utilisateur, celui-ci passe par toutes les étapes du traitement qu'effectuent tous les contrôles serveur ASP.NET.

Pour insérer un contrôle utilisateur dans une page Web Forms

1. Dans la page Web ASP.NET conteneur, créez une directive @ Register comprenant :
   - Un attribut **TagPrefix**, qui associe un préfixe au contrôle utilisateur. Ce préfixe sera inclus dans la balise d'ouverture de l'élément du contrôle utilisateur.
   - Un attribut **TagName**, qui associe un nom au contrôle utilisateur. Ce nom sera inclus dans la balise d'ouverture de l'élément du contrôle utilisateur.
   - Un attribut **Src**, qui définit le chemin d'accès virtuel au fichier contrôle utilisateur que vous incluez.
2. Dans le corps de la page Web, déclarez l'élément contrôle utilisateur à l'intérieur de l'élément **form**.
3. Éventuellement, si le contrôle utilisateur expose des propriétés publiques, définissez-les de façon déclarative.

1.5. *Validation des données*

La validation des données est en général la chose la plus importante dans un site web. Ici, nous allons pouvoir travailler côté client et côté serveur, c'est indispensable pour prévenir au plus tôt l'utilisateur d'une erreur éventuelle. En effet, il est inutile d'envoyer une demande au serveur si l'information transmise est erronée : cela génère une perte de temps et un encombrement inutile du serveur.

La validation côté client est donc celle qui intervient la première et se fait en général en JavaScript. ASP.NET fournit des contrôles de validation qui génèrent le code javascript associé, vous évitant de connaître à fond le langage et de devoir taper le code.

Les principaux contrôles de validation sont :

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

Voyons un peu les caractéristiques générales de chacun.

1.5.1. RequiredFieldValidator

Le plus fréquemment utilisé car il est le seul qui peut s'assurer qu'un champ n'est pas vide. En effet, tous les autres contrôles de validation acceptent un champ vide donc, associer ce contrôle de validation aux autres contrôles permet cette vérification essentielle.
Le *RequiredFieldValidator* a donc pour fonction de vérifier qu'un champ a été modifié. Ses propriétés principales à renseigner sont :

Nom de la propriété	Utilisation
ControlToValidate	doit contenir le nom du contrôle à valider
ErrorMessage	message à afficher en cas d'erreur dans le contrôle *ValidationSummary*
InitialValue	contient une valeur qui invalide le contrôle si celui-ci est égal à cette valeur précise
Text	texte affiché en cas de non validation

Exemple de RequiredFieldValidator sur une TextBox nommée TxtNom :

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" ControlToValidate="TxtNom" ErrorMessage="Admin
n'est pas un nom valide" SetFocusOnError="True"
InitialValue="Admin">
</asp:RequiredFieldValidator>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2"
runat="server" ControlToValidate="TxtNom" ErrorMessage="Le
champ nom est obligatoire" / >
```

Vous remarquez que pour valider le nom qui est obligatoire, il nous faut 2 contrôles *RequiredFieldValidator*.
Un pour signaler que le nom ne peut pas être un champ vide, l'autre pour interdire l'utilisation du nom "Admin".

1.5.2. RangeValidator

Comme son nom l'indique, il sera utilisé pour valider l'encodage entre des bornes données. Par exemple, encoder un nombre entre 1 et 10.
Les propriétés sont pratiquemment identiques à celles du contrôle précédent :

Nom de la propriété	Utilisation
ControlToValidate	doit contenir le nom du contrôle à valider
ErrorMessage	message à afficher en cas d'erreur dans le contrôle *ValidationSummary*
MinimumValue	valeur minimale de la plage de données
MaximumValue	valeur maximale de la plage de données
Text	texte affiché en cas de non validation

Exemple de validation entre 1 et 10 :

```
<asp:Label ID="Label1" runat="server" Text="Entrez une valeur
comprise entre 1 et 10 :"</asp:Label>
<asp:TextBox ID="TxtValeur" runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidator1" runat="server"
ErrorMessage="RangeValidator" MaximumValue="10" MinimumValue="1"
Type="Integer" ControlToValidate="TxtValue">Valeur entre 1 et 10
requise !</asp:RangeValidator>
```

1.5.3. CompareValidator

Il utilise un opérateur pour comparer les valeurs en présence et valider leur concordance. La situation la plus courante d'utilisation est, bien entendu, lors d'une deuxième saisie d'un mot de passe.
Les propriétés restent aussi dans les mêmes normes. Par contre, vous pouvez avoir plusieurs types de validation :

Comparaison à un type.

```
<asp:CompareValidator runat="server" ID="CompareValidator1"
ControlToValidate="TxtValeur" Type="Integer"
Operator="DataTypeCheck" ErrorMessage="Doit être un chiffre entier
de type integer !"</asp:CompareValidator>
```

Comparaison à une valeur.

```
<asp:CompareValidator runat="server" ID="CompareValidator1"
ControlToValidate="TxtValeur" Type="Integer"
Operator="GreaterThan" ValueToCompare="0" ErrorMessage="Un chiffre
positif est requis !"</asp:CompareValidator>
```

Comparaison à un autre champ.

```
<asp:CompareValidator runat="server" ID="CompareValidator1"
ControlToValidate="TxtMotPasse2" Type="String"
Operator="Equal" ControlToCompare="TxtMotPasse1"
ErrorMessage="Les mots de passe ne correspondent
pas !"</asp:CompareValidator>
```

1.5.4. RegularExpressionValidator

Ce contrôle valide un champ suivant une expression régulière. Il convient pour des tests de validation très complexes mais demande beaucoup de ressources donc, ne l'utilisez pas pour des validations qui peuvent se faire aisément avec plusieurs autres contrôles de validation.
Il utilise les mêmes propriétés que les contrôles précédents avec en plus une propriété *ValidationExpression* qui correspond évidemment à l'expression régulière de test.
Un petit exemple de validation d'un numéro de compte bancaire pour en voir l'application :

```
<asp:Label ID="Label1" runat="server" Text="Entrer votre numéro de
compte :"</asp:Label>
          <td><asp:TextBox ID="TxtCptBancaire"
runat="server"></asp:TextBox>
          <td><asp:RegularExpressionValidator
ID="RegularExpressionValidator1" runat="server"
           ErrorMessage="RegularExpressionValidator"
ControlToValidate="TxtCptBancaire"
           ValidationExpression="^\d{3}-\d{7}-\d{2}$">Format
incorrect
 </asp:RegularExpressionValidator>
```

1.5.5. CustomValidator

L'utilisateur définit lui-même une fonction pour effectuer la validation lorsque les contrôles standards ne peuvent pas assumer ce rôle.
Dans ce cas, les propriétés sont un peu différentes :

Dans le cas d'une validation côté client :

- La propriété **ClientValidationFunction** contient le nom de la fonction
- La fonction doit être sous la forme : *Function ValidationPersonnelle (source, arguments)*
- la **source** est l'objet *CustomValidator* côté client
- **arguments** est un objet comportant deux propriétés : *Value* et *IsValid*
- La propriété **Value** est la valeur à valider
- La propriété **IsValid** est un booléen retournant le résultat de la validation

La validation côté client s'effectue avec du code javascript soit entre les balises ad hoc, soit dans un fichier ".js" séparé.
Ce genre de code est bien connu des développeurs javascript :

```
<script language="javascript">
function Validation (obj, args)
{
}
       </script>
```

Dans le cas d'une validation côté serveur :
Placez le code de validation dans l'événement **OnServerValidate**

1.5.6. ValidationSummary

Ce contrôle n'est pas un contrôle de validation à proprement parler, il sert à afficher sous différentes formes le résultat de tous les contrôles de validation sur la page aspx si une erreur est survenue. Il est bien évident que vous pouvez l'omettre et gérer vous-même un affichage d'erreur.

Le contrôle *ValidationSummary* s'affiche dès que la propriété *IsValid* de la page est à *False*.
Il interroge les différents contrôles non valides et récupère la valeur de leur propriété *ErrorMessage*.
Pour afficher le résultat, vous avez les *DisplayMode* suivants à votre disposition :

- *List* : simple liste
- *BulletList* : liste avec puces
- *SingleParagraph* : les messages d'erreur sont concaténés les uns à la suite des autres, séparés par une virgule

L'emplacement de l'affichage peut s'effectuer de deux manières :

- à l'emplacement du contrôle *ValidationSummary* : mettre sa propriété *ShowSummary = True*
- dans une boite de dialogue : mettre sa propriété *ShowDialog = True*

Il est aussi intéressant de s'arrêter un peu à la propriété ValidationGroup des contrôles utilisateurs. En effet, regrouper certains contrôles sous un même nom dans la propriété ValidationGroup permet de valider d'abord une série de champs puis une autre suivant le résultat de la première validation.

2. L'accès aux données avec ASP.NET

2.1. *Introduction*

Les applications Web accèdent souvent à des sources de données aux fins de stockage et de récupération de données dynamiques. Vous pouvez écrire du code pour accéder aux données à l'aide de classes de l'espace de Synoms System.Data (connu sous le nom ADO.NET) et de l'espace de noms stem.Xml. Il s'agissait de l'approche généralement adoptée dans les versions antérieures d'ASP.NET.

Toutefois, ASP.NET permet également d'exécuter la liaison de données de façon déclarative. Cette liaison n'exige aucun code pour les scénarios de données les plus courants, et notamment :

- la sélection et l'affichage de données ;
- le tri, la pagination et la mise en cache de données ;
- la mise à jour, l'insertion et la suppression de données ;
- le filtrage de données à l'aide de paramètres d'exécution ;
- la création de scénarios maître/détails à l'aide de paramètres.

ASP.NET inclut deux types de contrôles serveur qui interviennent dans le modèle de liaison de données déclaratif : les contrôles de source de données et les contrôles liés aux données. Ces contrôles gèrent les tâches sous-jacentes exigées par le modèle Web sans état pour l'affichage et la mise à jour des données dans les pages Web ASP.NET. En conséquence, vous n'êtes pas tenu de connaître tout le déroulement du cycle de vie des demandes de page pour exécuter la liaison de données.

2.2. *Contrôles de source de données*

Les contrôles de source de données sont des contrôles ASP.NET qui gèrent les tâches de connexion à une source de données et de lecture et d'écriture de données. Les contrôles de source de données ne génèrent pas le rendu d'une interface utilisateur. Au lieu de cela, ils jouent le rôle d'intermédiaires entre un magasin de données particulier (base de données, objet métier ou fichier XML) et d'autres contrôles de la page Web ASP.NET. Les contrôles de source de données offrent des fonctionnalités puissantes de récupération et de modification de données, et notamment en termes de requêtes, de tri, de pagination, de filtrage, de mise à jour, de suppression et d'insertion. ASP.NET comprend les contrôles de source de données suivants :

Contrôle de source de données	Description
ObjectDataSource	Permet d'utiliser un objet métier ou une autre classe et de créer des applications Web qui s'appuient sur des objets de couche intermédiaire pour gérer des données.
SqlDataSource	Permet d'utiliser les fournisseurs de données managés ADO.NET, lesquels offrent un accès aux bases de données Microsoft SQL Server, OLE DB, ODBC ou Oracle.

AccessDataSource	Permet d'utiliser une base de données Microsoft Access.
XmlDataSource	Permet d'utiliser un fichier XML, ce qui est très utile pour les contrôles serveur ASP.NET hiérarchiques tels que les contrôles TreeView ou Menu.
SiteMapDataSource	Utilisé avec la navigation de site ASP.NET.

Les contrôles de source de données peuvent également être étendus pour prendre en charge d'autres fournisseurs d'accès au stockage des données.

2.3. *Contrôles liés aux données*

Les contrôles liés aux données génèrent le rendu des données en tant que balises au navigateur qui envoie la demande. Un contrôle lié aux données peut se lier à un contrôle de source de données et extraire automatiquement des données au moment opportun dans le cycle de vie de la demande de page. Les contrôles liés aux données peuvent tirer parti des fonctionnalités fournies par un contrôle de source de données, et notamment le tri, la pagination, la mise en cache, le filtrage, la mise à jour, la suppression et l'insertion. Un contrôle lié aux données se connecte à un contrôle de source de données via sa propriété DataSourceID.

ASP.NET comprend les contrôles liés aux données décrits dans le tableau suivant.

Contrôles de liste

Génère le rendu des données dans divers formats de liste. Les contrôles de type liste incluent les contrôles BulletedList, CheckBoxList, DropDownList, ListBox et RadioButtonList.

Contrôle	Fonctionnement
AdRotator	Génère le rendu des annonces dans une page en tant qu'images sur lesquelles les utilisateurs peuvent cliquer pour accéder à une URL associée à l'annonce.
Datalist	Génère le rendu des données dans une table. Le rendu de chaque élément est généré à l'aide d'un modèle d'élément que vous définissez.
DetailsView	Affiche un seul enregistrement à la fois sous une forme tabulaire et permet de modifier, de supprimer et d'insérer

	des enregistrements. Vous pouvez également parcourir plusieurs enregistrements.
FormView	Semblable au contrôle **DetailsView**, mais permet de définir une présentation de formulaire libre pour chaque enregistrement. Le contrôle **FormView** ressemble au contrôle **DataList** pour un enregistrement unique.
GridView	Affiche des données dans un tableau et propose une assistance pour l'édition, la mise à jour, le tri et la pagination des données sans nécessiter de code.
Menu	Génère le rendu des données dans un menu dynamique hiérarchique qui peut inclure des sous-menus.
Repeater	Génère le rendu des données dans une liste. Le rendu de chaque élément est généré à l'aide d'un modèle d'élément que vous définissez.
TreeView	Génère le rendu des données dans une arborescence hiérarchique de nœuds qu'il est possible de développer.

3. Master Page

3.1. Introduction aux MasterPages

Depuis longtemps, les développeurs ont toujours été contraint de dupliquer les sources HTML du design sur chaque page. D'autre solution intermédiaire existait, par exemple l'utilisation des frames en HTML, cependant cette fonction appartient au langage HTML et présente beaucoup de problème, notamment au niveau des emplacements, de plus les frames sont de moins en moins utilisées de nos jours. La fonctionnalité « MasterPage » a longtemps été demandée par les développeurs, elle n'existait toujours pas dans les versions précédentes de l'ASP.NET, grâce au MasterPage vous allez enfin pouvoir séparer les sources du design au code pur. En effet, intégré à la version 2.0, vous pouvez séparer la partie développement du design et développement fonctionnel du site, vous n'avez plus besoin de déployer les sources du design sur chaque page du site.

La MasterPage (extension *.master) contiendra la source (X)HTML du design et des zones d'édition (contentPlaceHolder), permettant ainsi de créer plusieurs page (contentPage) du même design. Il suffit ensuite de modifier les zones d'éditions sur chaque page. Par exemple, si l'on considère que le site de SUPINFO (www.supinfo.com) est en ASP.NET 2.0, une MasterPage est suffisante pour le design, ensuite chaque page fera appel à cette MasterPage dans l'en-tête, et chaque page devra insérer leurs informations spécifiques dans les zones d'éditions imposer dans la MasterPage

3.2. Création d'une MasterPage

Pour créer une MasterPage, faites un clique-droit sur le nom votre WebSite, puis sur « Add a New Item ».

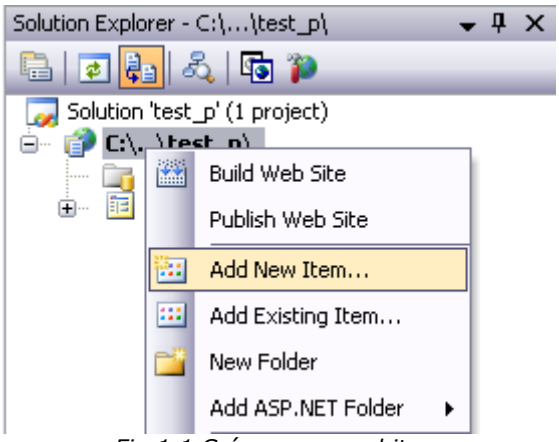


Fig 1.1 Créer un nouvel item

Une fenêtre apparaîtra, il vous suffira de choisir MasterPage et de renommer, si vous le souhaitez, le nom de la MasterPage.

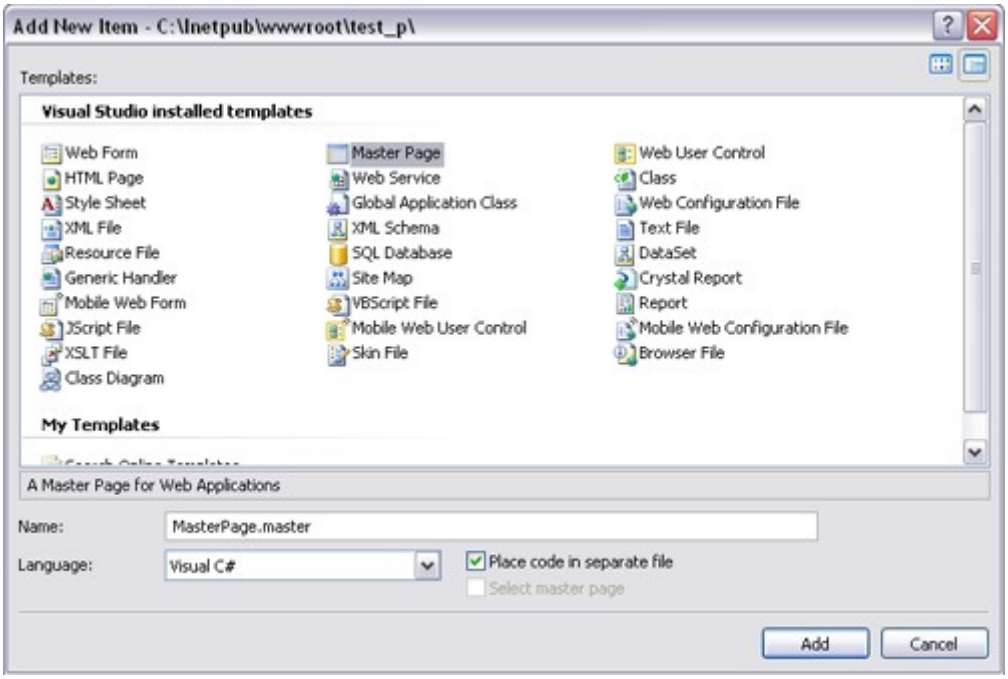


Fig 1.2 Créer un item MasterPage

Vous remarquerez que l'en-tête de la MasterPage contient le mot-clé « Master », à la différence d'une page simple qui contient le mot-clé « Page ».

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Il suffit ensuite d'insérer des zones d'éditions (contentPlaceHolder) aux endroits souhaiter grâce aux balise asp:contentPlaceHolder.

Voici un exemple simple de design avec 3 contentPlaceHolder :

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Test</title>
    <style type="text/css">
        body {
            font-size:14px;
            font-family:verdana;
            font-weight:bold;
            text-align:center;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
            <table   cellpadding="0"   cellspacing="0"   border="1"
style="width:600px; height:400px">
        <tr>
            <td colspan="2" style="height: 50px">
                                    <asp:ContentPlaceHolder  ID="top"
runat="server"></asp:ContentPlaceHolder>
            </td>
        </tr>
        <tr>
            <td style="width: 99px; height: 350px">
                            <asp:ContentPlaceHolder  ID="bottom_left"
runat="server"></asp:ContentPlaceHolder>
            </td>
            <td style="width: 500px; height: 350px">
                          <asp:ContentPlaceHolder  ID="bottom_right"
runat="server"></asp:ContentPlaceHolder>
            </td>
        </tr>
    </table>
    </form>
</body>
</html>
```

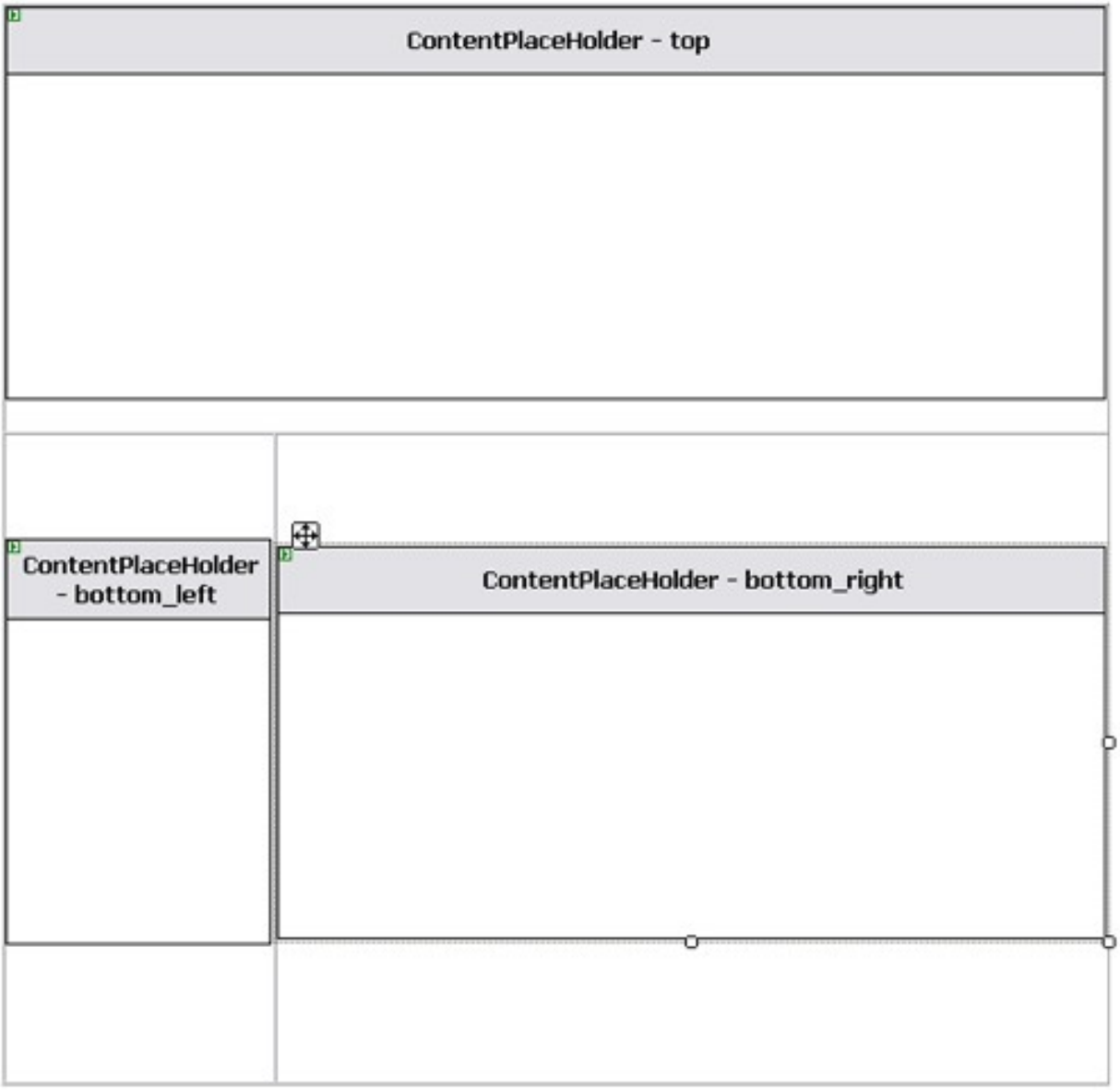


Fig 1.3 Exemple d'une MasterPage

Attention à ne rien rajouter dans les balises contentPlaceHolder si vous souhaitez éditer completement la zone. Par exemple si l'on rajoute une image dans la zone « top » de la MasterPage (exemple ci-dessus), cette image apparaîtra sur toutes les pages faisant appel à cette MasterPage.

3.3. Mise en place d'une MasterPage

Pour appliquer une MasterPage sur une nouvelle page, il faut tout d'abord enlever toute la source HTML qui vous ne sera pas utile, laisser uniquement l'en-tête :

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

Rajouter ensuite le paramètre MasterPageFile avec l'URL de la MasterPage à appliquer.

```
<%@ Page Language="C#" AutoEventWireup="true"
MasterPageFile="~/MasterPage.master" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

Enfin, il suffit de rajouter des balises contentPlaceHolder avec l'ID d'un contentPlaceHolder de la MasterPage, en voici un exemple :

```
<asp:Content ContentPlaceHolderID="top" runat="server">
TOP: HELLO, HOW ARE YOU?
</asp:Content>
<asp:Content ContentPlaceHolderID="bottom_left" runat="server">
BOTTOM LEFT: SUPINFO
</asp:Content>
<asp:Content ContentPlaceHolderID="bottom_right" runat="server">
BOTTOM RIGHT: 2005-2006
</asp:Content>
```

Voici le résultat de notre exemple :

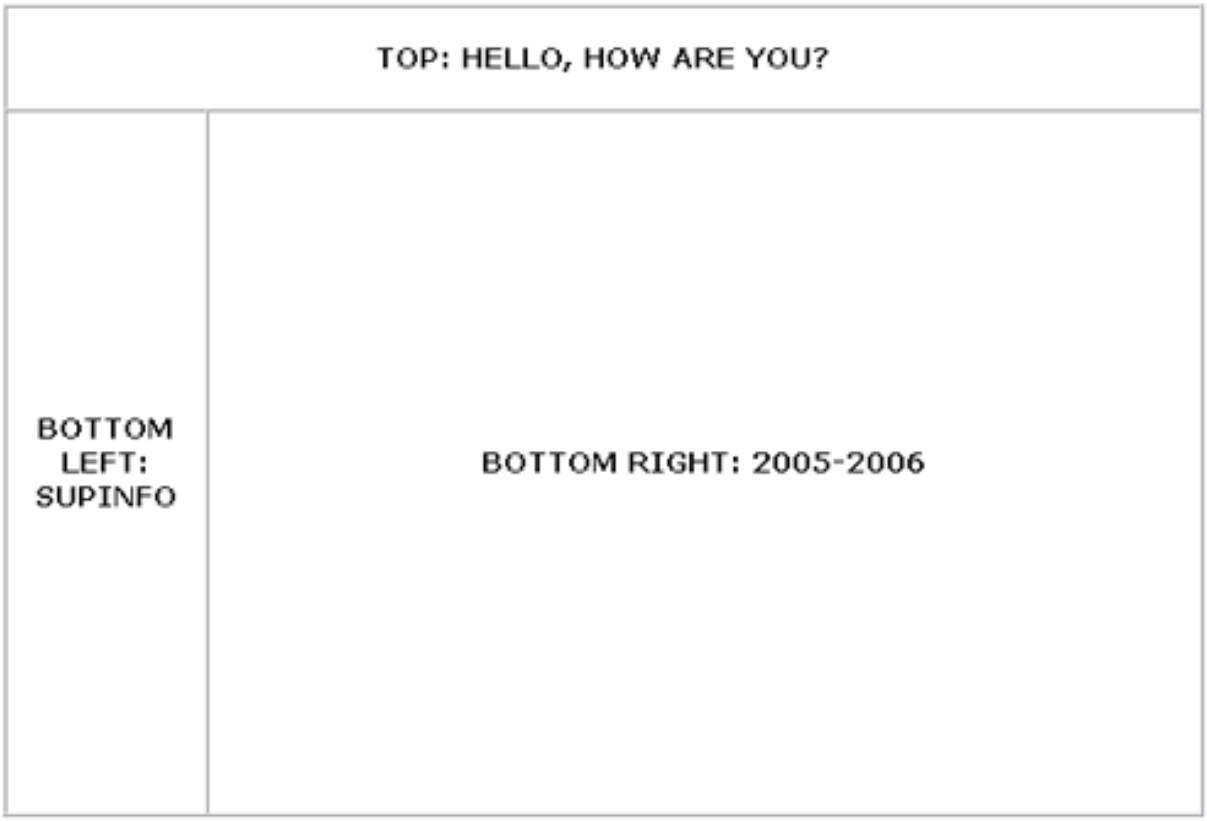


Fig 1.4 Exemple d'une page avec MasterPage

Les IDs des contentPlaceHolder sont uniques et l'affichage des IDs disponiblent lors de l'auto-complétion correspondent aux IDs des contentPlaceHolder de la MasterPage appellée.

3.4. Conclusion

Ainsi cette nouvelle fonctionnalité facilite la tâche du développeur en séparant le design du code, mais allège aussi l'architecture de l'application WEB. Grâce au MasterPage en ASP.NET 2.0, vous n'aurez plus besoin de

dupliquer le code source du design sur chaque page. Il est bien entendu possible de créer des MasterPage dynamiquement dans le code-behind, de créer d'imbriquer des MasterPage.

4. Thèmes et Skins

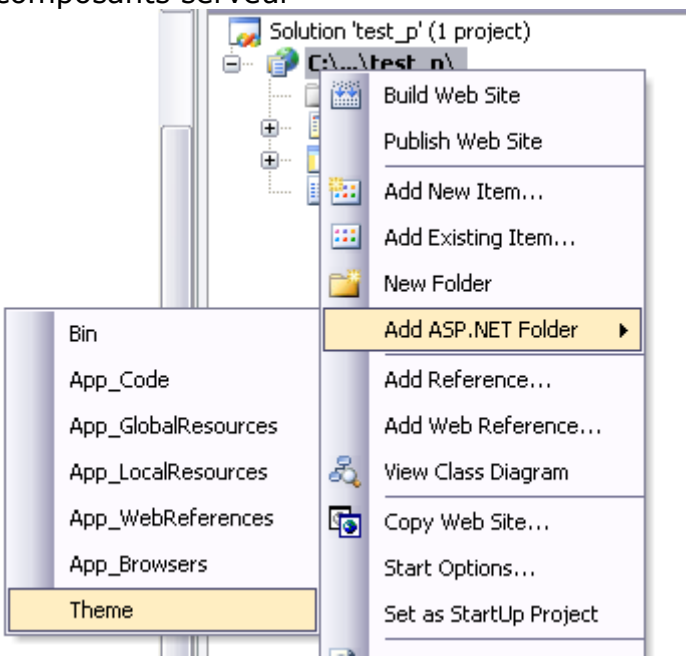
4.1. Introduction aux thèmes

Nous avons vu précédemment l'utilisation des MasterPages facilitant la mise en place et la duplication du design sur toutes les pages d'une application WEB, il manque cependant le formatage du site, plus exactement les couleurs, les polices, la taille des tableaux, la taille des différents composants etc….De plus les pages ne suivent pas forcément les même règles de formatages. C'est là qu'intervient les thèmes, plus exactement la mise en place d'un style sur un site.
La mise en place du style sur un site se faisait souvent par l'intermédiaire des sources HTML, par l'utilisation des CSS (Cascading Style Sheet). La différence entre les CSS et les thèmes est que les CSS ne permettent pas le formatage des contrôles ASP.NET. C'est pourquoi les fonctionnalités des CSS et thèmes sont tout à fait complémentaire dans l'implémentation un site en ASP.NET 2.0.

4.2. Création d'un thème

Les thèmes d'une application WEB en ASP.NET 2.0 sont stockés obligatoirement dans le dossier App_Themes, qui est un dossier spécifique ASP.NET. De plus, un thème représente un dossier composé au minimum d'un fichier Skin (*.skin).

Pour créer le dossier App_Themes, faites un clique-droit sur le nom de votre application WEB (et non la solution) puis «Add an ASP.NET Folder» puis « Theme ».



Vous avez ensuite la possiblité d'ajouter des nouveaux thèmes en allant dans App_Themes > Add ASP.NET Folder > Theme
Ensuite, pour ajouter un nouveau fichier Skin ou CSS à votre thème, il suffit tout simplement de faire un clique-droit sur le thème et « Add New Item ».

4.3. Les fichiers Skins

Les fichiers Skins sont composés essentiellement de côntrole ASP.NET avec leurs propriétés de formattage (police, couleur, taille…). Vous avez la possibilité d'ajouter des contrôles communs (sans SkinID) ou des contrôles spécifiques (avec SkinID). Si le thème est appliqué, les contrôles communs s'appliqueront à tous les contrôles du même type alors que les contrôles spécifiques seront appliqués à tous les contrôles du même type et ayant le même SkinID.

Exemple d'un contrôle commun de type TextBox :

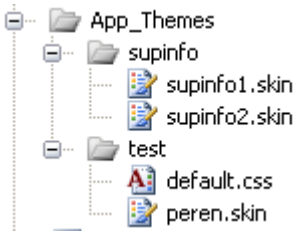
```
<asp:TextBox runat="server"
Font-Size="10px" Font-Names="Verdana" ForeColor="red" />
```

Si le thème est appliqué sur tout le site, toutes les TextBox auront le même style

Exemple d'un contrôle spécifique de type TextBox :

```
<asp:TextBox runat="server" SkinID="specifique"
Font-Size="10px" Font-Names="Verdana" ForeColor="red" />
```

Si le thème est appliqué sur tout le site, toutes les TextBox ayant comme SkinID « specifique » auront le même style



Voici un exemple de fichier Skin non exhaustif :

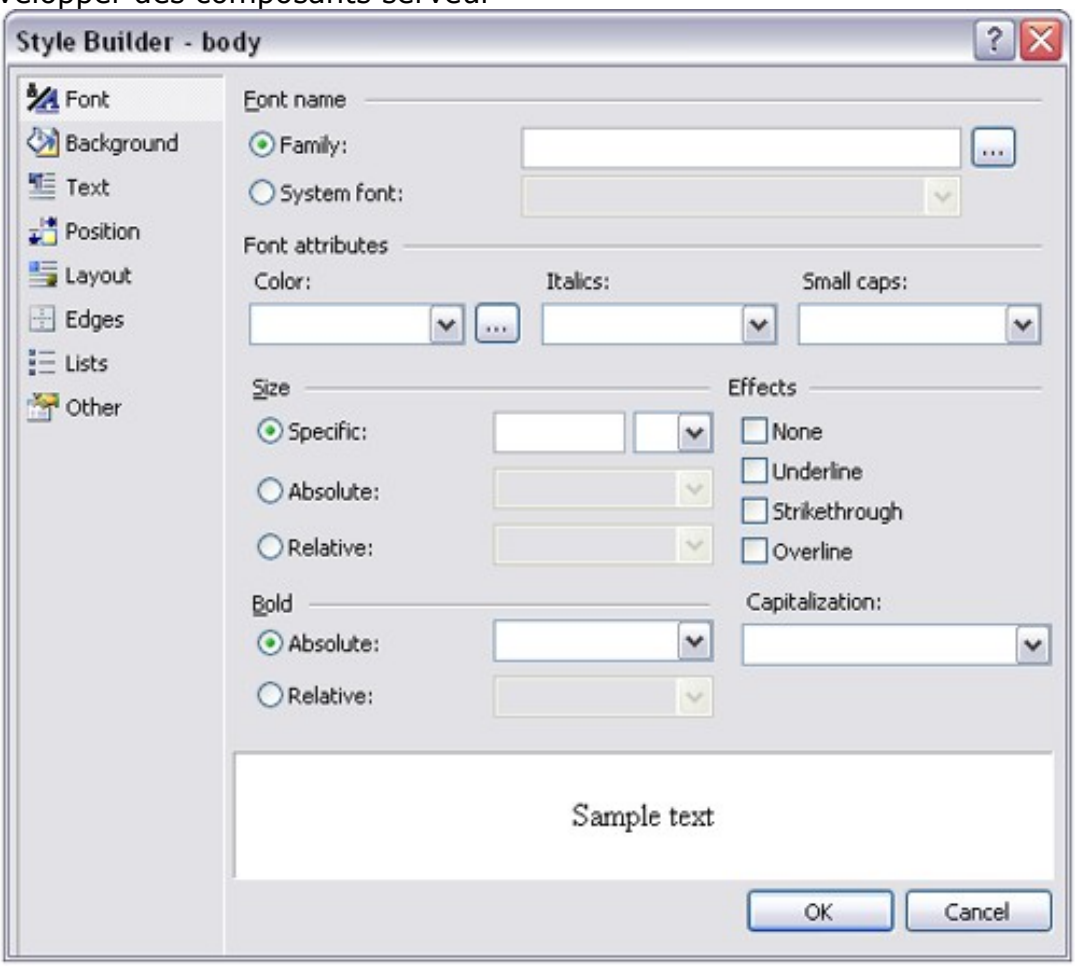
```
<asp:TextBox runat="server"
  ForeColor="#000000"
  font-name="Verdana"
  Font-Size="9px"
  height="13px"
  borderWidth="1px"
  borderStyle="Solid"
  BorderColor="#82ACFF"
  BackColor="#FFFFFF"
  font-bold="false" />

<asp:DropDownList runat="server"
  Font-Name="Verdana"
  Font-Size="10px"
  ForeColor="#000000"
  Font-Bold="false"
  borderWidth="1px"
  borderStyle="Solid"
  BorderColor="#82ACFF"
  BackColor="#FFFFFF" />

<asp:ListBox Runat="Server"  Font-Name="Verdana"
  Font-Size="10pt"
  BackColor="#FFFBFF"
  ForeColor="gray"/>

<asp:GridView Runat="Server" SkinId="test"
  CellPadding="1"
  BorderStyle="solid"
  BorderWidth="1px"
  BackColor="whitesmoke"
  ForeColor="#93B41A"
  HeaderStyle-Font-Names="Verdana"
  HeaderStyle-Font-Size="10px"
  HeaderStyle-ForeColor="gray"
  HeaderStyle-BackColor="white"
  HeaderStyle-Font-Bold="true"
  RowStyle-Font-Names="Verdana"
  RowStyle-Font-Size="10pt"
  RowStyle-ForeColor="#93B41A"
  RowStyle-BackColor="whitesmoke"
  AlternatingRowStyle-BackColor="white"
  AlternatingRowStyle-ForeColor="#93B41A" />

  <asp:Label runat="server" SkinID="LabelWhite"
  ForeColor="white"
  font-name="Verdana"
  Font-size="10pt" />

  <asp:Label runat="server" SkinID="LabelUnderline"
  ForeColor="Gray"
```

```
font-name="Verdana"
Font-size="10pt"
Font-Underline="true" />

<asp:FileUpload runat="server"
ForeColor="Gray"
font-name="Verdana"
Font-Size="10pt"
borderWidth="1px"
BorderColor="#CCCCCC"
BackColor="#FFFBFF" />

<asp:Button runat="server"
BackColor="#FFFBFF"
BorderColor="#CCCCCC"
BorderStyle="solid"
BorderWidth="1px"
Font-name="Verdana"
Font-size="8pt"
ForeColor="Darkgray" />

<asp:RadioButtonList runat="server"
Font-Name="Verdana"
Font-Size="11px"
ForeColor="gray" />

<asp:CheckBox runat="server"
BackColor="#FFFBFF"
Font-Name="Verdana"
Font-Size="11px"
ForeColor="gray" />

<asp:RequiredFieldValidator runat="server"
Font-Name="Verdana"
Font-Size="11px"
ForeColor="red" />

<asp:CompareValidator runat="server"
Font-Name="Verdana"
Font-Size="11px"
ForeColor="red" />
```

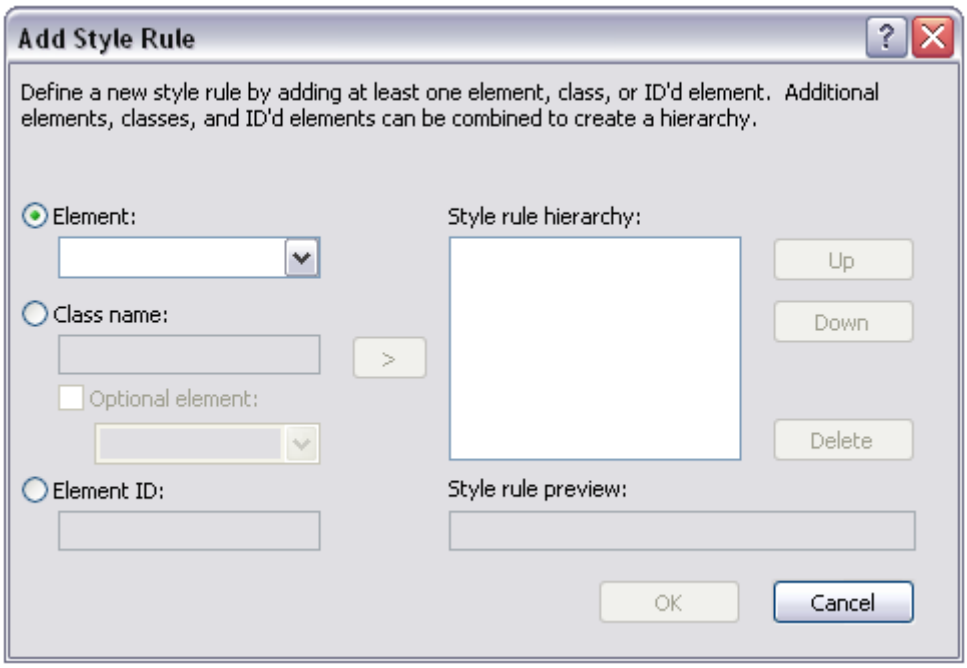
Il es tout à fait possible de créer plusieurs fichiers Skins dans un même thème, cependant les contrôles ayant un SkinID doivent être unique dans un thème, sinon vous ne pourrez pas compiler votre application WEB.

4.4. Les fichiers CSS

Visual Studio 2005 intègre un outil de génération de style CSS par l'interface « Style Builder » permettant de définir vos style par le biais d'une interface simple. Pour afficher l'interface, il suffit de faire un clique-droit dans votre fichier css (dans une zone de style) puis « Build Style ».



Une interface pour ajouter des style CSS est disponible par un clique-droit dans votre feuille de style CSS puis « Add Style Rule ».



L'utilisation des CSS dans le cas d'une application WEB permet uniquement d'appliquer un style aux contrôles HTML, mais pas aux contrôles ASP.NET

4.5. Application d'un thème

Un style peut être directement appliqué au contrôle ASP.NET en modifiant directement les propriétés d'un contrôle. Bien entendu, l'intérêt des Thèmes est de pouvoir « stocker » différents thèmes et pouvoir les appliquer plusieurs fois sans avoir à redéfinir le thème.

Voici les principaux propriétés de style des contrôles :

Propriétés	Type	Description
BackColor	Texte	Couleur de l'arrière plan
BorderColor	Texte	Couleur de la bordure
BorderStyle	Texte	Style de la bordure
BorderWidth	Pixels	Largeur de la bordure
Font-Bold	Booleen	Caractère en gras
Font-Italic	Booleen	Caractère en italique
Font-Name	Texte	Police
Font-Overline	Booleen	Caractère surligné
Font-Size	Pixels	Taille des caractères
Font-Strikeout	Booleen	Caractère barré
Font-Underline	Booleen	Caractère sougliné
ForeColor	Texte	Couleur des caractères
Height	Pixels	Hauteur du contrôle
Visible	Booleen	Visibilité du contrôle
Width	Pixels	Largeur du contrôle

Pour appliqer un thème automatiquement, il suffit de spécifier le nom du thème dans l'en-tête de la page.

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="supinfo"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

Enfin, il faudra spécifier dans chaque contrôle le SkinID si vous souhaitez appliquer des styles spécifiques ayant un SkinID.

Sans thème	Avec thème

4.6. Appliquer un thème global

Pour appliquer un thème à tout le site, pour éviter d'appliquer le thème page par page, il faut définir le thème à appliquer dans le fichier de configuration (Web.config) entre les balises system.web , en voici un exemple :

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
          <compilation debug="true"/>
          <authentication mode="Windows"/>
    <pages theme="test"/>
    </system.web>
  </configuration>
```

Dans ce cas, vous n'aurez pas à spécifier la propriété « Theme » sur chaque en-tête de chaque page. Par contre si votre thème contient des contrôles avec SkinID, il faudra les spécifier dans toutes vos pages.

4.7. Désactiver un thème

Les thèmes appliqués sont prioritaires, ce qui n'est pas pratique si l'on souhaite appliquer un style spécifier à un et un seul contrôle. Dans ce cas vous avez la possibilité d'utilisé la propriété EnableTheming (booleen) disponible pour un contrôle ou une page.

```
<%@        Page        Language="C#"        AutoEventWireup="true"
CodeFile="test.aspx.cs" EnableTheming="false" Inherits="test " %>
```

Les thèmes sont désactivés sur la page test.aspx.

```
<asp:TextBox  ID="TextBox1"  ForeColor="red"  EnableTheming="false"
runat="server"></asp:TextBox>
```

La TextBox n'aura pas de thème appliqué, seul les propriétés de style définit dans le contrôle sera valable.

5. Profiles

5.1. Introduction aux Profiles

Plusieurs façons de stocker les informations des utilisateurs sont envisageable telles que les sessions, les cookies utilisés principalement jusque là pour garder une trace d'un visiteur ou pouvoir stocker des informations tel qu'un panier, un nom. Cependant ces solutions requierent un minimum de code de la part du développeur. ASP.NET intègre une nouvelle solution, les Profiles, se basant sur une base de donnée (SQL Server, Access, Oracle…) et permet de stocker des informations permanant et automatiquement. Cette nouvelle fonctinnalité de nécessite aucun code pour le stockage, ASP.NET se charge de cette fonction et modifie à chaque fois la base de donnée. De plus, les Profiles permettent de stocker des informations à long terme, souvent associées à l'Authentification des utilisateurs dans les applications WEB.

L'utilisation des Profiles sera détaillée uniquement avec une base de donnée SQL Server ainsi que l'installation des Profiles par l'intermédiaire de l'application aspnet_regsql.exe de Microsoft, intégré dans le Framework 2.0.

5.2. Implémentation des Profiles

Microsoft met à disposition un executable permettant d'installer les Profiles (de même pour l'utilisation des Membership et Roles) dans une base de donnée.

Cette executable se trouve dans le chemin suivant :
C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxx\aspnet_regsql.exe

Il vous suffit de lancer le programme, choisir si vous voulez créer une nouvelle base (ASPNETDB sera le nom de la base par défaut) ou installer tous les composants dans une base existante.

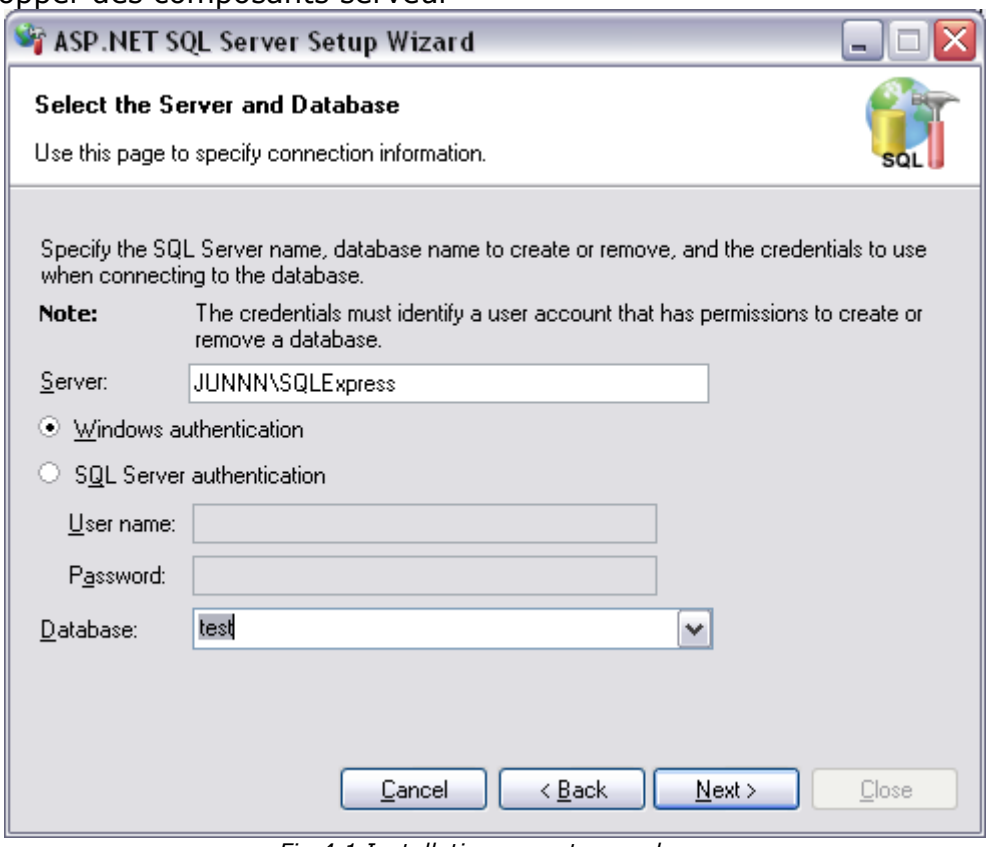


Fig 4.1 Installation aspnet_regsql.exe

Voici les nouvelles tables installés :

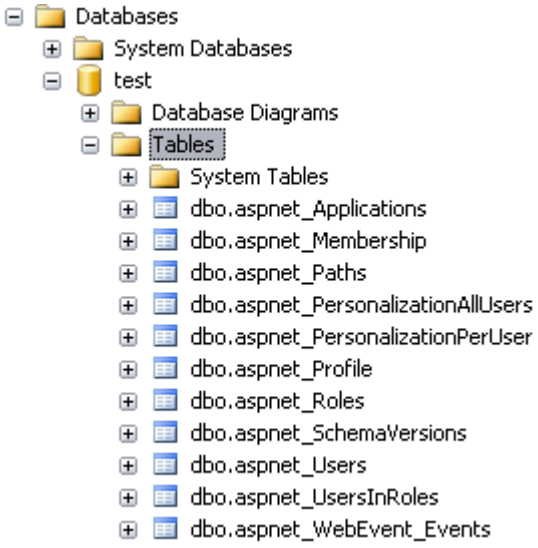


Fig 4.2 Tables installées

Et les procédures stockées concernant les Profiles :

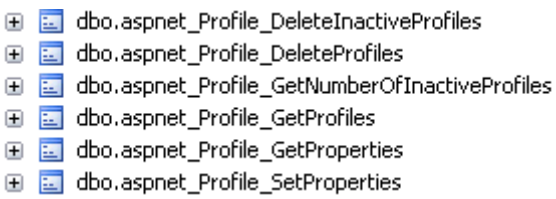


Fig 4.3 Procédures pour les Profiles

5.3. Description des tables et procédures

L'utilisation des Profiles nécessite une base de donnée stockant les informations grâce à des procédures stockées. Quelque soit la procédure d'installation des Profiles, vous tomberez sur le même schèma de base de donnée.

Table	Description
aspnet_Applications	Informations concernant toutes les application WEB disponibles
aspnet_Users	Informations concernant tous les utilisateurs (status de connexion, dernière connexion…). Relié à un ou plusieurs Profile et une application.
aspnet_Profile	Informations concernent les Profiles de plusieurs utilisateurs avec la dernière date de modification

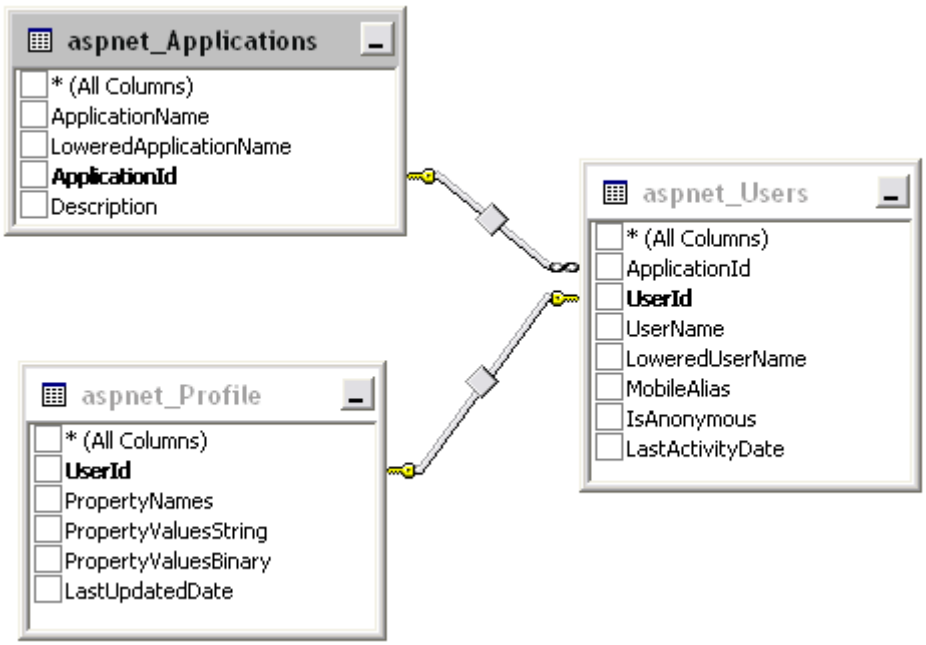


Fig 4.4 Structure des tables Profiles

Procédures	Description
aspnet_Profile_DeleteInactiveProfiles	Supprime les profiles inactifs selon une durée à spécifier
aspnet_Profile_DeleteProfiles	Supprime les Profiles selon les utilisateurs
aspnet_Profile_GetNumberOfInactiveProfiles	Récupère le nombre de Profiles inactifs depuis un certain temps (à spécifier)
aspnet_Profile_GetProfiles	Permet de récuperer des informations sur les Profiles, tel que le propriétaire utilisateur, la dernière date de

	modification
aspnet_Profile_GetProperties	Permet de récuperer le contenu d'un Profile selon le nom de l'utilisateur
aspnet_Profile_SetProperties	Permet de modifier un Profile selon l'utilisateur

5.4. Mise en place des Profiles

Les Profiles se définissent de le fichier de configuration de l'application WEB, le fichier XML web.config, entre les balises system.web.

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
          <compilation debug="true"/>
          <authentication mode="Windows"/>

                <profile defaultProvider="">
                <providers>
                ...
                </providers>
                <properties>
                <add ... />
                </properties>
          </profile>

    </system.web>
  </configuration>
```

L'utilisation des Profiles est associée à un Provider (Profile Provider), donc une base de donnée. En l'occurrence, vous avez la possibilité de choisir un Provider par defaut, c'est-à-dire le Provider définit dans le champs ConnectionString du fichier web.config, ou de définir vous-même le provider entre les balises Provider.

La balise Properties permet de définir les informations qui seront stockées dans la base de donnée, voici les propriétés de la balise Add:

Propriétés	Description
Name	Nom de la propriété, auquel on pourra accéder dans le code-behind
allowAnonymous	Booleen, autorisé l'utilisation de cette propriété pour les utilisateurs avec un Profile anonyme
Type	Le type de la propriété (String, Integer, …)
Provider	Le Provider associé
serializeAs	Le type de serialisation (String, Binary, …)
readOnly	Propriété en mode lecture uniquement
DefaultValue	La valeur par défaut de la propriété
Group	Ordonner les propriétés par groupe

Voici un exemple de Profile :

```
<profile enabled="true">
      <providers>
                                    <add     name="AspNetSqlProvider"
type="System.Web.Profile.SqlProfileProvider"
connectionStringName="xxx"></add>
      </providers>
      <properties>
             <add  name="panier"  allowAnonymous="true"  type="Cart"
provider="AspNetSqlProvider" serializeAs="Binary"/>
                  <add   name="destinataire"   allowAnonymous="true"
type="Destinataire"                    provider="AspNetSqlProvider"
serializeAs="Binary"/>
      </properties>
    </profile>
```

Dans cet exemple, les type « Cart » et « Destinataire » sont des classes.

5.5. Ajouter / Modifier les propriétés

Une fois les propriétés des Profiles misent en place dans le fichier web.config, vous allez pouvoir stocker des données dans un Profile. Attention, si vous souhaitez stocker des informations dans un Profile anonyme, il faut activer les utilisateurs anonyme dans le fichier web.config.

<anonymousIdentification enabled="true"/>

L'accès au propriété du profile en code-behind (C#) se fait de cette manière :

```
Profile.[propriete] = [valeur];
```

Voici un exemple :

Le fichier web.config :

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings>
       <add    name="testConnectionString"    connectionString="Data
Source=JUN\SQLExpress;Initial               Catalog=test;Integrated
Security=True"
   providerName="System.Data.SqlClient" />
 </connectionStrings>
    <system.web>
          <compilation debug="true"/>
          <authentication mode="Windows"/>
    <pages theme="test"/>

    <anonymousIdentification enabled="true"/>
    <profile enabled="true">
      <providers>
                                    <add     name="AspNetSqlProvider"
type="System.Web.Profile.SqlProfileProvider"
connectionStringName="testConnectionString"></add>
      </providers>
```

```
      <properties>
            <add  name="age"  allowAnonymous="true"  type="Integer"
provider="AspNetSqlProvider" serializeAs="String"  />
      </properties>
    </profile>
    </system.web>
  </configuration>
```

Le fichier test.aspx :

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
                              Age <asp:TextBox    ID="TxtBox_Age"
runat="server"></asp:TextBox>
        <br />
                  <asp:Button  ID="Button_validate"  runat="server"
Text="Valider" OnClick="Button_valide_Click" /><br />
        <br />
    </div>
    </form>
</body>
</html>
```

Le fichier test.aspx.cs :

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class test : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Button_valide_Click(object sender, EventArgs e)
    {
        Profile.age = Convert.ToInt32(TxtBox_Age.Text);
    }
}
```

Fig 4.5 Exemple stockage information

Le contenu de la table aspnet_Profile avant validation :

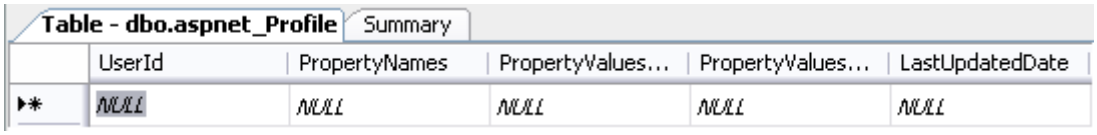


Table - dbo.aspnet_Profile | Summary

	UserId	PropertyNames	PropertyValues...	PropertyValues...	LastUpdatedDate
▶*	NULL	NULL	NULL	NULL	NULL

Fig 4.6 Tables aspnet_Profile avant validation

Le contenu de la table aspnet_Profile après validation :

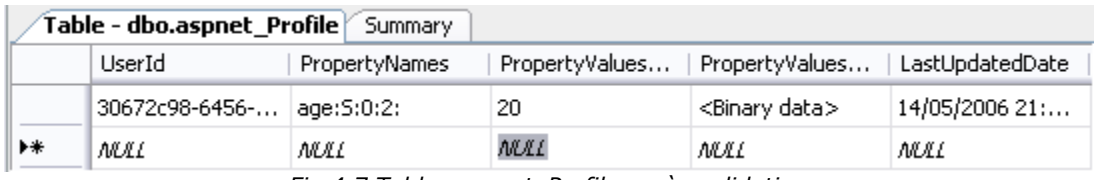


Table - dbo.aspnet_Profile | Summary

	UserId	PropertyNames	PropertyValues...	PropertyValues...	LastUpdatedDate
	30672c98-6456-...	age:S:0:2:	20	<Binary data>	14/05/2006 21:...
▶*	NULL	NULL	NULL	NULL	NULL

Fig 4.7 Tables aspnet_Profile après validation

Pour récuprer cette donnée, il suffit de la récupérer dans un Label ou autre contrôle avec ce bout de code :

```
… = Profile.age;
```

5.6. Les différents type de sérialisation

Les informations sont sérialisées puis stockés dans un Profile. Quatres type de sérialisations sont disponibles :

Type	Description	Symbole
String	Converti en String	S
Binary	Converti en Binaire	B
Xml	Converti en Xml	X
ProviderSpecific	Converti selon le type du Provider	P

La propriété « PropertyNames » a une structure permettant de reconnaître les différentes propriétés du Profile enregistré. Sa structure est composée du nom de la propriété enregistrée, du type de serialisation, le début de la chaîne concerné, la taille de la chaîne :

Nom_propriété:type_serialisation:debut_chaîne:taille_chaîne

S'il existe plusieurs propriétés, elles sont toutes concaténées.

5.7. Les groupes de propriétés

Les groupes de propriétés permettent de regrouper les propriétés à stocker dans la base de donnée selon leur fonction ou type, permettant d'ordonner le Profile si celui-ci est composé d'une multitude de propriétés. Cela permet aussi d'avoir le même nom de propriété dans chaque groupe puisque par défaut, une propriété est unique ou par groupe.

Voici un exemple de Profile « groupé » :

```
<profile enabled="true">
      <providers>
                                        <add     name="AspNetSqlProvider"
type="System.Web.Profile.SqlProfileProvider"
connectionStringName="testConnectionString"></add>
      </providers>
      <properties>
        <group name="Students">
          <add name="Campus_ID" />
          <add name="Nom" />
          <add name="Prénom" />
          <add name="Région" />
          <add name="Promo" />
          <add name="Pole" />
        </group>
      </properties>
    </profile>
```

Pour accéder aux propriétés, il suffit de spécifier le groupe, par exemple :

```
… = Profile.Students.Nom;
```

5.8. Conclusion

Cette nouvelle fonctionnalité permet ainsi de stocker automatiquement et permanant des informations dans une base de donnée. Bien entendu, les développeurs préfèreront développer eux-mêmes le stockage d'information temporaire ou à long terme, cependant la solution des Profiles de Microsoft permet une implémentation simple et rapide.

Si vous souhaitez migrer un Profile Anonyme vers un Profile Connecté (par exemple pour un panier) il faut créer un fichier Global.asax comme cet exemple :

```
<%@ Application Language="C#" %>

<script runat="server">

                    void   Profile_MigrateAnonymous(Object   sender,
ProfileMigrateEventArgs pe)
    {
        // vous ajoutez toutes les propriétés à migrer

        Profile.panier = Profile.GetProfile(pe.AnonymousID).panier;
                                              Profile.destinataire     =
Profile.GetProfile(pe.AnonymousID).destinataire;

       // on supprimer le profile anonyme
        AnonymousIdentificationModule.ClearAnonymousIdentifier();

    }

</script>
```

6. Sécurité en ASP.NET 2.0

6.1. *Introduction*

Chaque site Internet nécessite une certaine sécurité pour éviter tout problèmes. La nouvelle version d'ASP.NET 2.0 de Microsoft, se basant sur le Framework 2.0, permet une meilleure sécurité et simplifie la tâche du développeur. En effet, l'ASP.NET 2.0 est un nouveau langage de Web développement pouvant être qualifier de "Nouvelle génération". Elle fait partie des nouvelles innovations du WEB 2.0 (AJAX, ASP.NET 2.0...), la nouvelle génération des websites. Grâce à ce langage, nous allons pouvoir appliquer des autorisations d'accès à certaines pages/dossiers, implémenter une gestion d'utilisation sécurisée et cryptée par l'utilisation des Membership et roles.

L'ASP.NET 2.0 a été développé de tel sort qu'il falicite le développeur, en réduisant son temps de travail de 80%. En effet, l'ASP.NET 2.0 consiste vraissemblablement au VB en "Drag and Drop", d'où la hausse de productivité pour les sociétés qui utiliseront ce langage. Ce langage intègre, par ailleurs, de nouveaux composants tel que les thèmes et skins, les masterpages, les WebParts, la personalisation, les profils, et bien d'autres nouveautés.

6.2. *Le fichier de configuration: Web.config*

Le fichier Web.config est un fichier XML permettant de gérer la configuration de votre Website ou application. Il se trouve en général à la racine du site et chaque sous dossiers peuvent contenir un fichier de configuration.
Par exemple si vous êtes sur IIS, vous avez la possibilité de placer un fichier web.config à la racine (C:\InetPub\wwwroot), ainsi que chaque sous dossiers de wwwroot.

Ce fichier permet de gérer plusieurs paramètres d'un site web, notamment les accès, les autorisations, les sessions, les mails, les erreurs ainsi que leur redirections etc...

Voici l'exemple d'un fichier web.config:

```
<?xml version="1.0"?>

<configuration
xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
    <appSettings/>
     <connectionStrings>
          <add     name="xxxxxxx"     connectionString="Data
Source=    xxxxxxx;Initial    Catalog=    xxxxxxx;Integrated
Security=True" providerName="System.Data.SqlClient"/>
     </connectionStrings>
    <system.web>

          <machineKey
validationKey='EB5219CAB5504274A423EB78718F3E56DC8848
AFA6025B6D740D46AA2394EBAD1BB9FE6BD9141A691A971
7CFEDC77FB2788BBBC2CD80CA6C3EE02ACF99B04BA5'
```

```
        decryptionKey='685F5FA7DD04AEE2A3C203A55612D6
        A1A366F37491BED4B5'
        validation='SHA1'/>

             <membership defaultProvider="MembershipSqlProvider">
                   <providers>
                         <add name="MembershipSqlProvider"

             type="System.Web.Security.SqlMembershipProvider"
                              connectionStringName=" xxxxxxx "
                              enablePasswordRetrieval="true"
                              passwordFormat="Encrypted"
                              applicationName=" xxxxxxx ">
                         </add>
                   </providers>
             </membership>

             <roleManager                                    enabled="true"
        defaultProvider="RoleManagerSqlProvider">
                   <providers>
                      <add connectionStringName=" xxxxxxx"
                           applicationName=" xxxxxxx "
                           name="RoleManagerSqlProvider"
                        type="System.Web.Security.SqlRoleProvider" /
        >
                   </providers>
                   </roleManager>

             <profile enabled="true">
               <providers>
                  <add name="AspNetSqlProvider"
                     type="System.Web.Profile.SqlProfileProvider"
                   connectionStringName=" xxxxxxx ">
                   </add>
             </providers>
             <properties>
             <add          name="Cart"          allowAnonymous="true"
        type="W2TM.Web.framework.Cart"  provider="AspNetSqlProvider"
        serializeAs="Binary"/>
             </properties>
             </profile>

                   <anonymousIdentification
                     enabled="true"
                     cookieName="ASPXANONYMOUS"
                     cookieTimeout="20"
                     cookiePath="/"
                     cookieRequireSSL="false"
                     cookieSlidingExpiration="true"
                     cookieProtection="All"
                     cookieless="UseCookies">
                   </anonymousIdentification>

                   <sessionState              cookieless="UseCookies"
        mode="InProc"></sessionState>

                   <compilation debug="true"/>

                   <customErrors                                mode="Off"
        defaultRedirect="~/erreur.aspx"/>

                   <authentication mode="Forms">
```

```
                    <forms defaultUrl="default.aspx"
                           loginUrl="login.aspx"
                           requireSSL="false"
                           protection="All">
                      </forms>
                </authentication>

                <authorization>
                      <deny users="*" />
                </authorization>

          </system.web>
           <location path="admin">
                <system.web>
                      <authorization>
                            <allow roles="admin"/>
                      </authorization>
                </system.web>
           </location>
           <location path="erreur.aspx">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
           </location>
           <location path="Default.aspx">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
           </location>
           <location path="uploads">
                <system.web>
                      <authorization>
                            <allow users="*" />
                      </authorization>
                </system.web>
           </location>
           <location path="master">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
           </location>
           <location path="fr">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
           </location>
           <location path="en">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
           </location>
           <location path="login.aspx">
```

```
                <system.web>
                      <authorization>
                            <allow users="?"/>
                      </authorization>
                </system.web>
          </location>
          <location path="images">
                <system.web>
                      <authorization>
                            <allow users="*"/>
                      </authorization>
                </system.web>
          </location>
          <system.net>
                <mailSettings>
                      <smtp>
                            <network host="localhost" />
                      </smtp>
                </mailSettings>
          </system.net>
    </configuration>
```

Ce fichier web.config est tiré d'un des sites Exemples. Tout au long de cet article vous apprendrez petit à petit comment sécuriser votre site par ce fichier de configuration XML.

6.3. *Utilisation des Memberships et rôles*

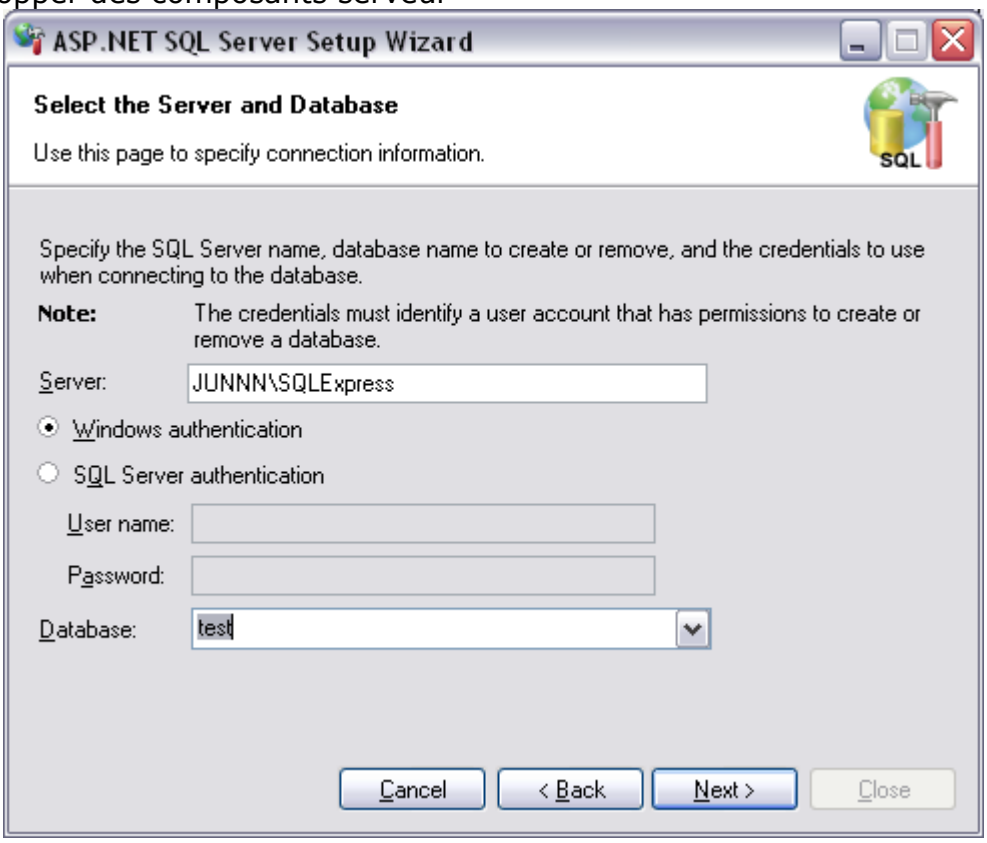
6.3.1. Installation de la base

Avant toutes choses, les memberships et rôles s'utilisent obligatoirement avec une base de donnée (SQL Server, Acces, Oracle). Nous allons uniquement prendre le cas de SQL Server.

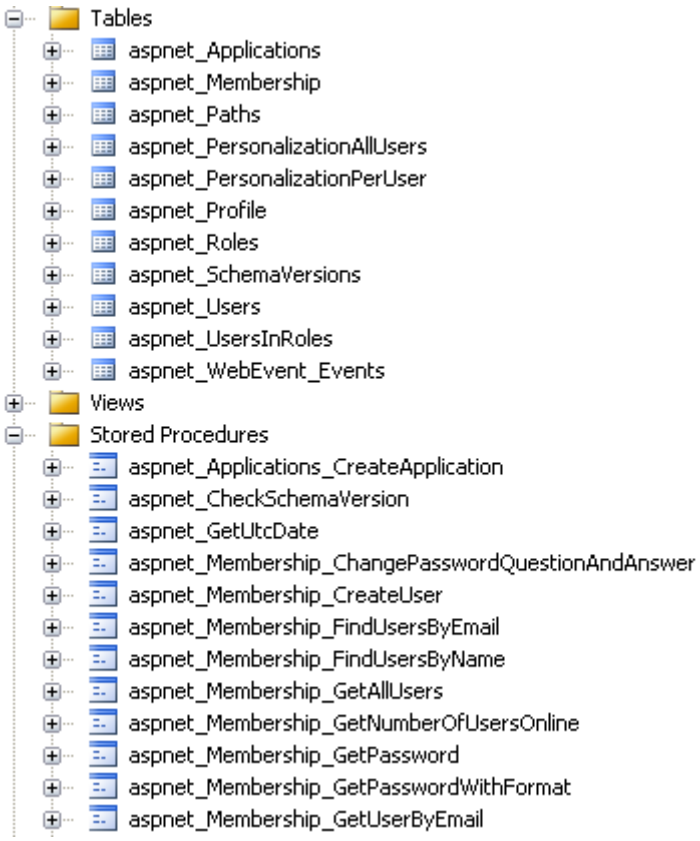
Pour SQL Server, il vous faut installer la base ASPNETDB (Framework 2.0) qui contient des tables, des procédures stockés, vous permettant d'utiliser les memberships, les rôles, les profiles et donc la personnalisation.

L'installation se trouve en général dans:
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50215\aspnet_regsql.exe

Il vous suffit de lancer le programme, choisir si vous voulez créer une nouvelle base (ASPNETDB sera le nom de cette base) ou installer tous les composants dans une base existante.

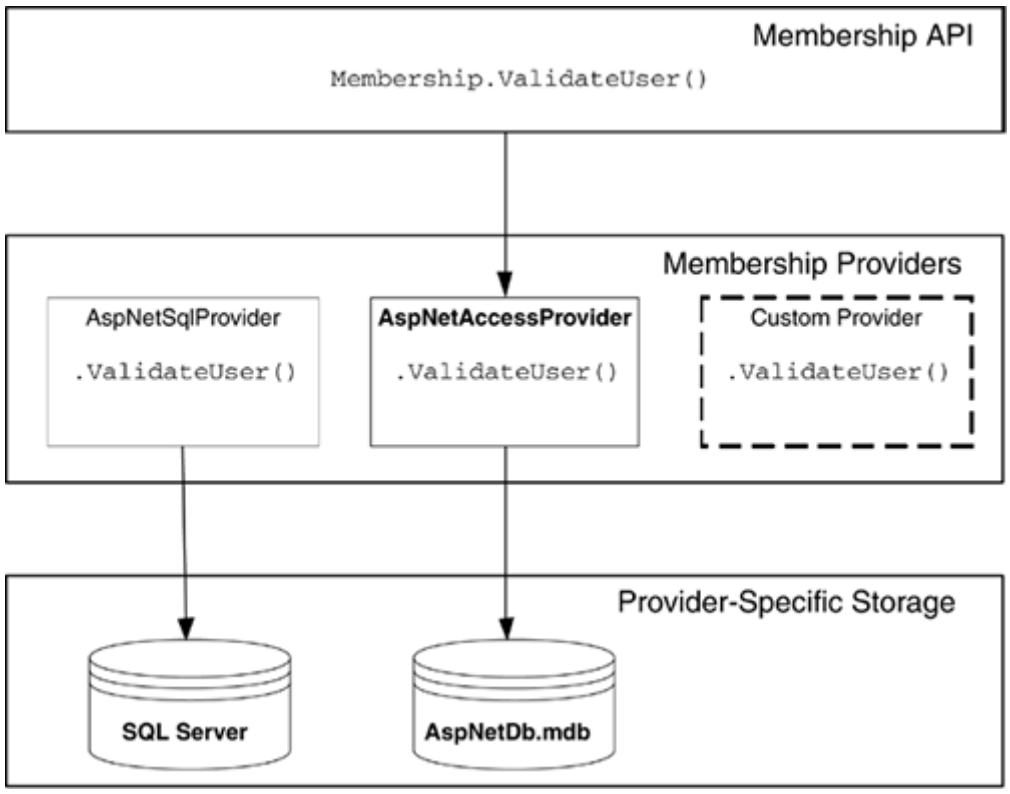


Voici une petite partie de l'arborescence de votre base de donnée après installaton:



6.3.2. Memberships

A la différence des versions précédentes, l'ASP.NET 2.0 inclut une utilisation simple des memberships et rôles, grâce à l'implémentation des providers, nouveauté dans le .NET Framework 2.0, permettant un accès à différentes bases, de pouvoir contrôler et choisir où stocker des informations dans une base. Il existe différents providers possibles pour les memberships: Acces, SQL Server... voici un schéma montrant le modèle du provider et son fonctionnement:



Pour utiliser les memberships dans votre projets, vous devez obligatoirement modifier le fichier Web.config (ou modifier par l'intermédiaire du WSAT), en voici un exemple (entre les balises system.web):

```
<membership defaultProvider="MembershipSqlProvider">
<providers>
<add name="MembershipSqlProvider"
type="System.Web.Security.SqlMembershipProvider"
connectionStringName="votre_ConnectionString"
enablePasswordRetrieval="true" passwordFormat="Encrypted"
applicationName="nom_application">
</add>
</providers>
</membership>
```

Les memberships peuvent s'utiliser dans le code-behind, en utilisant la classe membership qui est composée de méthodes statiques, ne nécessitant aucune instance de classe.
Voici une liste non-exhaustive de méthode:

Création d'un utilisateur (deux manières):

```
Membership.CreateUser(username, password);
OU
MembershipCreateStatus status;
MembershipUser newuser = Membership.CreateUser(stringLogin,
stringPassword, stringMail, stringQuestion, stringReponse, true,
out status);
```

Suppréssion un utilisateur:

```
Membership.DeleteUser(username);
```

Validité d'un utilisateur:

```
Membership.ValidateUser(username, password);
```

Chercher un utilisateur par son mail:

```
Membership.FindUsersByEmail(email);
```

Afficher tous les utilisateurs:

```
Membership.GetAllUsers();
```

Il vous suffit de tapper "Membership" dans le code-behind (*.aspx.cs), vous pourrez apercevoir par auto-complétion toutes les méthodes de cette classe. Bien entendu toutes ces méthodes peuvent être générer automatiquement en utilisant les contrôles de Login, qui consiste simplement à du Drag and Drop.

Voici un exemple simple d'un formulaire d'inscription en utilisant la class membership:

```
Login: <asp:TextBox ID="TBox_Login"
                    runat="server"></asp:TextBox><br />
Password:<asp:TextBox ID="TBox_Pwd"
                    runat="server"></asp:TextBox><br />
<br />
<asp:Button       ID="Bt_Register"       runat="server"
OnClick="Bt_Register_Click" Text="S'inscrire" /></div>
```

```
protected    sub    Bt_Register_Click(object    sender,
EventArgs e)

                  Membership.CreateUser(TBox_Login.Text,
TBox_Pwd.Text)
    End sub
```

6.3.3. Rôles

Les rôles consistent à regrouper les utilisateurs par groupes, par exemples les administrateurs dans le rôle "admin", les utilisateurs inscrits dans le rôle "user". L'utilité des rôles est de pouvoir contrôler un grand nombre d'utilisateur par l'intermédiaire d'un nom: le nom du rôle. Il vous faut cependant, activer les rôles, soit dans le WSAT, soit dans le web.confg comme ci-dessous.

```
<roleManager                                    enabled="true"
defaultProvider="RoleManagerSqlProvider">
<providers>
<add      connectionStringName="votre_ConnectionString"
applicationName="royal_fleur"
name="RoleManagerSqlProvider"
type="System.Web.Security.SqlRoleProvider" />
</providers>
</roleManager>
```

Comme pour les memberships, ils existent des classes pouvant utiliser les rôles, la principale classe est "roles".
Voic quelques exemples de méthodes pouvant être utiliser:

Création d'un rôle:
Roles.CreateRole(nom);

Suppréssion d'un rôle:
Roles.DeleteRole(nom);

Ajout d'un utilisateur à un rôle:
Roles.AddUserToRole(username, rolename);

Suppréssion d'un utilisateur d'un rôle:
Roles.RemoveUserFromRole(username, rolename);

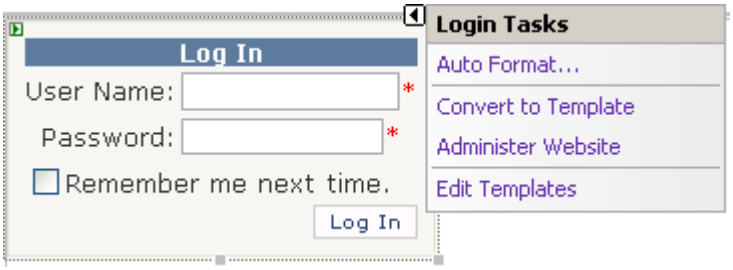
Les rôles, ainsi que les memberships, peuvent être configurer avec le WSAT.

6.4. *Les contrôles de login*

Dans le précédent chapitre, nous avons vu les différentes méthodes possibles pour la classe membership, cependant toutes ces méthodes peuvent être gérer automatiquement par les contrôles de login. Tous les templates des contrôles ci-dessous sont modifiables, y compris par l'intermédiaire d'un thème ou d'un fichier skin.

6.4.1. Login

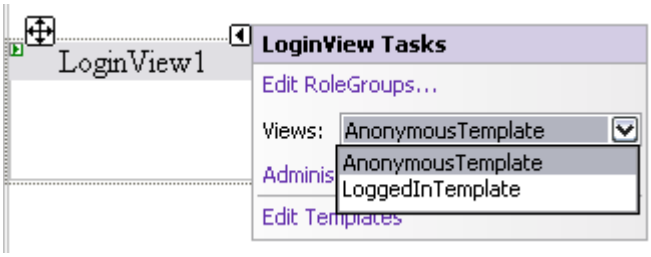
Le Login permet tout simplement à un utilisateur de se connecter en indiquant son login et son password, il vous suffit simplement de faire du Drag an Drop, prendre l'objet login et le déposer sur une page.



Vous avez la possibilité de modifier le template, soit en utilisant les templates intégrès au contrôle en cliquant sur "Auto Format", soit vous éditer vous même le template en cliquant sur "Convert To Template", cela convertira le contrôle en tableau HTML.

6.4.2. LoginView

Le loginView permet de gérer des templates différents:
- le template pour les utilisateurs anonymes
- le template pour les utilisateurs connectés

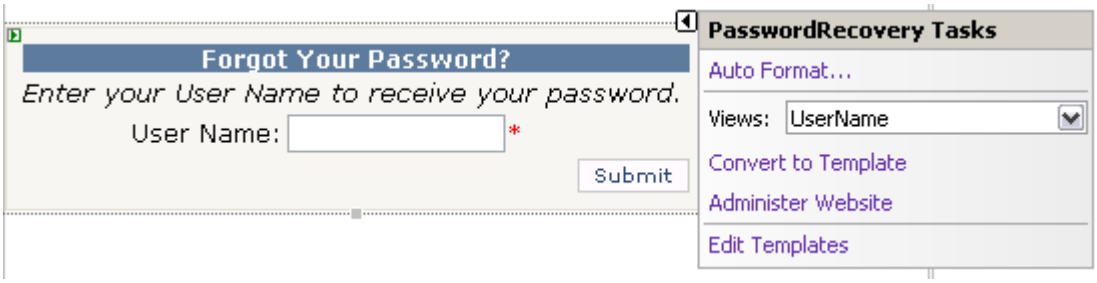



```
        <asp:LoginView ID="LoginView1" runat="server">
          <LoggedInTemplate>
                        Bonjour<asp:LoginName   ID="LoginName1"
        runat="server" />
          </LoggedInTemplate>
          <AnonymousTemplate>
                                    <asp:Login      ID="Login2"
```

```
        runat="server"></asp:Login>
          </AnonymousTemplate>
        </asp:LoginView>
```

Dans l'exemple ci-dessus, un utilisateur anonyme verrait le contrôle Login, alors qu'un utilisateur connecté verra "Bonjour [Login]".

6.4.3. PasswordRecovery




```
        <asp:PasswordRecovery              ID="PasswordRecovery1"
        runat="server">
                        <MailDefinition     From="votre@mail.fr"
        Subject="sujet">
            </MailDefinition>
        </asp:PasswordRecovery>
```

Attention, la configuration mail est obligatoire pour que ce contrôle fonctionne, en effet ce contrôle envoit un mail à l'utilisateur ayant rentrer son pseudo, sa question de sécurité ainsi que la réponse de sécurité. De plus, il faut configurer le mail dans le fichier de configuration comme ci-dessous:

```
        <system.net>
                    <mailSettings>
                          <smtp>
                                <network host="localhost" />
                          </smtp>
                    </mailSettings>
             </system.net>
```

Dans le cas ci-dessus, il s'agit d'un serveur SMTP en local.

6.4.4. LoginStatus

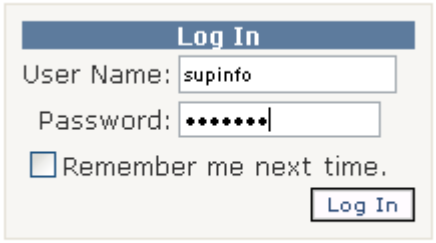
Le LoginStatus est affiche tout simplement un lien "login" si l'utilisateur est anonyme ou un lien "LogOut" si l'utilisateur est connecté.

(Voir l'exemple ci-dessous).

6.4.5. LoginName

LoginName, vu dans l'exemple du LoginView, permet de récupérer le Login de l'utilisateur connecté, pratique pour personnaliser un message avec le login pour chaque utilisateur.

```
<asp:LoginView ID="LoginView1" runat="server">
   <AnonymousTemplate>
         <asp:Login ID="Login1" runat="server"
              DestinationPageUrl="~/login.aspx">
         </asp:Login>
         <asp:LoginStatus ID="LoginStatus1"
                          LogoutPageUrl="~/default.aspx"
runat="server" />
    </AnonymousTemplate>
    <LoggedInTemplate>
        Bonjour <asp:LoginName ID="LoginName1"
                       runat="server" />!
                  <asp:LoginStatus   ID="LoginStatus1"
runat="server" />
    </LoggedInTemplate>
</asp:LoginView>
```

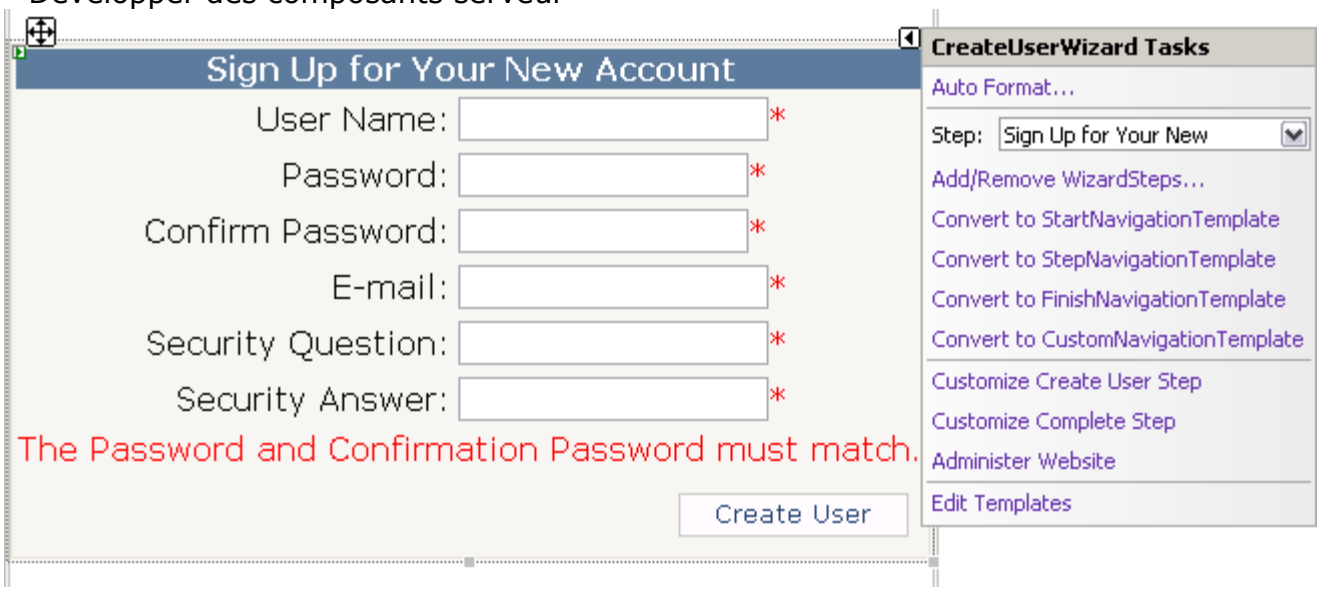


Login

Bonjour supinfo! Logout

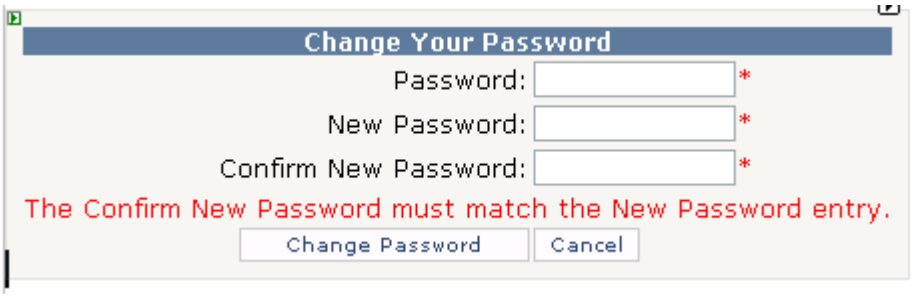
6.4.6. CreateUserWizard

Ce contrôle permet à un utilisateur de s'inscrire en rentrant quelques informations, cela ne nécessite aucune configuration, mis à part les memberships dans le web.config.



6.4.7. ChangePassword

Comme le nom l'indique, cela permet tout simplement de changer son password.



6.5. *Les différents fournisseurs d'authentification*

Le choix des fournisseurs se situe au niveau de la balise "authentification".

En voici un exemple:

```
<authentication mode="Forms">
<forms name=".ASPXUSERAUTH"
       defaultUrl="default.aspx"
       loginUrl="login.aspx"
       requireSSL="false"
       protection="All">
</forms>
</authentication>
```

6.5.1. Forms

L'authentification par formulaire est la plus répandue et la plus utilisées. Elle consiste en l'utilisation des cookies. Lorsqu'un utilisateur est connecté, un cookie est crée chez le client.
Ce fournisseur implique souvent l'utilisation des contrôles de sécurité (login).

6.5.2. Passport

Le fournisseur passport est spécifique à Microsoft, et requiert un compte passport (MSN etc...) qui sont centralisé chez Microsoft.

6.5.3. Windows

Ce fournisseur implique la présence de IIS sur le serveur, qui effectuera lui même les authentifications de chaque utilisateur.

6.5.4. None

Ce fournisseur correspond tout simplement à une authentification personnalisée, c'est-à-dire que le développeur devra lui même le développer, ce qui demanderait un surplus de travail.

6.6. *Appliquer des autorisations*

6.6.1. **Les balises**

Les autorisations permettent de restreindre l 'accès aux utilisateurs à certaines partie du site ou dossier. Par exemple interdire l'accès aux anonyme dans l'administration, interdire l'accès aux utilisateurs inscrit à un dossir d'image etc...
Tout cela se configure par l'intermédiaire du fichier web.config, qui comme vous le voyez est très utile.

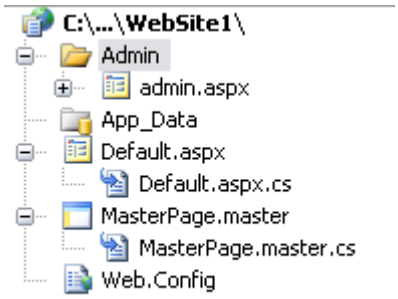
Voici la balise à mettre entre :

```
<authorization>
<deny users="?"/>
</< FONT><authorization>
```

- * signifie "tout le monde"
- ? signifie "utilisateurs anonyme"
- Si cette balise est entre cela s'applique au site entier (projet entier), dans notre cas, personne n'a accès au site.
- Pour autoriser, il suffit de remplacer deny par allow.
- Pour autoriser un rôle, et non un utilisateur, il suffit de remplacer users par roles.

6.6.2. Exemples d'autorisations

Voici une arborescence assez simple:



La meilleur politique d'autorisation est d'interdire, tout d'abord, l'accès au site entier, donc:

```
<authorization>
<deny users="*"/>
</< FONT><authorization>
```

Ensuite, il faut autoriser un par un les pages/dossiers.
Voici les balises à insérer (en dehors de system.web!):

```
<location path="default.aspx">  la page default.aspx
<system.web>
<authorization>
<allow users="*" />
</authorization>
</system.web>
</location>

<location path="MasterPage.Master">  la Masterpage
<system.web>
<authorization>
<allow users="*" />
</authorization>
</system.web>
</location>

<location path="admin">   Le dossier admin
<system.web>
<authorization>
<deny users="?">
<allow roles="admin" />
</authorization>
</system.web>
</location>
```

En général, il faut mettre en premier les interdictions (deny), puis les autorisations (allow).

Les autorisations s'avèrent très utiles et compliqué pour des sites complexes, c'est pourquoi cela demande une arborescence de fichier bien ordonné et réfléchi, établir à l'avance les rôles, les utilisateurs, les différents accès au site.
Comme vu ci-dessus, la meilleure politique d'autorisation est de "tout interdire" puis autoriser un par un.

6.7. *WSAT - Web Site Administration Tool*

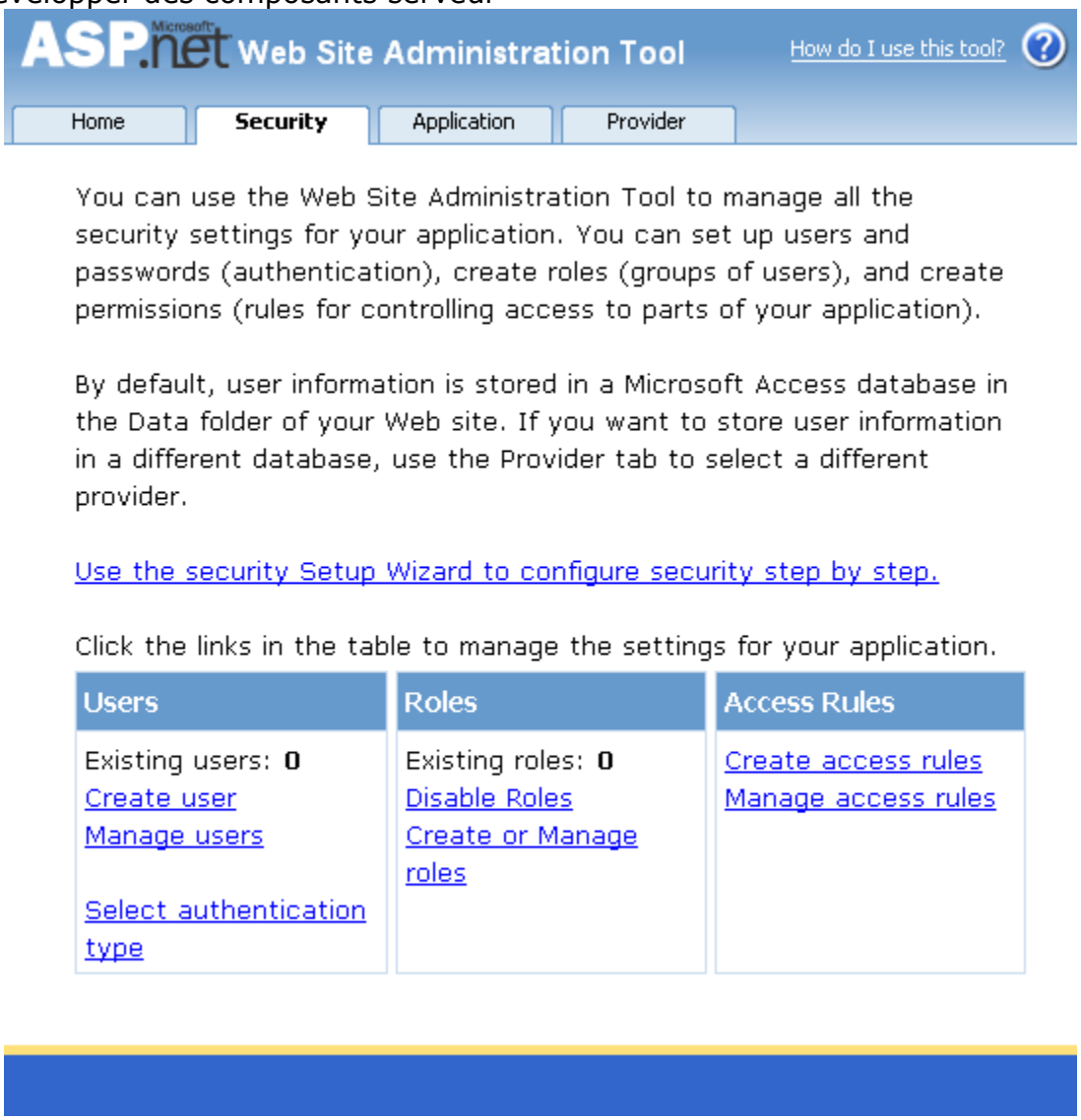
Le WSAT est une interface d'administration simplifié du fichier de configuration, permettant de configurer votre application/ site. Le WSAT est composé de 3 sections:

- Security
- Application
- Provider

Vous avez aussi la possiblité de modifier les profiles par l'intermédiaire du WSAT.

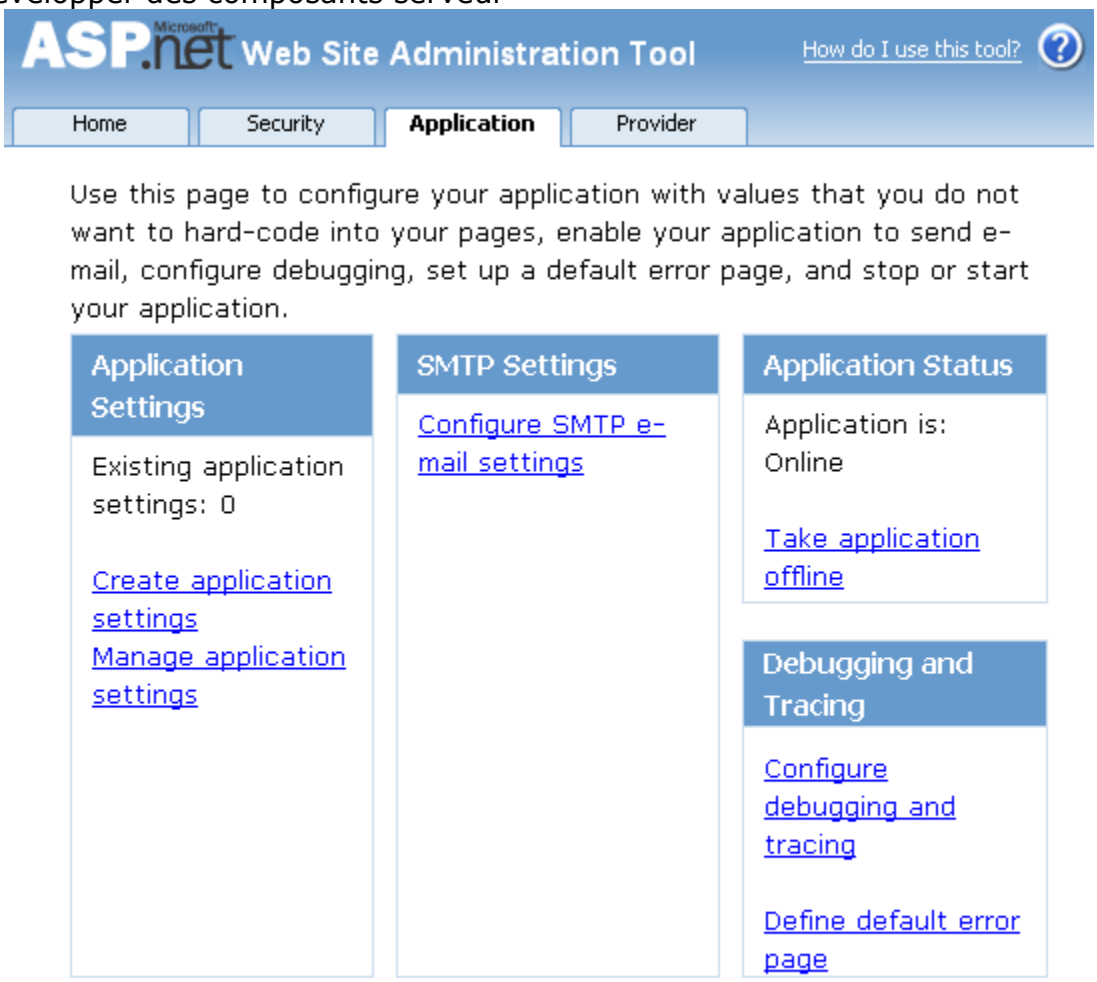
6.7.1. Security

La partie sécurité concerne tout ce que l'on a vu jusque là. Grâce à cette administration, vous pourrez ajouter/modifier/supprimer des utilisateurs, ajouter/modifier/supprimer des rôles, ainsi que les autorisations.



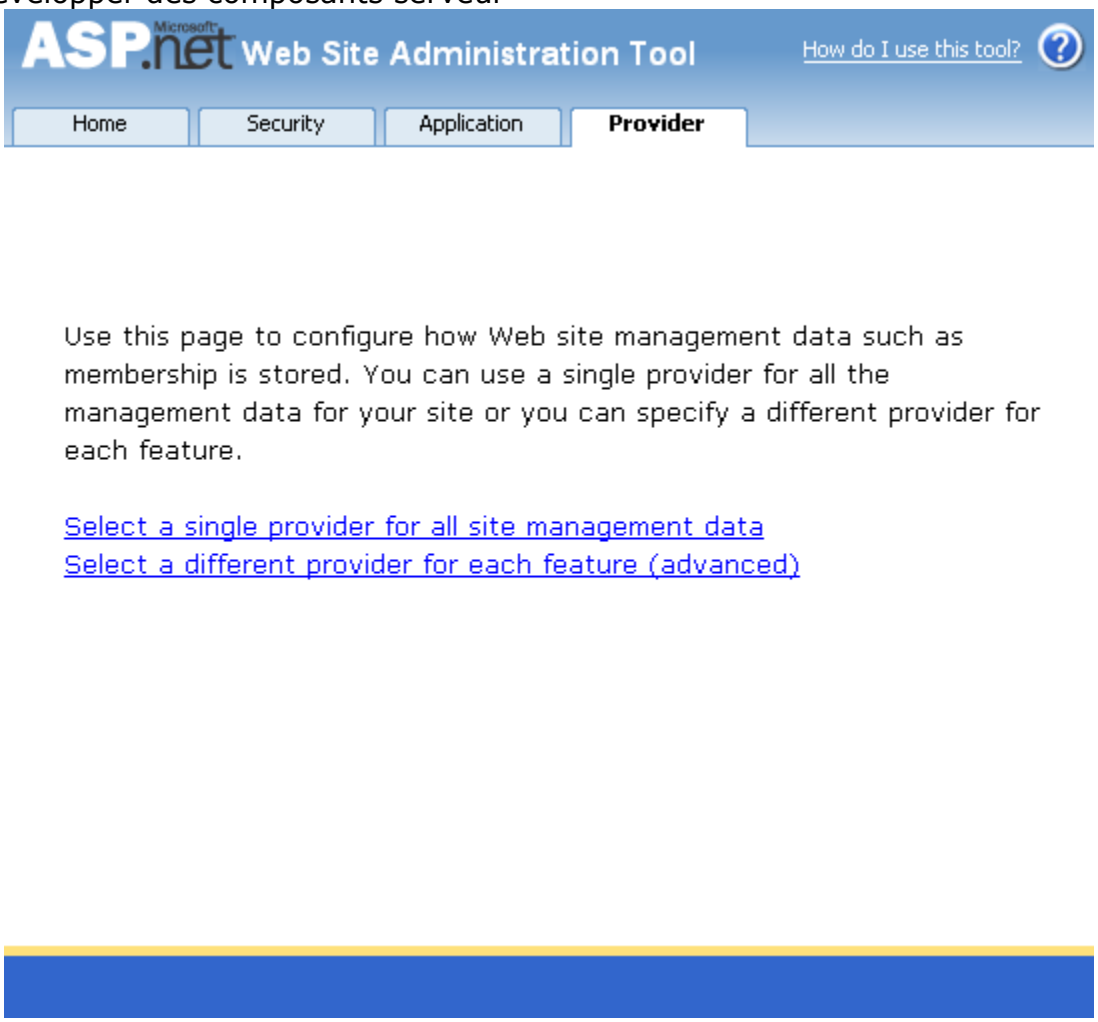
6.7.2. Application

Cette partie concerne la configuration de l'application, notamment la configuration mail (smtp), le debug et tracing avec cassini (serveur web inclus) ou IIS.



6.7.3. Provider

Vous avez la possiblité de gérer vos providers dans cette partie de l'administration. Bien entendu cela ne permet pas de créer une base de donnée. Ici, vous pouvez soit, choisir un provider pour tous les composants, soit un par un, par exemple vous aurez le choix de choisir un provider pour les memberships, roles, profiles. Si vous ajoutez un provider par l'intermédiaire du fichier web.config, cela apparaîtra dans le WSAT aussi.



7.Web Parts

7.1. Introduction aux WebParts

On voit apparaître de plus en plus souvent sur des sites, des portails, un système de zone de fenêtrage tel que sur MyMSN, sur Google et Yahoo : ce sont les WebParts.
Les WebParts représentent une nouvelle organisation et conception d'un site WEB dont le principe se base sur un regroupement de contrôle dans des zones modifiables dont on peut contrôler l'aspect en reduisant la zone, en fermant la zone, en échangeant des contrôles d'une zone à un autre dynamiquement. Cette fonctionnalité se base ainsi sur des zones omposés de WebPart (contrôle). Les utilisateurs peuvent contrôler l'affichage d'une page, en choisissant les contrôles à afficher, la position des contrôles, de changer l'apparence de ces contrôles, tout cela dynamiquement.
L'utilisation des WebParts est associée aux memberships et la personnalisation qui permet à l'utilisateur de contrôler tous les aspects des WebParts selon les autorisations appliquées.

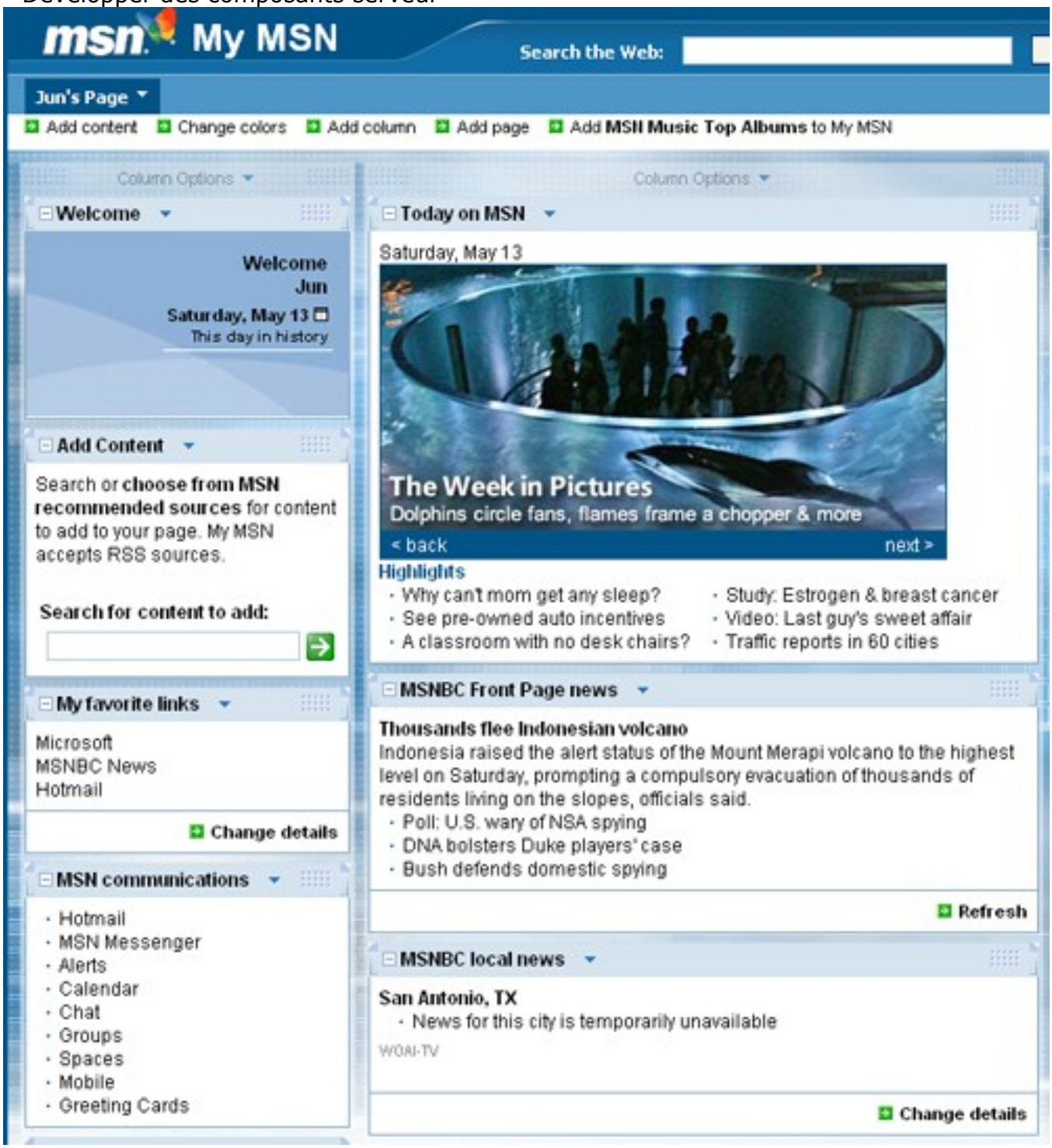


Fig 3.1 MyMSN

7.2. Les différentes zones de WebParts

Propriétés	Description
WebPartZone	Zone par défaut et visible permettant l'insertion de WebPart, déplacer et déposer possible entre deux WebPartZone.
EditorZone	Zone d'édition pour interagir avec les WebPartZone, composé de WebParts Editor. Zone invisible par défaut

CatalogZone	Zone contenant les WebParts invisible par défault, des WebPart que l'utilisateur à fermer Zone invisible par défault
ConnectionsZone	Zone permettant d'éditer les liens entre les WebParts. Zone invisible par défault

Toutes ces zones sont contrôlés par un WebPartManager, unique par page.

7.3. Création des WebParts

Les WebPart et WebPartZone sont contrôlés par le WebPartManager, il faut donc le déposer sur la page pour pouvoir utiliser les WebParts. Ensuite, il faudra placer des WebPartZones qui en général sont placés dans des tableaux HTML (ou div).

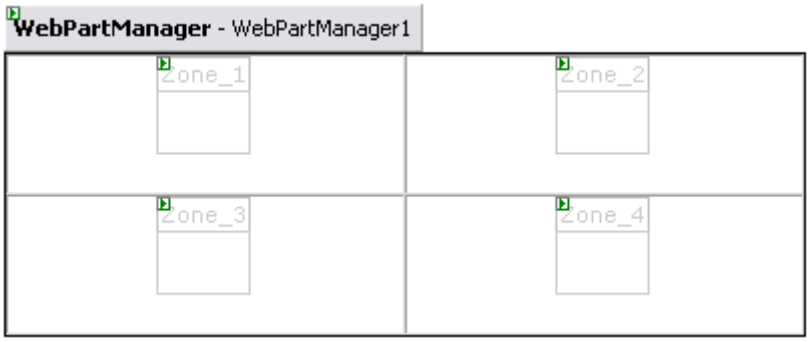


Fig 3.2 Exemple création des WebPartZone (mode design)

Il faut maintenant remplir les WebPartZone par des contrôles, par exemple déposer un Label, une TextBox, … .

Voici un exemple qui servira tout au long de l'article :

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:WebPartManager ID="WebPartManager1" runat="server">
        </asp:WebPartManager>
             <table style="width:400px; border: solid 1px black;
vertical-align: middle; text-align:center"
                 border="1" cellpadding="0" cellspacing="0">
            <tr>
            <td style="width:200px">
                <asp:WebPartZone ID="Zone_3" runat="server">
                    <ZoneTemplate>
                                <asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>
                    </ZoneTemplate>
                </asp:WebPartZone>

            </td>
            <td style="width:200px">
```

```
                <asp:WebPartZone ID="Zone_3" runat="server">
                    <ZoneTemplate>
                                                <asp:TextBox  ID="TextBox1"
runat="server"></asp:TextBox>
                    </ZoneTemplate>
                </asp:WebPartZone>

            </td>
            </tr>
            <tr>
            <td style="width:200px">
                <asp:WebPartZone ID="Zone_3" runat="server">
                    <ZoneTemplate>
                                <asp:Button ID="Button1" runat="server"
Text="Button" />
                    </ZoneTemplate>
                </asp:WebPartZone>
            </td>
            <td style="width:200px">
                <asp:WebPartZone ID="Zone_4" runat="server">
                    <ZoneTemplate>
                                  <asp:Label ID="Label2" runat="server"
Text="Label"></asp:Label>
                    </ZoneTemplate>
                </asp:WebPartZone>
            </td>
            </tr>
        </table>
        <br />
    </form>
</body>
</html>
```

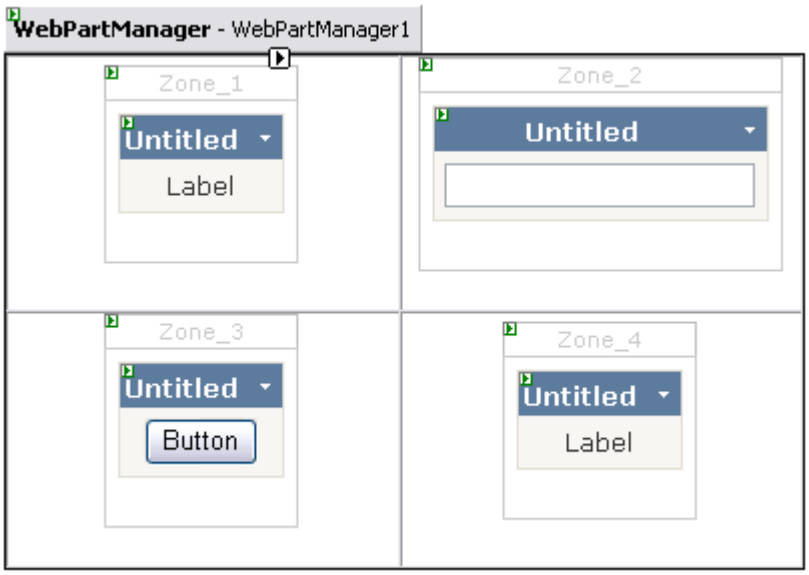


Fig 3.3 Exemple de création de WebParts (mode design)

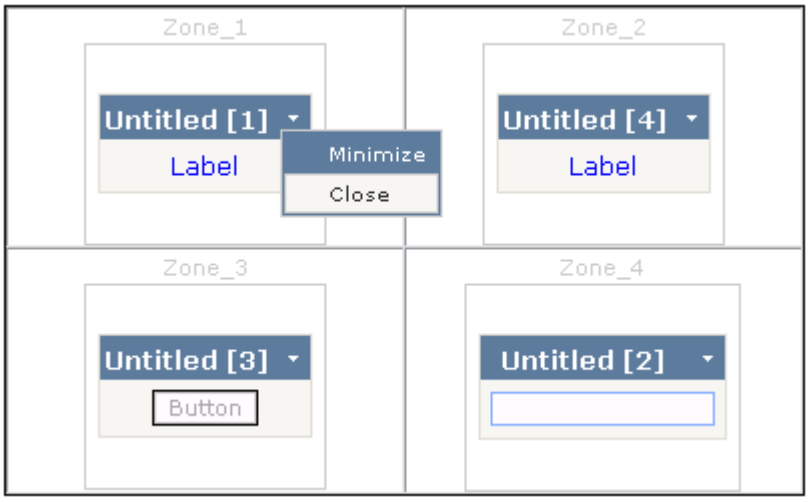


Fig 3.4 Exemple de création de WebParts (mode Browser)

Attention, vous devez obligatoirement être connectés (avec l'utilisation des Membership) pour voir les WebParts. Par défaut, vous êtes dans le mode « Design » permettant seulement de réduire ou fermer les WebParts.

7.4. Formatter des WebParts

Par défaut, les WebParts n'ont aucun style d'où l'apparence pauvre. Pour cela, vous avez la possibilité d'appliquer des styles pré-définies :

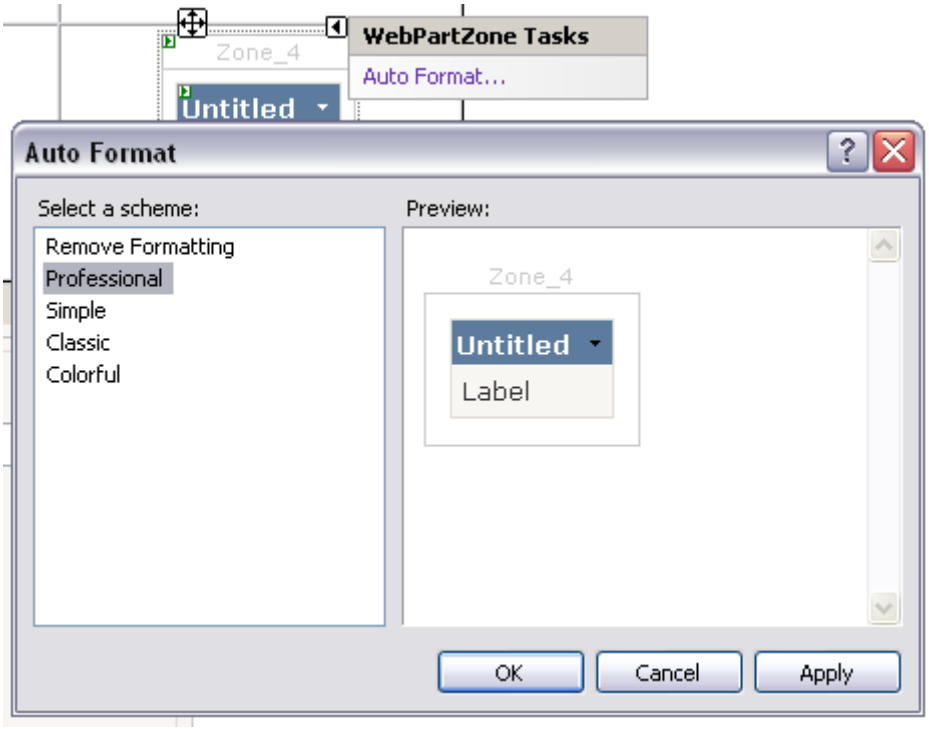


Fig 3.5 Formats proposés par défaut

L'autre manière consiste à appliquer un style définie par l'utilisateur même, grâce aux éléments de style d'une WebPartZone.

Propriétés	Description
PartChromeStyle	Style de la zone vide dans une

	WebPartZone
MenuLabelHoverStyle	Style du Label lorsque la souris survole le Label
EmptyZoneTextStyle	Style de la zone de texte vide
HeaderStyle	Style de l'en-tête
MenuVerbStyle	Style du menu d'un WebParts
PartStyle	Style des WebParts dans la zone
MenuLabelStyle	Style des Labels dans le menu
MenuPopupStyle	Style du menu en haut à droite
PartTitleStyle	Style de titre du WebParts

Voici un exemple de style pour les WebPartZone :

```
<asp:WebPartZone ID="Zone_2" runat="server" BorderColor="#CCCCCC"
Font-Names="Verdana"
                    Padding="6">
                                <PartChromeStyle BackColor="#F7F6F3"
BorderColor="#E2DED6" Font-Names="Verdana" ForeColor="White" />
                    <MenuLabelHoverStyle ForeColor="#E2DED6" />
                    <EmptyZoneTextStyle Font-Size="0.8em" />
                    <MenuLabelStyle ForeColor="White" />
                            <MenuVerbHoverStyle BackColor="#F7F6F3"
BorderColor="#CCCCCC" BorderStyle="Solid"
                        BorderWidth="1px" ForeColor="#333333" />
                                        <HeaderStyle Font-Size="0.7em"
ForeColor="#CCCCCC" HorizontalAlign="Center" />
                                <MenuVerbStyle BorderColor="#5D7B9D"
BorderStyle="Solid" BorderWidth="1px" ForeColor="White" />
                                        <PartStyle Font-Size="0.8em"
ForeColor="#333333" />
                         <TitleBarVerbStyle Font-Size="0.6em" Font-
Underline="False" ForeColor="White" />
                                <MenuPopupStyle BackColor="#5D7B9D"
BorderColor="#CCCCCC" BorderWidth="1px" Font-Names="Verdana"
                        Font-Size="0.6em" />
                          <PartTitleStyle BackColor="#5D7B9D" Font-
Bold="True" Font-Size="0.8em" ForeColor="White" />
                    <ZoneTemplate>
                                            <asp:TextBox ID="TextBox1"
runat="server"></asp:TextBox>
                    </ZoneTemplate>
</asp:WebPartZone>
```

7.5. Changement de mode

Dans le mode par défaut « Browse Mode », vous avez seulement la possibilité de minimier la zone ou la fermer. Pour pouvoir éditer les WebParts, changer certaines propriétés, récuperer des WebParts, il faut changer de mode. Pour cela, il faudra implémenter des fonctions permettant le changement de mode. Nous allons prendre un exemple avec une DropDownList composé des 5 différents mode (Browse Mode, Design, Catalog Mode, Edit Mode, Connect Mode).

```
<asp:DropDownList ID="DisplayModeDropdown"
    runat="server"
    AutoPostBack="true"
    EnableViewState="false"
    OnSelectedIndexChanged=
```

```
            "DisplayModeDropdown_SelectedIndexChanged" />
```

```
WebPartManager _webmanager; // on crée un objet WebPartManager

    void Page_Init(object sender, EventArgs e)
    {
        Page.InitComplete += new EventHandler(InitComplete);
    }

    protected void InitComplete(object sender, System.EventArgs e)
    {
                                                    _webmanager      =
WebPartManager.GetCurrentWebPartManager(Page);

                                          String     browseModeName     =
WebPartManager.BrowseDisplayMode.Name;

        // remplit la dropdownlist avec les modes disponibles
                             foreach    (WebPartDisplayMode    mode    in
_manager.SupportedDisplayModes)
        {
            String modeName = mode.Name;
            // vérification de la validité d'un mode
            if (mode.IsEnabled(_webmanager))
            {
                  ListItem item = new ListItem(modeName + " Mode",
modeName);
                DisplayModeDropdown.Items.Add(item);
            }
        }

    }

    // Changement de page selon le mode sélectionné
     protected void DisplayModeDropdown_SelectedIndexChanged(object
sender,
       EventArgs e)
    {
        String selectedMode = DisplayModeDropdown.SelectedValue;

                                       WebPartDisplayMode     mode     =
_manager.SupportedDisplayModes[selectedMode];
        if (mode != null)
            _webmanager.DisplayMode = mode;

    }

    protected void Page_PreRender(object sender, EventArgs e)
    {
                                  DisplayModeDropdown.SelectedValue    =
_webmanager.DisplayMode.Name;
```

}

- La fonction Page_Init permet d'initialiser les contrôles pendant le chargement de la page et fait appel à la fonction InitComplete.

- La fonction InitComplete récupère le WebPartManager de la page concernée, étant donné qu'il est unique il existe un et un seul WebPartManager. Il stock le WebPartManager dans l'objet _webmanager. Ensuite, une boucle permet de remplir la DropDownList des modes d'affichage selon leur disponibilité. Par exemple si un Catalog Zone est disponible, on affichera le mode « Catalog Mode » dans la DropDownList, dans le cas contraire il ne sera pas affiché.

- La fonction DisplayModeDropdown est appelée suite au changement d'index de la DropDownList et permet le changement de page selon le mode sélectionné.

- La fonction Page_PreRender permet tout simplement de mettre à jour l'objet _webmanager en fonction du mode sélectionné dans la DropDownList.

Etant donné que ces différents modes ne sont pas visibles par défaut, et nécessitent une connection, il faudra rajouté le contrôle de Login pour permettre à l'utilisateur de se connecter et d'accéder ensuite aux différents modes.

Voici l'apparence des WebParts et mode disponible par défaut :

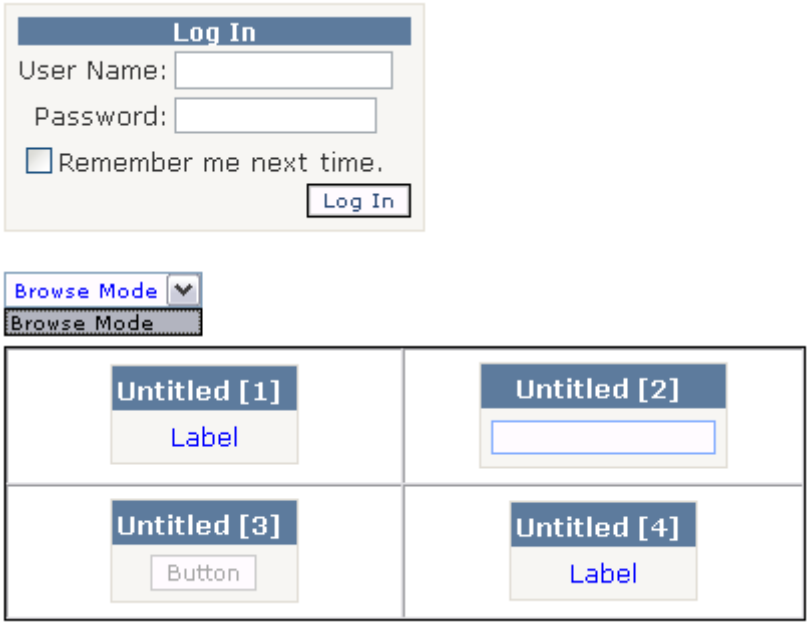


Fig 3.6 Mode par défaut avant connection

Apparence des WebParts et mode disponible après connexion :

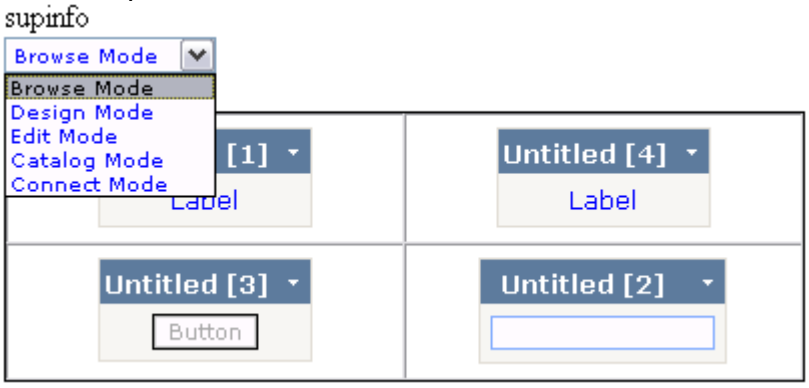


Fig 3.7 Mode disponible après connexion

7.6. CatalogZone

Un catalogue contient les WebParts qui ne sont pas utilisés ou qui ne doivent pas apparaître par défaut. Il contient aussi les WebParts que l'utilisateur à fermer. L'utilisateur connecté pourra ainsi construire sa page de WebParts grâce à un catalogue que l'administrateur aura remplit de WebPart auparavant. Qui dit catalogue, dit possibilité de choisir à plusieurs reprise le même WebPart et l'insérer dans une zone, par exemple si vous choisissez un Label du catalogue, vous aurez la possibilité de l'insérer autant de fois que vous le souhaitez, étant donné qu'il n'est pas unique.
Un catalogue est donc essentiellement une collection de WebParts disponible, d'où le nom de « Catalog ».
Pour construire son catalogue, il faut tout d'abord insérer une zone catalogue dans lequel on pourra ajouter trois type d'éléments. Un catalogue de page (« Page Catalog ») qui sera composé des WebParts fermés par l'utilisateur, une déclaration de catalogue (« DeclarativeCatalogPart ») qui permet à l'administrateur de l'application WEB de remplir le catalogue de WebParts, et enfin un importateur de WebPart (« Import Catalog Part ») permettant à l'utilisateur d'importer des WebParts depuis sa machine.

Voici le code source d'un CatalogZone sans aucun style :

```
<asp:CatalogZone ID="CatalogZone1" runat="server">
            <ZoneTemplate>
                        <asp:PageCatalogPart  ID="PageCatalogPart1"
runat="server" />
                                    <asp:DeclarativeCatalogPart
ID="DeclarativeCatalogPart1" runat="server">
                    <WebPartsTemplate>
                                <asp:Label  ID="Label3"  runat="server"
Text="Label"></asp:Label>
                        <br />
                                    <asp:TextBox  ID="TextBox2"
runat="server"></asp:TextBox>
                    </WebPartsTemplate>
                </asp:DeclarativeCatalogPart>
                    <asp:ImportCatalogPart  ID="ImportCatalogPart1"
runat="server" />
            </ZoneTemplate>
</asp:CatalogZone>
```

Le PageCatalogPart et ImportCatalogPart ne sont pas éditables, il suffit tout simplement de les déposer dans le CatalogZone. Pour ce qui est de la DeclarativeCatalogPart, il faut éditer le template pour pouvoir insérer des WebParts pour construire le catalogue.

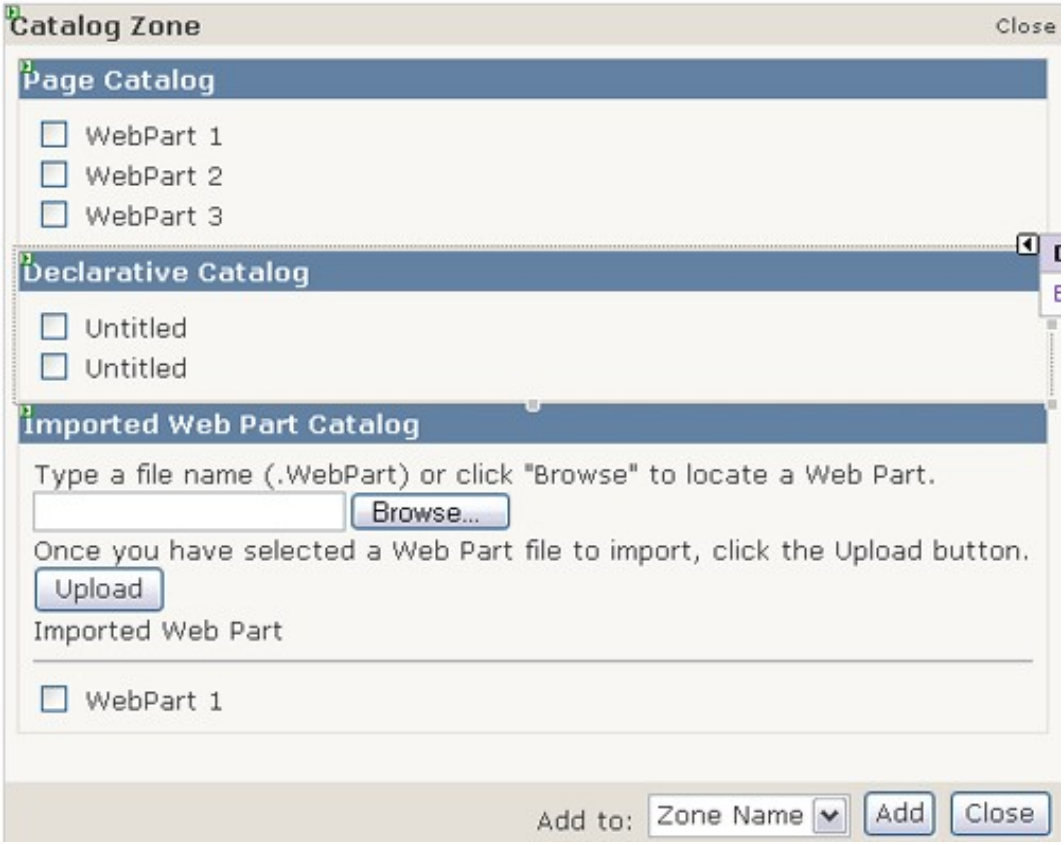


Fig 3.8 CatalogZone

Le PageCatalog ci-dessous contient quatre WebParts fermés par l'utilisateur. Pour récupérer ces WebParts, il suffit de cocher les WebPart à récupérer puis choisir la zone puis cliquez sur le bouton « Add ».

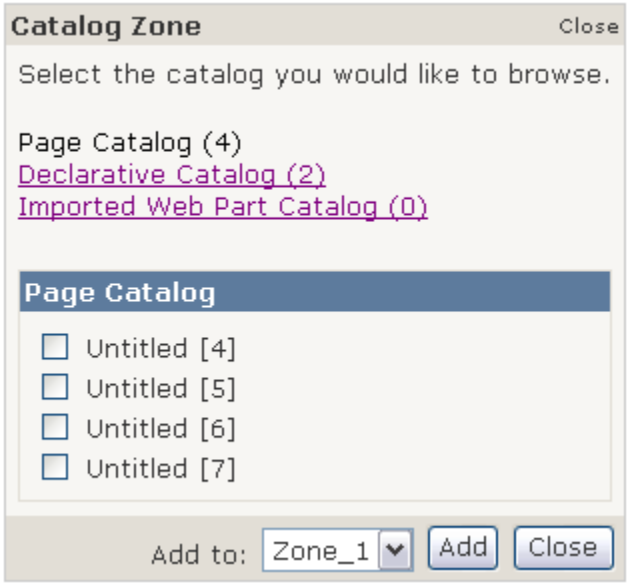


Fig 3.9 Page Catalog

La DeclarativeCatalog comme ci-dessous contient deux WebParts. Pour les ajouter dans une nouvelle zone il suffit de cocher les WebParts concernées et la zone.

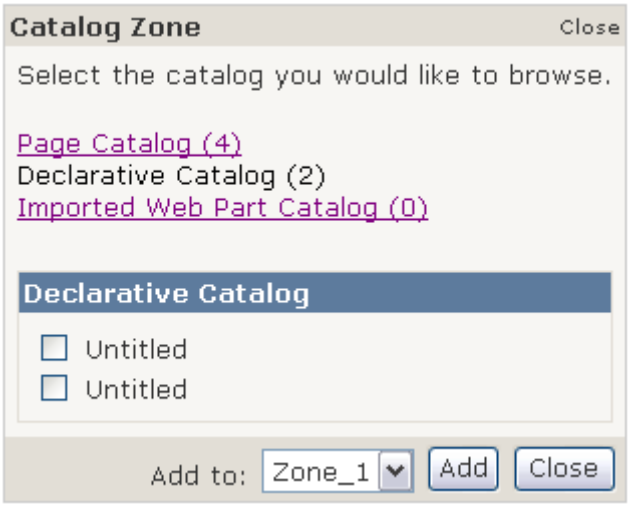


Fig 3.10 Declarative Catalog

L'Import Catalog Part comme ci-dessous permet d'importer des WebPart, pour cela il faut spécifier le chemin et la zone.

Fig 3.11 Import WebParts

7.7. EditorZone

Les WebParts sont modifiables dans la zone d'édition « Editor Zone » par l'intermédiaire de quatre types d'élément de modification :

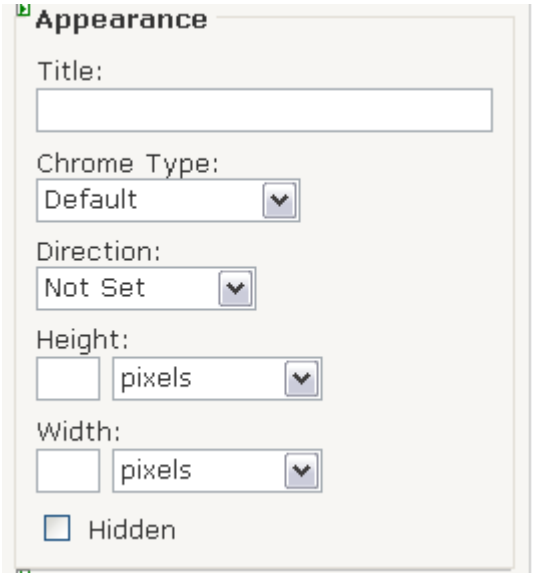
- ApparenceEditorPart
- BehaviorEditorPart
- LayoutEditorPart
- PropertyGridEditorPart

Voici les sources d'un EditorZone sans style appliqué :

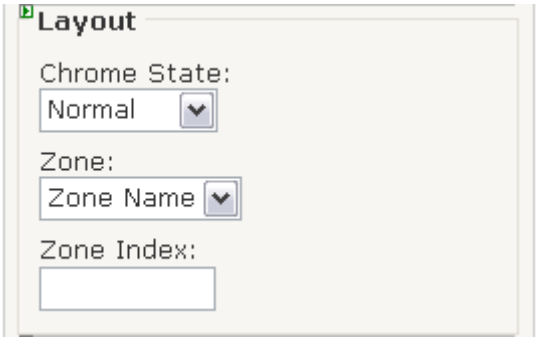
```
<asp:EditorZone ID="EditorZone1" runat="server">
        <ZoneTemplate>
          <asp:AppearanceEditorPart runat="server"
            ID="Appearance1">
          </asp:AppearanceEditorPart>
          <asp:LayoutEditorPart runat="server" ID="Layout1">
          </asp:LayoutEditorPart>
                  <asp:BehaviorEditorPart  ID="BehaviorEditorPart1"
runat="server" />
                                          <asp:PropertyGridEditorPart
ID="PropertyGridEditorPart1" runat="server" />
        </ZoneTemplate>
</asp:EditorZone>
```

Pour accéder aux différents menus de modification, il faut être connecté, être dans le mode « Edit Mode » et choisir le WebPart à modifier en accédant à son menu (en haut à droit de chaque WebPart) puis choisir Edit.

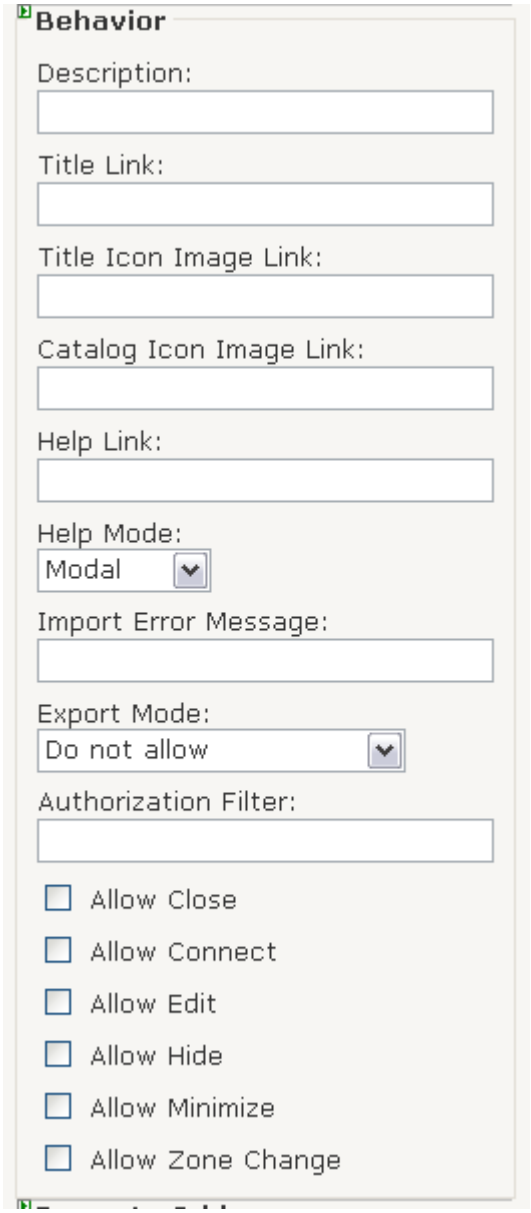
ApparenceEditorPart va vous permettre de modifier l'apparence des WebParts notamment le titre, la taille, la visiblité.



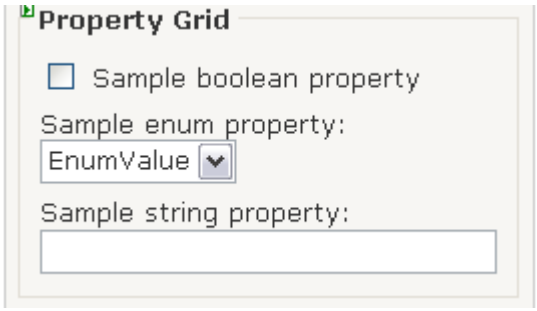
Le LayoutEditorPart permet principalement de modifier la zone conteneur.



Le BehaviorEditorPart permet de contrôler le comportement des WebParts, de pouvoir fermer, diminuer, modifier, cacher les WebParts, de modifier leur description, les liens.



Le PropertyGridEditorPart permet d'éditer des propriétés dans le code source ayant comme attribut « WebBrowsable ».



7.8. Conclusion

Les WebParts permettent ainsi de créer des portails dynamique et facilement personnalisable. Cette nouvelle fonctionnalité, de plus en plus répandue sur de grand site tel que Google et Yahoo, permet à l'utilisateur

de construire sa page, d'ajouter ou d'enlever des fenêtres selon son choix, de définir l'apparence et la disposition de ses fenêtre. L'utilisateur a un contrôle quasi-total de la page Web. Il est cependant dommage que les WebParts intégrés par défaut à l'ASP.NET 2.0 ne soit pas un contrôle coté client, plus exactement avec l'intégration d'un CallBack sans aucun rafraîchissement de la page. Il est aujourd'hui possible de résoudre ce problème en implémentant la version AJAX de Microsoft pour l'ASP.NET, ATLAS.

8. Conclusion

Les nouveautés du langage ASP.NET 2.0 de Microsoft apportent non seulement une simplicité au développeur dans son travail, mais aussi des nouvelles interfaces WEB, des nouvelles fonctionnalités pour les utilisateurs et clients. Ce langage innovant apporte une nouvelle conception des sites Internet de « nouvelle génération », de plus les applications WEB développées en ASP.NET 2.0 seront combinées avec la version AJAX de Microsoft, ATLAS pour rendre la navigation fluide et sans aucun rafraîchissement. D'autres nouveautés n'ont pas été traitées dans cette article sont aussi important, comme les nouveaux contrôles de Login et de données (GridView, DetailsView, …), et la sécurité des applications.