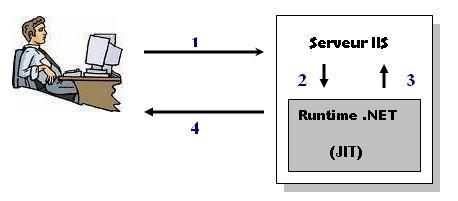
Techniques de développement informatique

Développer des composants serveur

# Introduction ASP.NET 2.0

## 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 (Internet Information Services).  
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 stocke 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.
* Au fait, qu'est-ce qu'une application Web?
* Il s'agit de l'ensemble des fichiers, pages, gestionnaires, modules et code situés dans un répertoire virtuel et ses sous-répertoires sur un serveur Web donné.

## Environnement de développement

### 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 :

1. **Sites Web IIS locaux**

Les sites Web IIS locaux s'exécutent à l'aide d'une copie d'IIS installée sur votre ordinateur. Lorsque vous créez un site Web IIS local, les pages et les dossiers de votre site sont stockés dans un sous-dossier du dossier IIS par défaut pour les sites Web, qui se trouve dans [lecteur]:\Inetpub\wwwroot. Visual Studio crée également la configuration IIS appropriée afin que le site Web soit reconnu par IIS comme une application.

1. **Sites Web de système de fichiers**

Dans un site Web de système de fichiers, vous pouvez créer et modifier des fichiers dans n'importe quel dossier, que ce soit sur votre ordinateur local ou dans un dossier sur un autre ordinateur auquel vous accédez via un partage réseau.

Il n'est pas obligatoire d'exécuter IIS sur votre ordinateur.

Au lieu de cela, vous pouvez tester des pages à l'aide du serveur de développement ASP.NET.

|  |  |
| --- | --- |
| **Type de site Web** | **Résumé** |
| Site Web IIS local | IIS est installé sur votre ordinateur local.  **Avantages :**   * Le site est accessible à partir d'autres ordinateurs. * Vous pouvez le tester avec les fonctionnalités IIS, par exemple l'authentification basée sur http.   **Inconvénients :**   * Vous devez posséder des droits d'administration pour créer ou déboguer un site Web IIS. * Un seul utilisateur peut déboguer un site Web IIS sur l'ordinateur à un moment donné. * Par défaut, l'accès à distance est activé pour un site Web IIS local. |
| Site Web de système de fichiers | Utilisez un site Web de système de fichiers lorsque vous souhaitez créer des pages Web sur votre ordinateur local ou sur un lecteur partagé alors qu'IIS n'est pas installé.   |  | | --- | | **RemarqueRemarque :** | | Vous pouvez créer un site Web de système de fichiers, puis créer ultérieurement un répertoire virtuel IIS qui pointe vers le dossier contenant vos pages. |   **Avantages :**   * L'accès au site peut se faire uniquement à partir de l'ordinateur local, ce qui réduit les failles de sécurité. * IIS ne doit pas obligatoirement être installé sur votre ordinateur. * Vous n'avez pas besoin de droits d'administration pour créer ou déboguer un site Web de système de fichiers local. * Si l'ordinateur est configuré pour autoriser des connexions Bureau à distance, plusieurs utilisateurs peuvent créer et déboguer des sites Web de système de fichiers locaux en même temps.   **Inconvénients :**   * Vous ne pouvez pas tester un site Web de système de fichiers avec les fonctionnalités IIS, par exemple l'authentification basée sur http. |

### 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\v2.0.50727\aspnet\_regiis.exe -i* ou xx est la version du Framework 2.0 présente sur votre ordinateur.

### ­Un EDI, c'est nécessaire ?

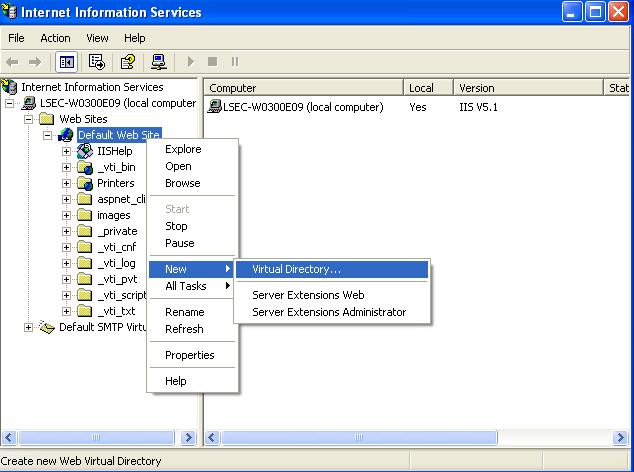
Nous avons tous l'habitude de travailler dans un environnement de développement intégré bien que cela ne soit pas toujours nécessaire mais plutôt bien pratique. Il en est de même avec le développement ASP.NET. Vous pouvez, comme pour des applications Winforms, écrire du code dans un éditeur de texte.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

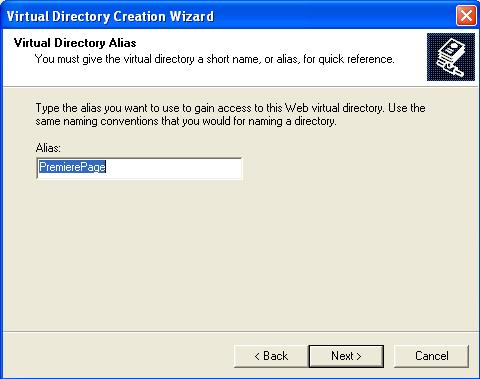
### Etape 1

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

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

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







Terminer votre création en laissant les paramètres par défaut.

### Etape 2

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

**<%@ Page Language="CS" %>**

**<html>**

**<body>**

**<h1>Bonjour</h1>**

**<br />**

**<h2>Nous sommes le <%= DateTime.Now.ToString() %>.</h2>**

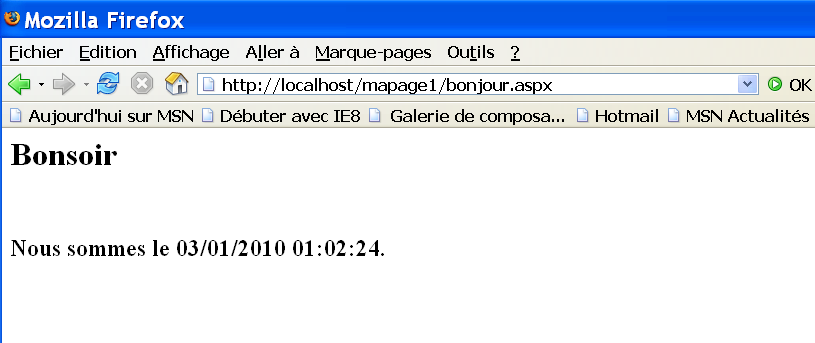
**</body>**

**</html>**

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

### Etape 3

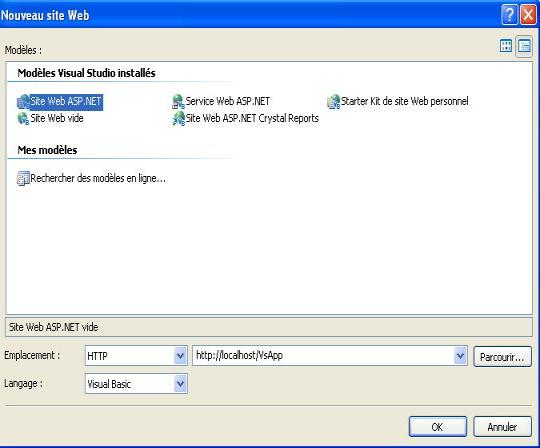
Exécutez cette page aspx dans votre navigateur en tapant son adresse dans la barre de navigation :  
*http://localhost/PremierePage/bonjour.aspx*  
  
et vous devez avoir une page web comme suit :



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.

### Visual Studio ou Visual Web Developper

Il est bien évident qu'un EDI digne de ce nom vous offre une multitude d'avantages comme la complétion du code, l'initialisation automatique de vos pages, les contrôles utilisateurs, ... malgré que, dans quelques cas, il est parfois plus avantageux de coder directement son contrôle dans la page HTML plutôt que de passer par le Designer de l'EDI.   
Que vous utilisiez Visual Studio ou Visual Web Developper (EDI gratuit et téléchargeable sur le site de Microsoft France), le démarrage de votre application sera presque le même. La seule différence vient du fait que Visual Studio étant un EDI qui regroupe plusieurs possibilités de développement, vous devrez spécifier que vous désirez travailler avec un nouveau projet web avant d'arriver sur cette page de configuration :



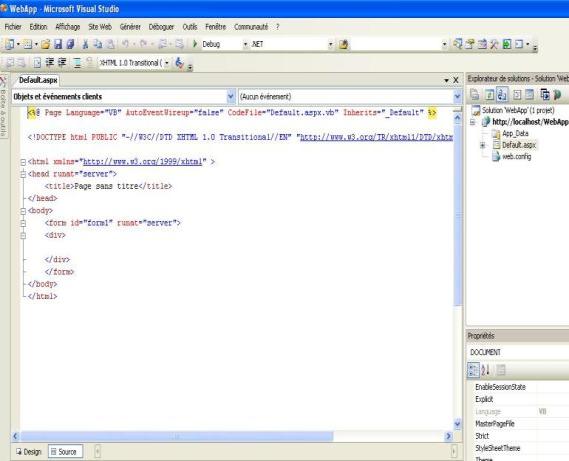
Il est primordial de bien configurer les informations dans **Emplacement** (Location) et **Langage** (Language) ainsi que l'adresse du site que vous désirez développer sur votre **serveur IIS local** (localhost).

## La gestion d’Etat

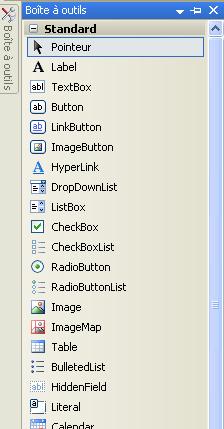
### Première page

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 :



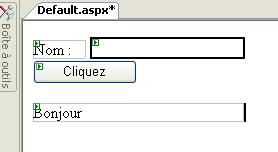
Vous pouvez remarquer que l'interface des applications ASP.NET diffère des applications Winforms mais nous y retrouvons quand même pas mal de points communs, notamment :   
  
**l'explorateur de solution** contenant notre projet "WebApp", sa localisation "http://localhost/WebApp" et la page par défaut "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 :



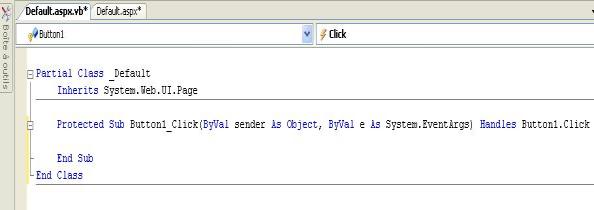
Passons maintenant en mode "design". Faites glisser sur la page les contrôles suivant et changez leurs propriétés en suivant le tableau ci-après :

|  |  |  |
| --- | --- | --- |
| **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 :  
  
textbox2.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.

### Des événements particuliers

ASP.NET possède des événements mais, certains sont assez particuliers et très importants pour le déroulement et le contrôle de ce genre d'application.

### Application

L’objet Application permet d'accéder à des méthodes et des événements d'application.

Permet également d'accéder à un cache d'application que vous pouvez utiliser pour stocker des informations.

L'état de l'application étant stocké dans la mémoire serveur, il est perdu chaque fois que l'application est arrêtée ou redémarrée. Par exemple, si le fichier Web.config est modifié, l'application est redémarrée et tout l'état de l'application est perdu.

Contrairement à l'état de session, spécifique à une session mono-utilisateur, l'état de l'application s'applique à tous les utilisateurs et à toutes les sessions. C'est donc un emplacement utile pour stocker de petites quantités de données souvent utilisées et qui restent identiques d'un utilisateur à un autre.

|  |  |
| --- | --- |
| **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 |

### Session

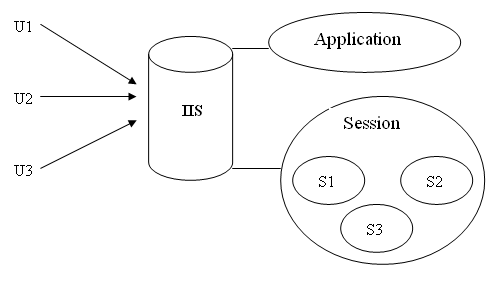
# L'état de session ASP.NET vous permet de stocker et de récupérer des valeurs pour un utilisateur.

# HTTP est un protocole sans état, ce qui signifie que votre serveur Web traite chaque demande de page HTTP comme une demande indépendante ; par défaut, le serveur ne connaît pas les valeurs de variables utilisées pendant des demandes antérieures.

# L'état de session ASP.NET identifie les demandes transmises par le même navigateur pendant une période limitée en tant que session, et permet de faire persister les valeurs de variables pour la durée de cette 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 |

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

### PostBack

**IsPostBack**

Une page ASP.NET est généralement utilisée plusieurs fois à la suite, dans une série d'échanges avec l'utilisateur.

Supposons qu'une page d'accueil (accueil. htm) soit affichée. L'utilisateur clique sur un lien qui conduit à la page saisie.aspx. Comme l'extension de la page est aspx, celle-ci est traitée par ASP.NET. Il s'agit alors de la première fois que la page est appelée.

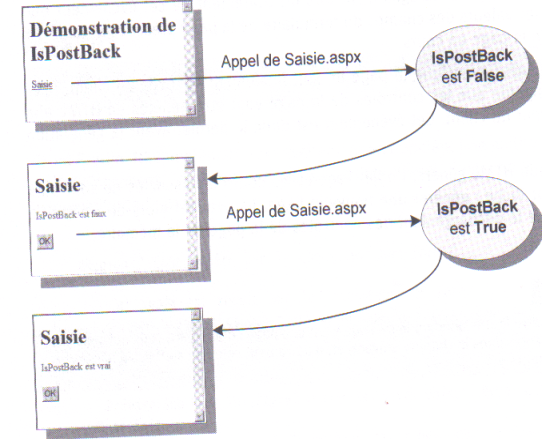
Cette page de saisie peut comprendre des contrôles qui provoquent un appel au serveur, par exemple des boutons de type asp:Button.

Quand l'utilisateur clique sur un bouton, la même page est appelée mais là ce n'est pas la première fois : la page précédente était la même.

Une propriété de la classe Page indique si la page est appelée la première fois ou non : IsPost­Back.

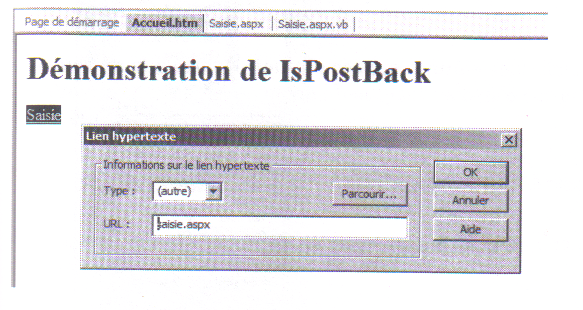
Si la valeur de cette propriété est False il s'agit du premier appel de la page.

Si la valeur de IsPostBack est True, la page est appelée par elle-même.



*Pour créer une* ***application qui met en évidence la propriété*** IsPostBack, ***vous pouvez effectuer*** *les* ***manipulations suivantes :***

* Créez un nouveau projet Application Web ASP.NET.
* Supprimez la page initiale Default.aspx et ajoutez un nouveau formulaire Web appelé Saisie.
* Ajoutez au projet une page HTML appelée accueil.htm.
* Cliquez à droite sur Accuei 1 dans l'explorateur de solutions et sélectionnez Définir comme page de démarrage dans le menu contextuel.
* Placez un lien sur la page Accueil dont l'URL de destination est Saisie.aspx



**Dans la page Saisie, placez un contrôle Web Form Label appelé IblIsPostBack et un con­trôle Button**

**« Vous pouvez maintenant ajouter le code suivant dans l'événement Page\_Load de la page Saisie:**

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

if **(IsPostBack==** false**)**

**{LblIsPostBack.Text =** " IsPostBack est faux "**;**

**LblIsPostBack.ForeColor =** Color.Red;}

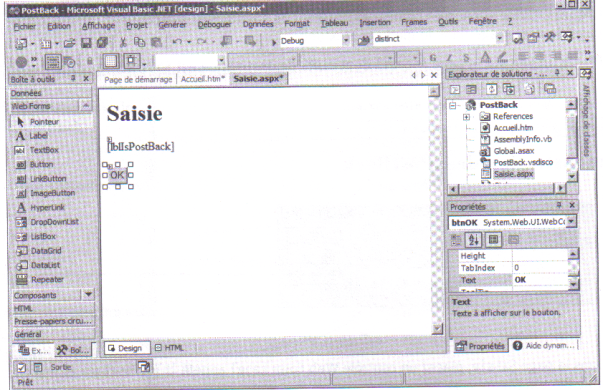
else

**{LblIsPostBack.Text =** " IsPostBack est vrai "**;**

**LblIsPostBack.ForeColor =** Color.Green;

**}**

**}**



### Les Servers 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".

**Gestion de l'état, la session, les cookies**

• Mise en évidence du problème

• Stockage des données sur le client

• Stockage des données sur le serveur

**On a vu que chacune des pages effectuait un travail spécifique, les seules relations entre une page et une autre étant un appel avec un hyperlien.**

La question qui se pose est comment effectuer un traitement sur plusieurs pages et particulièrement comment retrouver dans le code associé à une page les données d'une autre page.

le Web est par essence un système sans état: l'utilisateur demande une page, celle-ci est renvoyée par le serveur, puis tout est oublié ! Lors de la prochaine demande de l'utilisateur, le serveur ne se « rappellera » de rien. En d'autres termes, le serveur ne fait que répondre à des demandes ponctuelles de l'utilisateur, une par une, sans aucune connexion entre elles.

Pour resoudre de probleme on peut distinguer plusieurs situations et systèmes pour gérer l'état, c'est-à-dire faire passer la valeur de données d'une page à une autre.

Les quatre premières techniques se servent du client pour stocker les données :

* **Utiliser le ViewState, l'état d'affichage des pages Web mis en œuvre dans des sacs d'état *(state bags).***
* **Utiliser des champs cachés.**
* **Passer les données par l'URL.**
* **Placer les données dans des cookies sur le poste de l'utilisateur.**

**Les techniques suivantes stockent les données sur le serveur :**

* **Stocker les données dans des variables de session.**
* **Faire de même avec des variables d'application.**
* **Utiliser le contexte.**
* **Placer les données dans le cache.**

**La gestion de l'état concerne deux catégories de données :**

* **Les valeurs des variables de l'application, principalement les variables de la classe asso­ciée à la page.**
* **Les valeurs des propriétés des contrôles de la page.**

**TRAVAUX PRATIQUES :**

**Mise en évidence du problème**

Prenons un exemple simple pour montrer comment la gestion de l'état diffère dans les applica­tions Web de ce qu'elle est dans les applications classiques.

La page PageEtat1 présente un contrôle TextBox (txtNom) et un bouton OK (btnOK). Quand l'uti­lisateur clique sur le bouton, le contenu de la zone de saisie est recopié dans une variable de classe appelée Nom (il s'agit d'un membre de la classe associée à la page) :

**' Variable contenant le nom**

**String Nom=null ;**

**Private Sub btnOK\_Click(...) Handles btnOK.Click**

**‘ Stocke le nom dans une variable**

**Nom = txtNom. Text ;**

**End Sub**

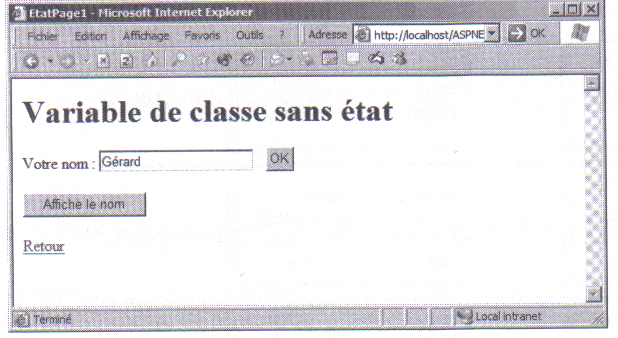
**Un second bouton sur la page (btnAfficheNom) permet d'afficher dans un contrôle Label (IblNom) le contenu de la variable Nom**

**Private Sub btnAfficheNom\_Click(...) Handles btnAfficheNom.Click**

**' Affiche le contenu de la variable**

**IblNom.Text = Nom;**

**End Sub**



### ViewState

Prenons un peu le temps de voir le code HTML de la page exécutée :



Vous constatez que des champs cachés ont été générés. Le champ nommé **\_VIEWSTATE** .

Les données **d'état de** la page

À chaque page est associé un état d'affichage (View State), qui stocke l'ensemble des données de la page et de ses contrôles. Cet état d'affichage est implémenté dans un objet de classe StateBag (littéralement, sac d'état), qui enregistre les données sous la forme de paires de clés et de valeurs, dans un dictionnaire.

NOTE :Pour que l’état d’affichage soit opérationnel, il faut que la propriété EnableviewState de la page soit True.

**Ces données sont transportées du serveur à une page sur le poste de l'utilisateur, puis de celle-| ci au serveur à nouveau, dans un champ caché du formulaire (un champ de type <input type=hidden>). Le contenu de ce champ correspond à l'ensemble des valeurs qui se trouvent dans l'objet StateBag, codées de tette façon qu'elles soient transportables sur le protocole HTTP et qu'il ne soit pas facile de les décoder ou de les modifier.**

**L'état d'affichage n'est utilisable que sur la même page appelée plusieurs fois, pas entre plusieurs pages différentes.**

**On peut accéder à l'objet StateBag associé à une page grâce à la propriété ViewState de l'objet Page. La clé associée à une donnée est automatiquement créée si celle-ci n'existe pas, ou elle est remplacée dans le cas contraire.**

**On peut ainsi écrire :**

**ViewState("Nom") = Value**

**pour stocker le contenu de Value sous le nom Nom. On pourra ensuite relire cette donnée**

**Nom = ViewState("Nom")**

**On peut ainsi transformer la page de l'exemple précédent afin de stocker le nom, non plus dans une variable de la classe, mais dans l'objet StateBag de la page.**

**On peut remplacer la déclaration de la variable Nom par une propriété de même nom et qui utilise l'objet StateBag :**

privateString **getNom()**

**{**

**nom = ViewState [**"nom"**].ToString ();**

return **nom;**

**}**

privatevoid **setNom(**String **value)**

**{**

**ViewState [**"nom"**] = value;**

**}**

**Ainsi, le code qui utilise la donnée Nom reste le même :**

**Private Sub btnOK\_Click(...) Handles btnOK.Click**

**' Stocke le nom dans une variable**

**ViewState [**"nom"**] = TextBox1 .Text ;**

**Private Sub btnAfficheNom\_Click(...) Handles btnAfficheNom.Click**

**' Affiche le contenu de la variable**

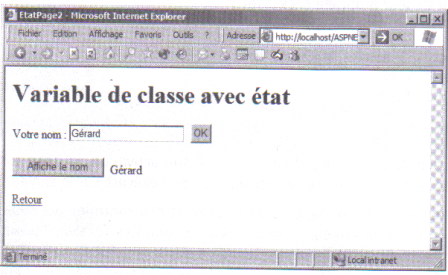
if **(ViewState[**"nom"**] !=** null**)**

**{**

**Label1.Text = ViewState[**"nom"**].ToString();**

**}**

**Avec cette nouvelle version, le nom s'affiche bien quand on clique sur le bouton.**



### Cookies

Stockage des données dans des cookies

Un cookie est du texte stocké sur le poste du client. Il est généralement enregistré à la demande du serveur, par l'intermédiaire de la propriété **Cookies** de l'objet HttpResponse retourné par la propriété Response de la page. Il peut être lu à travers la propriété Cookies de l'objet HttpRequest retourné par la propriété Request de la page.

**Pour écrire un cookie, qui est un couple nom-valeur, il suffit de lui donner une valeur, en indi­quant son nom comme paramètre de la collection Cookies. Si le cookie existait déjà, il est remplacé, dans le cas contraire, il est créé :**

**Response.Cookies[''MonCookie"].Value = "La valeur" ;**

**Un cookie écrit de cette façon n'est pas permanent : il n'existe qu'en mémoire, donc pendant la durée de l'application. Il disparaîtra quand celle-ci s'arrêtera.**

**Pour rendre un cookie permanent, il faut indiquer une date d'expiration. Par exemple :**

Response.Cookies["nom"**].Value = TextBox1.Text;**

**Response.Cookies[**"nom"**].Expires =** newDateTime**(2030, 1, 1);**

**Le cookie sera alors écrit sur le disque de l'utilisateur et y restera jusqu'à la date d'expiration ou jusqu'à ce qu'il soit effacé.**

**On peut lire un cookie en utilisant la même collection Cookies, maie appliquée à l'objet HttpRequest. Voici le code qui lit le cookie écrit précédemment :**

HttpCookie **cookie = Request.Cookies[**"nom"**];**

**TextBox1.Text = cookie.Value;**

**Ou** TextBox1.Text = Request.Cookies["nom"**].Value;**

**Si le cookie existait dans cette application, sa valeur est retournée. Dans le cas contraire, la valeur de retour est une chaîne de caractères vide.**

**L'ensemble des cookies d'une application est transmis avec chaque demande de l'utilisateur. Il est donc préférable de ne placer que de petites quantités de données dans les cookies, afin de ne pas grossir la trame HTTP circulant sur le réseau, d'autant plus que la taille d'un cookie est elle-même limitée.**

**La page exemple SaisieNom dispose d'un bouton Affiche avec cookie. Quand l'uti­lisateur clique dessus, le contenu du champ de saisie est placé dans un cookie temporaire appelé Nom. L'application est ensuite redirigée vers une autre page, à l'aide de la méthode Redirect de l'objet HttpResponse :**

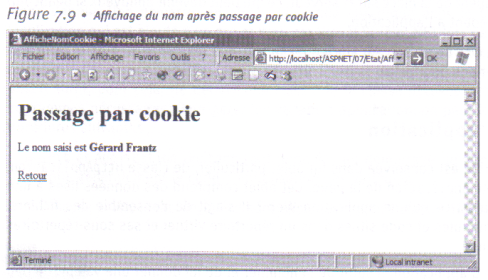
**Private Sub btnOKCookie\_Click(...) Handles btnOKCookie.Click**

**Response.Cookies[**"nom"**].Value = TextBox1.Text;**

**Response.Cookies[**"nom"**].Expires =** newDateTime**(2030, 1, 1);**

**Response.Redirect(**"AfficheNomCookie.aspx"**);**

**La page AfficheNomCookie affiche le texte lu dans le cookie.**



protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

**TextBox1.Text = Request .Cookies[**"nom"**].Value;**

**}**

**Stockage des données sur le serveur**

Toutes les techniques présentées précédemment stockent les données sur le poste du client. Celles de cette section les placent sur le serveur.

Cela présente quelques avantages :

* Les données ne sont jamais visibles par le client.
* Elles ne circulent pas sur le réseau et ne l'encombrent donc pas.
* Elles ne sont pas liées au poste utilisé par le client, comme c'est le cas pour les cookies (si l'utilisateur utilise un autre poste, il ne dispose pas de ses cookies).
* Leur accès est plus rapide, puisqu'elles se trouvent déjà sur le serveur.

Le stockage des données sur le serveur présente également quelques inconvénients :

* Les données occupent de la place sur le serveur, ce qui peut devenir ennuyeux si beaucoup d'utilisateurs accèdent à l'application.
* L'utilisateur peut être lié au serveur sur lequel se trouvent les données, bien qu'ASP.NET propose des solutions pour cela.

Les variables **d'application**

Une variable d'application est conservée dans un objet particulier, de classe **HttpApplication**, retourné par la propriété Application de la page.

Cet objet comprend des données liées à une application.

Pour créer une variable d'application, il suffit de la nommer et de lui donner une valeur, un peu comme pour les cookies présentés plus haut :

| Application["NomVariable"] = "Valeur variable"

Si la variable du nom indiqué existait déjà, sa valeur est remplacée, sinon elle est créée.

L'utilisation de variables d'application est donc extrêmement simple.

La page AfficheNomApplication affiche une donnée placée dans une variable Application appelée Nom par la page appelante, SaisieNom. Celle-ci exécute le code suivant lors du clic sur le bouton Affiche avec application:

protectedvoid **btnOKApplication \_Click(**object **sender,** EventArgs **e)**

**{**

**Application.Lock();**

**Application[**"nom"**] = TextBox1.Text;**

**Application.UnLock();**

**Response.Redirect(**"AfficheNomApplication.aspx"**);**

**}**

**On peut remarquer dans ce code que l'affectation de la valeur à la variable Application est accompagnée d'un appel aux méthodes Lock puis UnLock. Lock verrouille l'objet Application afin d'éviter qu'une modification y soit effectuée en même temps par un autre thread.**

**La page** AfficheNomApplication **affiche la valeur stockée dans la variable Application lors de son chargement :**

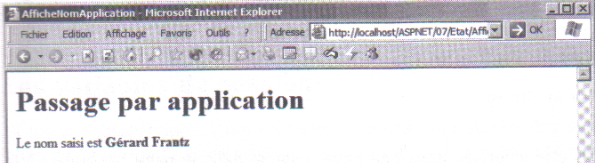
protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

**TextBox1.Text = Application[**"nom"**].ToString();**

**}**

**}**



**EXEMPLE D’UTILISATION PLUS COURANT:**

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.

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

if **(!IsPostBack)//la page est chargee pour la premiere fois**

**{**

int **cpt = 0;**

**Application.Lock();**

if **(Application[**"Compteur"**] !=** null**)**

**{**

**cpt =**int.Parse ( Application["Compteur"**].ToString ());**

**}**

**cpt += 1;**

**Application[**"Compteur"**] = cpt;**

**Application.UnLock();**

**lblVisite.Text =** "Page vue : " **+ cpt +** " fois"**;**

**}**

**}**

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.  
Les variables partagées

On ajoute une classe Global.cs dans laquelle on déclare des variables static qu’on peut manipuler dans tout le projet.

**Voici, par exemple, une variable déclarée dans une classe Global .cs**

publicclassGlobal

**{**

publicstaticstring **nom =** ""**;**

public **Global()**

**{**

//

// TODO : ajoutez ici la logique du constructeur

//

**}**

La variable Nom peut alors être valorisée dans une page, comme dans la page SaisieNom à la suite d'un clic sur le bouton Affiche avec variable Static:

protectedvoid **VariableStatic\_Click(**object **sender,** EventArgs **e)**

**{**

lock **(**Global.nom)

**{**

Global.nom = TextBox1.Text;

**}**

**Response.Redirect(**"PassageParVariableStatic.aspx"**);**

**}**

Le problème de l'accès simultané à la donnée est géré par le mot clé lock

L'utilisation de la donnée se fait dans une autre page, PassageParVariableStatic.

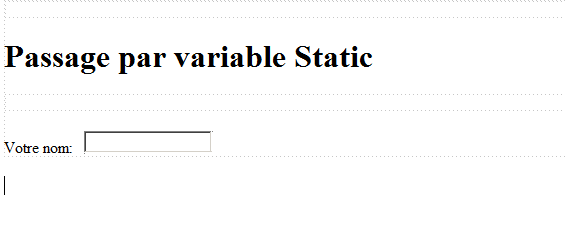
***Affichage du nom après passage par une variable partagée***

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

**TextBox1.Text =** Global.nom.ToString();

**}**

****

**Les variables de** session

Si les variables d'application sont intéressantes pour stocker les données d'une application, elles ne permettent pas de distinguer un utilisateur d'un autre. Les variables de session répon­dent à cette insuffisance, car elles sont associées à l'utilisateur courant.

La notion de session

Pour mettre en œuvre les variables de session, ASP.NET définit une notion de session qui com­prend l'ensemble des actions d'un utilisateur dans une application. Une session est reconnue par un identificateur de session créé par ASP.NET.

Quand un nouvel utilisateur appelle une page d'une application ASP.NET pour la première fois, un nouvel identificateur lui est attribué. Celui-ci accompagne ensuite toutes les réponses du système et les demandes de l'usager, ce qui permet de l'identifier. Deux techniques peuvent être utilisées pour cela : un cookie particulier enregistré sur le poste de l'utilisateur, ou l'inclu­sion de l'identificateur dans l'URL, essentiellement si les cookies ne sont pas autorisés par le navigateur de l'utilisateur.

Pour accéder aux informations d'une session, la classe Page expose une propriété, Session, qui retourne une référence à un objet HttpSessionState. Celui-ci dispose de propriétés et de méthodes, dont la propriété SessionID qui renvoie l'identificateur de session courant. On peut écrire, pour placer l'identificateur de session dans un contrôle label appelé IblSessionID :

| IbISessionID.Text = Session.SessionID ;

Le résultat est une chaîne de caractères comprenant l'identificateur de session (voir l'exemple suivant).

On peut d'ailleurs forcer une fin de session, en appelant la méthode Abandon :

| Session.Abandon();

Cela peut être effectué, par exemple, en réponse à un clic de l'utilisateur sur un bouton de décon­nexion placé sur la page. Après la fermeture d'une session, automatiquement ou manuellement, toutes ses données sont détruites.

Les variables de session

Comme les variables d'application, les variables de session sont simplement fabriquées en les nommant : si la variable existe, elle est utilisée, sinon elle est créée. Pour donner une valeur à la variable de session Nom, on peut simplement écrire :

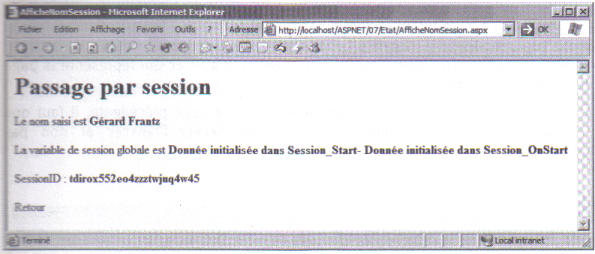
Session ["Nom"] = "Nouvelle valeur" ;

II n'est pas nécessaire de mettre en œuvre un mécanisme de synchronisation pour les variables de session, car elles ne sont normalement rejointes que par un seul thread, n'étant liées qu'à un seul utilisateur.

L'initialisation des variables de session peut se faire dans une procédure Session\_OnStart (ou Session\_Start) et Session\_OnEnd permet d'effectuer les traitements de fin de session.

Ces deux procédures doivent être écrites dans le fichier global.asax, comme cela a été expliqué dans la section relative aux variables d'application.

La page AfficheNomSession affiche une variable de session initialisée lors de la saisie du nom de l'utilisateur dans la page SaisieNom. Elle affiche également la valeur d'une variable initialisée dans la procédure Session\_OnStart .

****

**La valorisation de la variable à la suite de la saisie du nom est effectuée par le code suivant**

protectedvoid **BtnOkSession\_Click(**object **sender,** EventArgs **e)**

**{**

**Session[**"nom"**] = TextBox1.Text;**

**Response.Redirect(**"AfficheParSession.aspx"**);**

**}**

**L'initialisation des variables dans global .asax est :**

void **Application\_Start(**object **sender,** EventArgs **e)**

**{**

// Code qui s'exécute au démarrage de l'application

**Application[**"Application\_Start"**] =** "Application\_Start"**;**

**Application[**"somme"**] = 1000;**

**}**

void **Session\_Start(**object **sender,** EventArgs **e)**

**{**

// Code qui s'exécute lorsqu'une nouvelle session démarre

**Session[**"Start"**] =** "Donnée initialisée dans Session\_Start"**;**

**}**

**Enfin, l'affichage des variables est réalisé par le code suivant**

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

**TextBox1.Text = Session[**"nom"**].ToString ();**

**TextBox2.Text = Session.SessionID;**

**lblSession\_Start.Text = Session[**"Start"**].ToString ();**

**}**

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

### QueryString

QueryString permet de faire passer des informations via l'URL d'une page à une autre.

protectedvoid **BtnOkQueryString\_Click(**object **sender,** EventArgs **e)**

**{**

**Response.Redirect(**"AfficheParQuery.aspx?nom=" **+ TextBox1.Text);**

**}**

Pour récupérer la valeur saisie :

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{**

**Label1.Text = Request.QueryString[**"nom"**].ToString ();**

**}**

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

Résumé

Nous avons vu que si la gestion de l'état est un véritable problème pour les applications Web, elle dispose également de nombreuses solutions. Plusieurs mécanismes per­mettent de stocker les données d'une page sur le client, ou sur le serveur. Des techniques nouvelles dans ASP.NET permettent même de stocker les données sur un serveur partagé par plusieurs serveurs Web.

## 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](http://msdn.microsoft.com/fr-fr/library/system.web.ui.control(VS.80).aspx) ou de [WebControl](http://msdn.microsoft.com/fr-fr/library/system.web.ui.webcontrols.webcontrol(VS.80).aspx).

### 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](http://msdn.microsoft.com/fr-fr/library/ydy4x04a(VS.80).aspx), le contrôle utilisateur contient une directive [@ Control](http://msdn.microsoft.com/fr-fr/library/d19c0t4b(VS.80).aspx) 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](http://msdn.microsoft.com/fr-fr/library/system.web.ui.webcontrols.button(VS.80).aspx) 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.

<%@ControlLanguage="C#"AutoEventWireup="true"CodeBehind="WebUserControl1.ascx.cs"Inherits="ControlUtilisateur.WebUserControl1"%>

<asp:HyperLinkid="lnkLogin"runat="server"

NavigateURL="Login.aspx">**Login**</asp:HyperLink>

&nbsp; **|**

<asp:HyperLinkid="lnkAddToCart"runat="server"

NavigateURL="Cart.aspx">**Add to Cart**</asp:HyperLink>

&nbsp; **|**

<asp:HyperLinkid="lnkTechSupport"runat="server"

NavigateURL="TechSupport.aspx">**Technical Support**</asp:HyperLink>

&nbsp; **|**

<asp:HyperLinkid="lnkAbout"runat="server"

NavigateURL="AboutUs.aspx">**About Us**</asp:HyperLink></P>

## 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.

### 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".

### 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>

### 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> |

### 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>

Caractère d’expression régulière :

a : la lettre a doit être utilisée en minuscule.

1 :le chiffre 1 doit être utilisé

?: 0 ou 1 élément

\*:de 0 a n éléments

+ :de 1 a n éléments.

[0-n] :valeur entière,comprise entre 0 et n

{n} :la longueur doit être de n caractères.

\ :le caractère suivant est un caractère de commande.

\w : doit contenir un caractère.

\d : doit contenir un chiffre.

\. :doit contenir un point

([-+.]\w)\* :peut contenir un ou plusieurs traits d’union ou un point et une chaine qui contient un caractère au moins.

Exemple :

ValidationExpression = " \w+@\w+\.\w+"

\w+ : une chaine qui contient un caractere au minimum.

@ :un signe @

\w+ : une chaine qui contient un caractere au minimum.

\.  : un point

\w+ : une chaine qui contient un caractere au minimum.

### 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**

### 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.

# L’accès aux données avec ASP.NET

## 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](http://msdn2.microsoft.com/fr-fr/library/system.data(VS.80).aspx) (connu sous le nom ADO.NET) et de l'espace de noms system.Xml](http://msdn2.microsoft.com/fr-fr/library/system.xml(VS.80).aspx). 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.

## 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.objectdatasource(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.sqldatasource(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.accessdatasource(VS.80).aspx) | Permet d'utiliser une base de données Microsoft Access. |
| [XmlDataSource](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.xmldatasource(VS.80).aspx) | Permet d'utiliser un fichier XML, ce qui est très utile pour les contrôles serveur ASP.NET hiérarchiques tels que les contrôles [TreeView](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.treeview(VS.80).aspx) ou [Menu](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.menu(VS.80).aspx). |
| [SiteMapDataSource](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.sitemapdatasource(VS.80).aspx) | 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.

## 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.basedataboundcontrol.datasourceid(VS.80).aspx).

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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.bulletedlist(VS.80).aspx), [CheckBoxList](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.checkboxlist(VS.80).aspx), [DropDownList](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.dropdownlist(VS.80).aspx), [ListBox](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.listbox(VS.80).aspx) et [RadioButtonList](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.radiobuttonlist(VS.80).aspx).

|  |  |
| --- | --- |
| Contrôle | fonctionnement |
| [AdRotator](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.adrotator(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.detailsview(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.formview(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.gridview(VS.80).aspx) | 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](http://msdn2.microsoft.com/fr-fr/library/system.web.ui.webcontrols.repeater(VS.80).aspx) | 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. |

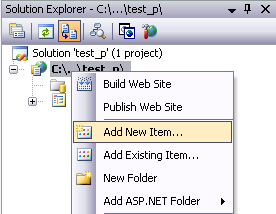
# Master Page

## Introduction aux MasterPages

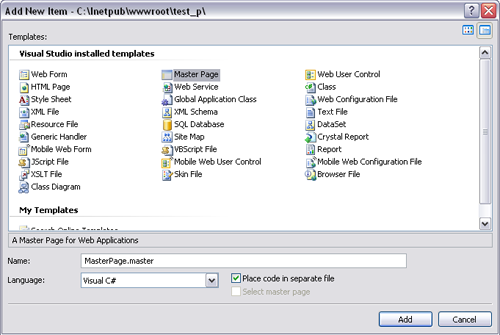
Depuis longtemps, les développeurs ont toujours été contraint de dupliquer les sources HTML du design sur chaque page. D’autre solution intermédiaire existait, par exemple l’utilisation des frames en HTML, cependant cette fonction appartient au langage HTML et présente beaucoup de problème, notamment 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

## 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>

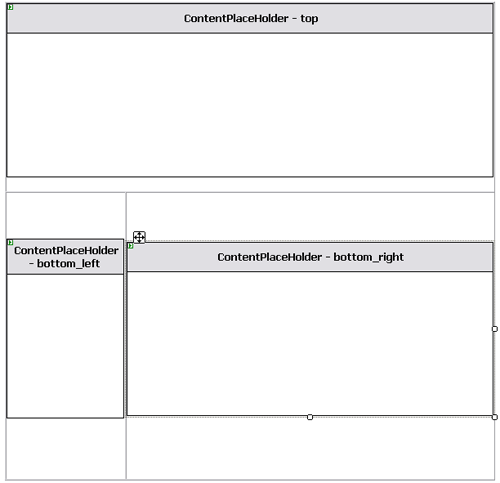
</tr>

</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.

## 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

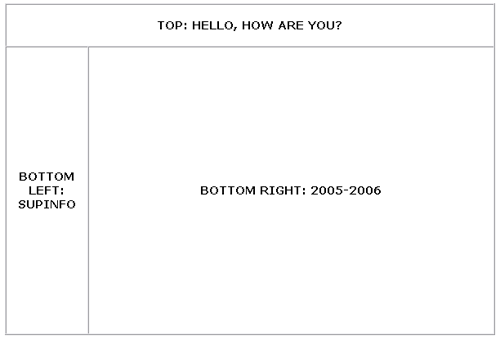
</asp:Content>

<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.

## 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.

# Thèmes ( apparences(Skins) et feuilles Css)

## Introduction aux thèmes

Les 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 :

* les couleurs
* les polices
* la taille des tableaux
* la taille des différents composants etc….

C’est là qu’interviennent les thèmes, plus exactement la mise en place d’un style sur un site.   
Avant ce sont des CSS (Cascading Style Sheet),mais ils ne permettent pas le formatage des contrôles ASP.NET.

## Création d'un thème

Les thèmes sont stockés obligatoirement dans le dossier App\_Themes(à créer)composé au minimum d’un fichier Skin (\*.skin),sinon de plusieurs fichier skin et Css.(bouton droit sur le projet /Ajouter le dossier ASP.NET/Thèmes)  
  
 Ajouter ensuite 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 ».

## Les fichiers Skins

Les fichiers Skins sont composés essentiellement de contrôle ASP.NET avec leurs propriétés de formatage (police, couleur, taille…).

Vous avez la possibilité d’ajouter :

* des contrôles communs (sans SkinID)
* des contrôles spécifiques (avec SkinID).

Si le thème est appliqué :

* les contrôles communs s’appliqueront à tous les contrôles du même type.
* 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" />

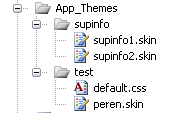
Si le thème est appliqué sur tout le site, toutes les TextBox auront le même style.

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



Pour créer une apparence, une méthode facile consiste à ajouter le contrôle à une page et à le configurer comme il vous convient. Par exemple, vous pouvez ajouter un contrôle Calendar à une page et définir son en-tête des jours, la date sélectionnée et d'autres propriétés. Ensuite, vous pouvez copier la définition du contrôle de la page en mode source dans un fichier .skin, mais vous devez supprimer l'attribut ID.

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" />

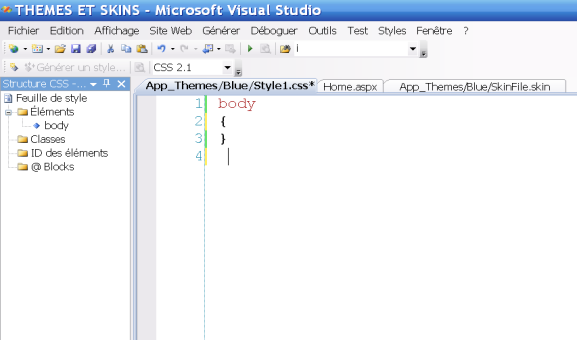
Il est tout à fait possible de créer plusieurs fichiers Skins dans un même thème, cependant les contrôles ayant un SkinID doivent être uniques dans un thème, sinon vous ne pourrez pas compiler votre application WEB.

## Les fichiers CSS

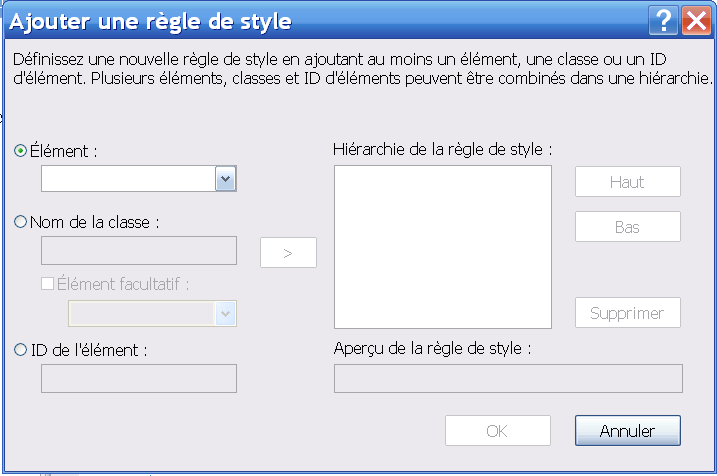
Pour créer une feuille de style click droit sur le dossier App\_themes/Theme1 ,ajouter un nouvel élément choisir feuille de style donner un nom à la feuille de style.

La feuille est vide pour ajouter des elememts et des classes de style,double click sur la feuille.

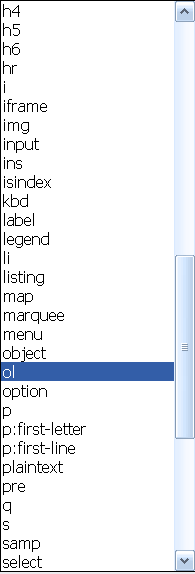
Un menu structure Css apparaît à gauche :



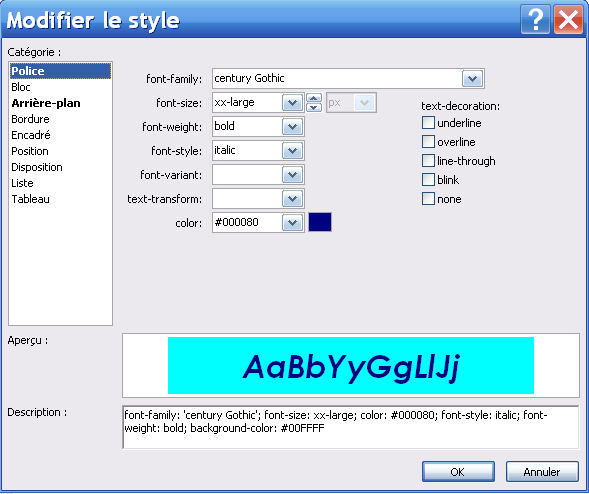
Click droit sur Elements ,Ajouter une regle de style :



Choisir l’élément pour lequel on veut créer un style :







Faire la même chose pour ajouter une classe.

## 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="theme1" 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 |
| theme_apply1.png | theme_apply2.png |

## 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.

## 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.

# Profiles

## Introduction aux Profiles

Plusieurs façons de stocker les informations des utilisateurs :

Les sessions, les cookies …

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 permanentes et automatiquement.

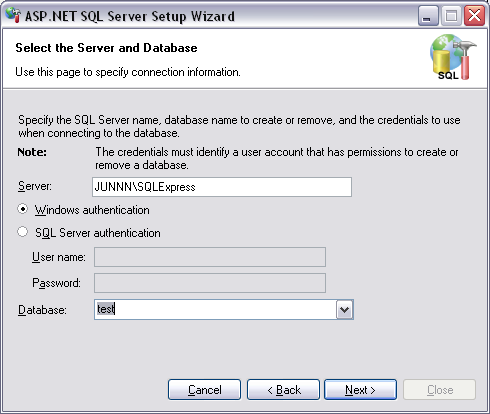
La fonctionnalité de profil ASP.NET vous permet de stocker des valeurs simples (scalaires), des collections et d'autres types complexes, ainsi que des types définis par l'utilisateur.

## Implémentation des Profiles

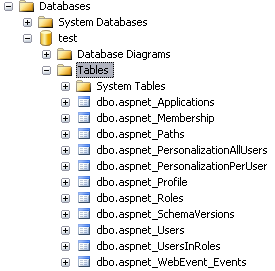
Microsoft met à disposition un exécutable permettant d’installer les Profiles (de même pour l’utilisation des Membership et Roles) dans une base de donnée.

Cette exécutable se trouve dans le chemin suivant : **C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet\_regsql.exe**

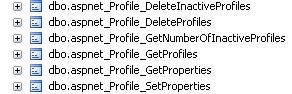
Il vous suffit de lancer le programme, choisir si vous voulez créer une nouvelle base (ASPNETDB sera le nom de la base par défaut) ou installer tous les composants dans une base existante.

  
Fig 4.1 Installation aspnet\_regsql.exe

Voici les nouvelles tables installées :

  
Fig 4.2 Tables installées

Et les procédures stockées concernant les Profiles :

  
Fig 4.3 Procédures pour les Profiles

## Mise en place des Profiles

Les Profiles se définissent dans 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 défaut, c'est-à-dire le Provider définit dans le champ 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 :

Donnees simples :

<system.web>

<anonymousIdentification enabled="true" />

<profile enabled="true"

defaultProvider="AspNetSqlProfileProvider">

<properties>

<add name="PostalCode"

type="System.String"

allowAnonymous="true" />

</properties>

</profile>

</system.web>

## 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 se fait de cette manière :

Profile.[propriete] = [valeur]

Voici un exemple :

Pour remplir la balise connectionStrings automatiquement ajouter un DropDownList et le configurer avec la base de profil.

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&nbsp;<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 :

protectedvoid **Page\_Load(**object **sender,** EventArgs **e)**

**{ Label2.Text = Profile.age.ToString();}**

protectedvoid **Valider\_Click(**object **sender,** EventArgs **e)**

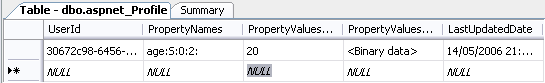
**{ Profile.age =** int.Parse (TextBox1.Text); **}**

  
Fig 4.5 Exemple stockage information

Le contenu de la table aspnet\_Profile avant validation :

profiles_define1.png  
Fig 4.6 Tables aspnet\_Profile avant validation

Le contenu de la table aspnet\_Profile après validation :

  
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;

## 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;

## 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.

# Sécurité en ASP.NET 2.0

## Introduction

Chaque site Internet nécessite une certaine sécurité pour éviter tous problèmes.

Nous allons pouvoir appliquer des autorisations d'accès à certaines pages/dossiers.

## Le fichier de configuration: Web.config

Le fichier Web.config est un fichier XML permettant de gérer la configuration de votre Website ou application.   
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.

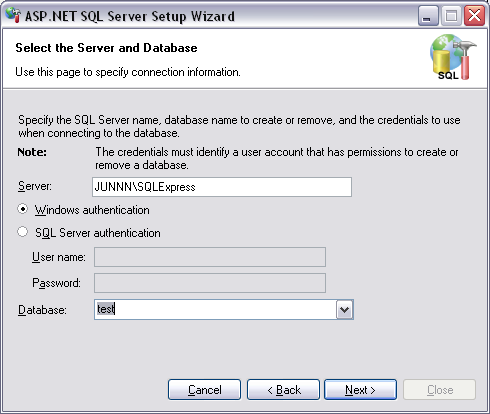
## Utilisation des Memberships et rôles

### 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\v2.0.50727\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.



### Memberships

La 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...

Pour utiliser les memberships dans votre projet, vous devez obligatoirement modifier le fichier Web.config (ou modifier par l'intermédiaire du WSAT :website Administration tool), 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"/>  </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 taper "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 |

### 

### 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 "rôles".  
Voici 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.

Haut du formulaire

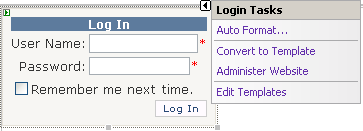


## Les contrôles de login

Dans le précédent chapitre, nous avons vu les différentes méthodes possibles pour la classe membership, cependant toutes ces méthodes peuvent être géré 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.

### Login

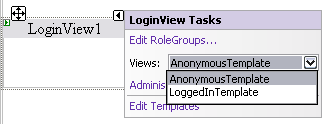
Le Login permet tout simplement à un utilisateur de se connecter en indiquant son login et son password, il vous suffit simplement de faire du Drag an Drop, prendre l'objet login et le déposer sur une page.



Vous avez la possibilité de modifier le template, soit en utilisant les templates intégrés 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.

### LoginView

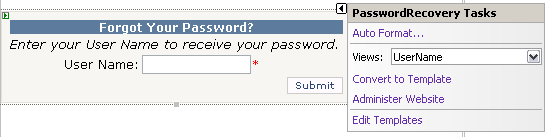
Le loginView permet de gérer des templates différents:  
- le template pour les utilisateurs anonymes  
- le template pour les utilisateurs connectés



|  |
| --- |
| <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]".

### 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.

### LoginStatus

Le LoginStatus 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).

### 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.

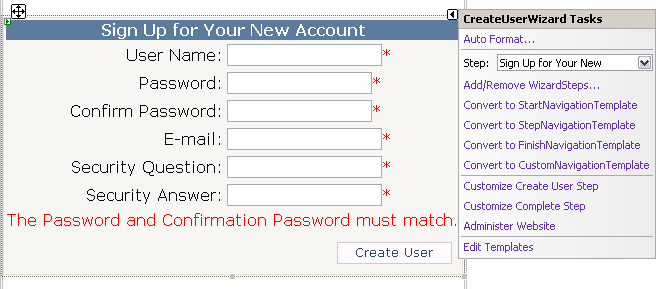
|  |
| --- |
| <asp:LoginView ID="LoginView1" runat="server">  <AnonymousTemplate>  <asp:Login ID="Login1" runat="server"  DestinationPageUrl="~/login.aspx">  </asp:Login>  <asp:LoginStatus ID="LoginStatus1"  LogoutPageUrl="~/default.aspx" runat="server" />  </AnonymousTemplate>  <LoggedInTemplate> 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



loginstate_connected.png

### 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.



### ChangePassword

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



## 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> |

### 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).

### Passport

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

### Windows

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

### 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.

## Appliquer des autorisations

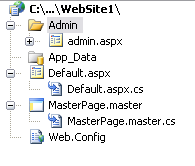
### Les balises

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 dossier 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="?"/>  
</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.

### 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="\*"/>  
</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.aspx">   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.

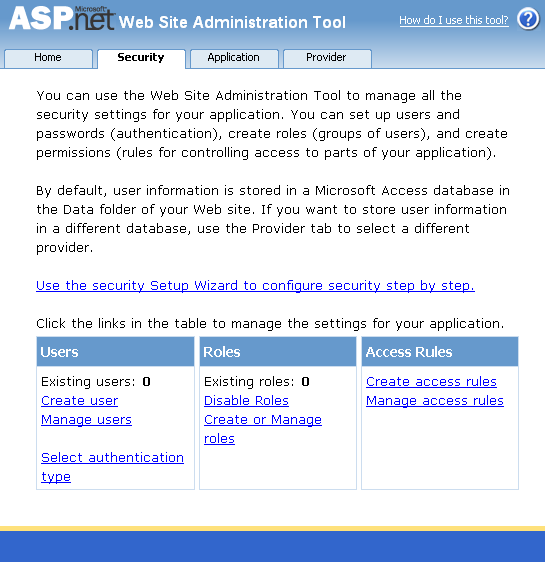
## WSAT - Web Site Administration Tool

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.



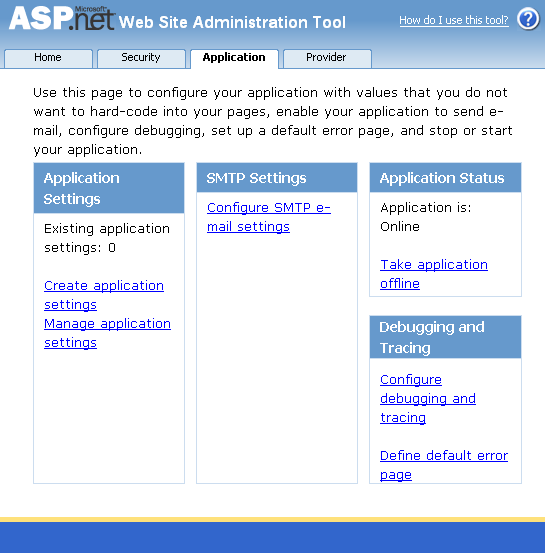
### Security

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.



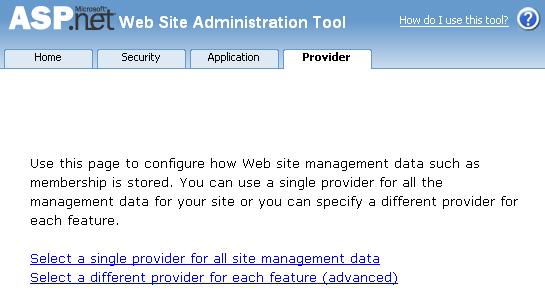
### Application

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



### Provider

Vous avez la possibilité 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.



Bas du formulaire

# 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.