## ASP.NET

LA PROGRAMMATION DU SERVEUR INTERNET: ASP.NET

Cette séance fait suite à la séance de programmation du serveur internet en ASP ou en PHP - Elaboration de maquettes de serveur WEB , répartition des données (SGBD) et des traitements.

OBJECTIFS A REALISER

Présenter les concepts nouveaux introduits par l'infrastructure .NET

Mettre en œuvre les aspects essentiels d'ASP.NET (Web Forms, accès aux données par ADO.NET) et s'initier au langage C#

Programmation:

Utiliser Visual Studio .NET pour gérer le projet

Utiliser les classes fondamentales du FRAMEWORK .NET

Utiliser le compilateur C#

PRE REQUIS

Maîtriser un langage de programmation objet du type java, C++ ou VB.NET

Avoir déjà développé des pages dynamiques côté serveur en asp, jsp ou php.

Connaître le langage de requêtes SQL et les bases du DHTML côté navigateur client.***SOMMAIRE…………………..***

#### INTRODUCTION à ASP.NET

* Les points forts
* L’infrastructure .NET
* XML et Internet

#### CARACTERISTIQUES D’ASP.NET

* La compilation des pages
* L’objet
* La sécurité
* Le fonctionnement d’ASP.NET
* Pourquoi choisir le langage C#

#### CREATION D’ UNE APPLICATION ASP.NET

* Rappels généraux
* Les bases de l'ancienne technologie ASP
* D'ASP vers ASP.NET
* Principes d'ASP.NET
* Mise en œuvre des différentes structures d’une page ASP.NET; exemples

#### CREATION D' UNE APPLICATION AU MOYEN D' UN PROJET SOUS VISUAL STUDIO .NET *formulaire WebForm1*

* Principes
* Les fichiers du projet
* mécanismes objet et événementiel
* La mise au point par le débogueur

#### CREATION D' UN PREMIER FORMULAIRE WEB

* Principes *formulaire WebForm1*
* Mise en œuvre : composition graphique du formulaire
* une première conclusion

#### LES EVENEMENTS DES CONTROLES ASP.NET

#### LES COMPOSANTS ET CONTROLES DES PAGES WEB FORMS

* System.Web.UI.HtmlControls *formulaire WebForm2*
* une première conclusion
* System.Web.UI.WebControls
* une deuxième conclusion

#### LES COMPOSANTS DE VALIDATION *formulaire WebForm3*

* Principes
* Exercice dirigé

#### UTILISATION D' UN ESPACE DE NOM : System.IO

* Principes *formulaire WebForm3*
* Mise en oeuvre

#### Les composants de contrôle de données

#### *formulaire WebForm4*

* Principes du Data Binding
* Un premier exemple : Data Binding sur une propriété d'un composant de contrôle
* Un deuxième exemple : Data Binding sur une fonction
* Un troisième exemple : l'utilisation des composants de contrôle DataList et DataGrid

#### Association des composants de contrôle avec

#### des bases de données *formulaire WebForm5*

* Principes
* Exemples

#### L'outil de "Connexions de données" de Visual Studio

#### PRESENTATION D' ADO.NET

* Principes
* Exemple d'accès en mode déconnecté : le DataSet *formulaire WebForm6*
* Exemple d'accès en mode connecté : le DataReader *formulaire WebForm7*
* Les DataView *formulaire WebForm8*
* Les commandes sur les procédures stockées *formulaire WebForm9*
* Les commandes qui retournent une valeur scalaire *formulaire WebForm9*
* Conclusion

#### LA MODIFICATION DES DONNEES DANS LA BASE

*formulaire WebForm10*

#### LA GESTION DES ETATS *formulaire Index*

* Première problématique
* Deuxième problématique
* Gestion des états côté client
* Gestion des états côté serveur
* Retour sur le fichier Global.asax

#### La télétransmission des fichiers clients : l' UPLOAD

*formulaire WebForm11*

#### Ajouter des références DE COMPOSANTS au projet

*formulaire WebForm12*

Documents distribués:

Des sites:

www.asp.net www.gotdotnet.com www.ibuyspy.com msdn.microsoft.com

#### INTRODUCTION à ASP.NET

* **Les points forts**

ASP.NET (Application Service Provider) propose un environnement de conception et de diffusion d’applications web situées sur un serveur web, IIS ou autre. C’est une refonte complète des ASP.

* Indépendance du langage de programmation grâce à un module CLR (common language runtime) qui fédère le code en provenance de plusieurs pages.
* Gain de performance grâce à la compilation des pages.
* Grandes fonctionnalités de déboguage de l'environnement .NET
* Orientation composants, événements, et objets (possibilité d’héritage et d’enrichissement des classes standard). Le programmeur est déchargé de l’écriture des scripts clients de gestion des formulaires et des scripts serveurs de validation de ces formulaires. Les contrôles IHM fonctionnent côté serveur par l'attribut *"runat=server"*
* Séparation entre le code de présentation et la logique de traitement (plusieurs fichiers décrivent la page).
* Notion de "services métiers" grâce aux web services qui encapsulent des classes développées dans différents langages.
* Grande richesse de classes et de supports d'accès aux données (via ADO.NET.)
* Déploiement simplifié : tout projet est un ensemble de fichiers textes qu’il suffit de recopier sur le serveur cible.
* Cohabitation entre les applications ASP et les applications ASP.NET
* **L’infrastructure .NET**

*L’infrastructure .NET et son VisualStudio.Net permettent de concevoir et déployer des applications Windows, des applications console DOS, des applications Web et enfin des services Web et Windows, sous une approche Windows DNA (Distribued Network Application)*

*Lorsque vous créez des applications faisant intervenir une interface utilisateur, vous avez le choix entre utiliser des Windows Forms ou des Web Forms :*

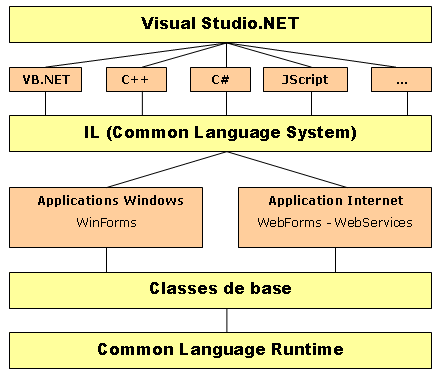
* *Les Windows Forms sont utilisés pour développer des applications dans lesquelles le client est supposé prendre à sa charge une part importante des traitements. Les applications utilisant des Windows Forms étant construites sur une infrastructure Windows, elles ont accès aux ressources du système de l'ordinateur client (fichiers locaux, registre Windows, imprimante, etc.). Ce niveau d'accès peut être restreint.*
* *Les Web Forms d'ASP.NET sont utilisés pour créer des applications dans lesquelles la principale interface utilisateur est un navigateur. Les navigateurs disposant habituellement d'un accès limité aux ressources du système de l'utilisateur, les Web Forms ont un accès restreint au système du client.*

*Le* ***.NET Framework*** *est une nouvelle plate-forme informatique qui simplifie le développement d'applications. Le .NET Framework contient deux composants principaux : le CLR (Common Language Runtime) et la bibliothèque de classes du .NET Framework.*

*Le* ***CLR*** *manage le code au moment de l'exécution, fournit des services essentiels comme la gestion de la mémoire, la gestion des threads, et l'accès distant. Il applique également une stricte sécurité des types et d'autres formes d'exactitude du code qui garantissent un code sécurisé et robuste. Le code qui cible le runtime porte le nom de* ***code managé****, tandis que le code qui ne cible pas le runtime porte le nom de* ***code non managé****.*

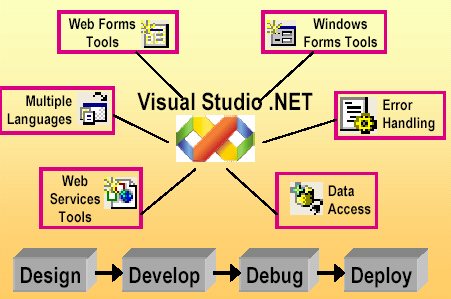
*La* ***bibliothèque de classes*** *est une collection complète orientée objet, formée de* ***types*** *(environ 1700) utilisables pour développer des applications à ligne de commande, ou à interface graphique utilisateur, ou des services Web XML et Web Forms.*

*Microsoft .NET Framework SDK Documentation*



L’édition **ASP.NET** comporte :

* Les outils de développement, déploiement et exécution de services et d’ applications de l’environnement .NET Framework Les trois compilateurs VB.Net, C# et JScript.Net
* Les librairies de classes .NET accessibles à tous les langages
* Le sous système ADO.NET d’accès aux données
* Le support fondamental d’ASP.NET (un jeu de contrôles qui encapsule des éléments d'interface utilisateur HTML communs, et des services d'infrastructure, tels que la gestion des états de session )



* **XML et internet**

La technologie XML s’associe à la technologie CLR (Common Language Runtime) pour créer l’infrastructure .NET

Quatre interfaces de programmation sont fournis et permettent l’exploitation de documents XML :

* Le DOM (modèle d’objet documentaire) : des fonctions permettent de lire et modifier le document qui est alors complètement chargé en mémoire. Très utile côté client.
* L’interface SAX (Simple API for XML) qui ne charge en mémoire que la partie utile du document, en le parcourant comme un flux séquentiel et en réagissant à chaque nœud. Très utile avec les gros documents.
* Le langage XSL (extensible stylesheet language), évolution des CSS, qui contrôle la présentation du document XML (en format HTML par exemple) en lui associant un document feuille de style XSL (avec sa syntaxe propre appelée XSLTransformation) et la recommandation Xpath de circulation dans un arbre hiérarchique.
* Des fonctions spécifiques à .NET pour accéder aux documents XML

#### CARACTERISTIQUES D’ASP.NET

*ASP.NET est une plate-forme servant au développement et à l'exécution d'applications sur un serveur Web. ASP.NET fait lui-même partie du .NET Framework, de sorte qu'il permet d'accéder à toutes les fonctionnalités de cette infrastructure. Vous pouvez, par exemple, créer des applications Web ASP.NET à l'aide de n'importe quel langage de programmation .NET (Visual Basic, C#, les extensions managées de C++ et bien d'autres encore) et des fonctionnalités de débogage .NET. Vous accédez aux données via ADO.NET. De la même façon, vous pouvez accéder aux services de système d'exploitation à l'aide des classes .NET Framework, etc.*

*Microsoft .NET Framework SDK Documentation*

* **La compilation des pages**
* Unification de plusieurs langages grâce à la **CLR** (Common Language Runtime) :

Le code source de chaque langage est compilé par la CLR vers un langage intermédiaire (**MSLI**, Microsoft Intermediate Language) grâce à une spécification commune (**CLS**, Common Language Specification) , puis lié avec des méta données pour créer un exécutable (ou une dll) appelé **assembly**.

La CLR joue ensuite le rôle de machine virtuelle qui compile en code natif (code machine optimisé pour le processeur) et exécute le fichier binaire MSLI.

Microsoft fournit gratuitement une spécification commune de cette CLR aux différents éditeurs de logiciels.

* Rapidité d’exécution :

Au moment de l’exécution de la page, il y a traduction de ce langage intermédiaire par le compilateur JIT (Just In Time) de la CLR en code natif.

Exemple: une page est compilée lorsqu'elle doit s'afficher, mais les méthodes de traitement des événements de ses contrôles ne sont compilées qu'au moment où ces événements surviennent. Ces compilations n'ont lieu qu'une seule fois et restent en mémoire pour les prochaines invocations de la page.. Cependant elles peuvent être supprimées en cas de modification du code de la page ou en cas de besoin d'espace mémoire.

* Sécurité d’exécution

En cours d’exécution de la page, la CLR surveille (débordements mémoire, …) et recycle (garbage collector) l’espace mémoire

* **L’objet**

La programmation .NET repose sur un ensemble très riche de classes qui remplacent les objets COM (accès fichiers, mailing, téléchargements …). Cependant certains objets COM sont exploitables par le biais de couches d’adaptation.

Les applications Windows et les applications Web utilisent le même jeu de librairies de classes.

Les classes sont regroupées thématiquement dans des "**espaces de noms**" qui sont en fait des conteneurs hiérarchisés (comme les packages java). Ces espaces sont des fichiers .exe ou .dll .

Il suffit d’importer l’espace des noms adéquat pour utiliser une classe dans une page asp.net

Cependant certains espaces de noms sont importés par défaut (pour les collections, les contrôles IHM, les accès aux fichiers, les exceptions, …).

* **La sécurité**

La sécurité est renforcée par le biais de services d’authentification (Authentification Provider) qui assurent les contrôles grâce à des listes ACL de Windows, des bases de données d’utilisateurs ou des fichiers XML.

Il y a trois modes d’authentification (Window, Passport, Cookie).

La sécurité est également renforcée par le biais du contrôle d’accès aux ressources (contrôle au niveau fichier si authentification windows , contrôle au niveau URL par un fichier de configuration – web.config).

Enfin ces mécanismes d’authentification et de contrôle d’accès peuvent être renvoyés au serveur IIS par un mode dit « Impersonation »

Le suivi de session sous ASP.NET est plus sécurisé que sous asp, et ne nécessite plus forcément l’utilisation d’un cookie dans le navigateur du client (propriété cookieless du fichier de configuration web.config). Dans ce cas un ID unique est marqué dans l’URL .

De plus les données de la session peuvent être conservées soit dans le processus ASP.NET (comme cela se fait déjà en asp), soit hors de ce processus dans une mémoire séparée ou dans une base de données (plus lent mais plus sécurisé en cas de chute du serveur).

* **Le fonctionnement d’ASP.NET**

L’utilisateur demande une page d’extension **.aspx** au serveur.

Celle ci est lue sur le disque, éventuellement compilée en mémoire, puis elle génère du code HTML qui est transmis au serveur qui le renvoie à l’utilisateur.

Mais si la page .aspx comporte des composants d’interface interactifs, qui demandent un traitement côté serveur (attribut **runat= "server"),** le processus n'est pas terminé.

En effet une interaction de la part de l'utilisateur re-soumet la page aspx avec cette fois des éléments cachés dans le formulaire (de type "**hidden**") qui lui permettront de savoir comment réagir (exécuter un gestionnaire d'événement par exemple).

La page génère à nouveau une réponse HTML à l'utilisateur.

Et ainsi de suite … Le traçage de ce processus se fait donc par le biais de champs cachés.

Le détail sera vu dans le chapitre "Web Forms".

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpovrintroductiontonetframeworksdk.htm

* **Pourquoi choisir le langage C#**

*"Le langage C# est un langage orienté composants, conçu pour la programmation du Web.*

*Il reprend la plupart des concepts de SmallTalk et de Java en les intégrant dans une syntaxe héritée du C++. L’idée est de corriger les défauts du C++ tout en l’étendant."*

dixit http://www.microsoft.com/france/vcsharp/utilisez/infotech/default.asp

#### CREATION D’UNE APPLICATION ASP.NET

* **Rappels généraux**

Les pages de scripts serveurs ASP, PHP, JSP …permettent de développer des applications coté serveur qui rendent plus facile la construction d'applications Web dynamiques.

De telles pages peuvent contenir du code HTML, du code exécutable sur le serveur et du code exécutable sur le client (décrit en JavaScript ou en VBScript). Lorsque le serveur reçoit une requête HTTP pour une page script, il crée une page HTML virtuelle et temporaire qui contiend du code HTML et du code client, statiques ou créés dynamiquement par le script serveur .

Les scripts serveurs disposent d'objets (ou de classes) qui facilitent le développement d'applications Web inter-actives.

Ces objets permettent entre autre de récupérer les informations transmises par le client (objets de type ***Requête***), d'envoyer des informations en retour au client (objets de type ***Réponse***), de prendre en compte le contexte utilisateur à travers sa requête http, enfin de suivre le cheminement de l'utilisateur et de stocker des informations qui seront conservées lorsque l'utilisateur passe d'une page à l'autre (objets de type ***Session***).

De même, un objet de type *Server* permet d'accéder aux fonctionnalités et aux caractéristiques du serveur Web.

Enfin l'objet de type ***Application*** -propre à Microsoft et à ses ASP- va permettre de configurer une application Web particulière et de stocker des informations pour tous les utilisateurs de cette application.

* **Les bases de l'ancienne technologie ASP**
* ASP un composant serveur de echnologie propriétaire (Microsoft), combinaison de pages HTML, de scripts (ASP est fourni avec des moteurs de script pour Microsoft® Visual Basic® Scripting Edition (VBScript) et Microsoft® JScript™) et de composants ActiveX comme ADO (composants réseau qui inter agissent entre eux, indépendamment du langage dans lequel ils sont écrits: technologie COM de Microsoft qui définit les interfaces, propriétés et méthodes des objets)
  + - à l’extrême un fichier asp peut ne contenir que du code HTML ou que du code JScript (il sert alors d’"aiguillage" par redirection sur d’autres pages)
* le fichier contient l'HTML du client plus du script interprété à la volée par le serveur - ce script est délimité par les balises *<% %>*
* Les objets intrinsèques les plus utilisés d'ASP sont:

1°) L'objet Request qui sert à accéder à toute information transmise dans une requête HTTP.

Cela inclue les *paramètres* transmis à partir d'un formulaire HTML utilisant soit la méthode POST – dans ce cas dans sa collection **Request.Form**- , soit la méthode GET – dans ce cas dans sa collection **Request.QueryString-,** les *cookies* – dans sa collection **Request.Cookies** - et les *certificats* de client.

Enfin sa méthode **Request.ServerVariables** permet d'exploiter les variables d’environnement de la machine serveur.

Principe d'utilisation de ces collections (Form par exemple):

On récupère la chaîne complète des paramètres par : Request.Form

On vérifie l'existence d'un paramètre par: Request. Form ("nomParamètre").Count != 0

On accède à chaque paramètre par : Request. Form ("nomParamètre")

ou Request. Form ("nomParamètre")(i) si plusieurs de même nom

On peut boucler sur tous les paramètres par : for (X in Request. Form) { X….}

2°) L'objet Response qui sert à contrôler les informations envoyées à l'utilisateur.

Sa collection **Response.Cookies** permet de déposer des cookies sur le client.

Sa méthode Response**.**Write permet d'envoyer des informations au navigateur du client, sa méthode Response**.Redirect** permet de rediriger le navigateur du client vers une autre URL.

3°) L'objet **Server** utilisé pour:

- créer des instances de composants server, type ActiveX, par **Server.CreateObject**

- encoder des chaînes au format URL (avec %codeascii pour les caractères de ponctuation, les accentués … et un + pour l’espace) par **Server.URLEncode**

- encoder des chaînes au format HTML (avec *&lt;* pour *<* etc…) par **Server.HTMLEncode**

- retrouver l’emplacement physique d’un fichier sur le disque serveur par **Server.MapPath** (ex : server.mappath("default.asp") va rendre la chaîne "c:\inetpub\wwwroot\default.asp"

* Tout comme le code "en ligne", les procédures asp sont définies par défaut dans le langage de l’ASP, et entre les balises <% et %>.

Mais les procédures peuvent être définies au choix dans d’autres langages VBScript ou Jscript et doivent alors être entre les balises : – où *runat* évite qu’elles soient exécutées sur le client ! - : **<SCRIPT RUNAT=SERVER LANGUAGE=VBSCRIPT>** et **</SCRIPT>**

On notera qu’entre ces balises, on peut mettre du script directement interprété "en ligne", en sachant qu’il s’exécute comme un sous bloc, mais que les variables qui y seraient déclarées sont locale et perdues pour le reste du programme asp

* Un fichier peut être partagé par plusieurs asp du site et dans ce cas il doit être inclus par l’instruction d’imbrication côté serveur **#include** (**SSI : Server –Side Include statement)**

Les fichiers inclus sont fusionnés dans le fichier cible avant que le script serveur soit traité.

Ajoutez à votre fichier .asp une directive **#INCLUDE FILE** (pour un chemin d'accès relatif au fichier en cours) ou une directive **#INCLUDE VIRTUAL** (pour un chemin d'accès relatif au dossier virtuel de votre application web – site racine / ou "sous site" /appliEtudiant .

* La portée des variables :

Les variables globales ne sont accessibles que dans une seule page ASP.

Pour rendre une variable accessible au-delà de la page concernée, donnez à cette variable **l'étendue session**(disponibilité pour toutes les pages d'une application ASP, lorsqu'elles sont appelées par un utilisateur donné.) ou **l'étendue application** (disponibilité pour toutes les pages d'une application ASP demandées par un n'importe quel utilisateur. ) .

ASP fournit deux objets incorporés dans lesquels vous pouvez stocker ces variables : l'objet **Session** et l'objet **Application.**

* Le fichier **global.asa**

Global.asa est un fichier optionnel et unique, situé à la racine du site ou d'une application : (exemple: wwwroot/global .asa)

Il sert entre autre à inclure des bibliothèques de type pour les composants activeX et à instancier sur le serveur (attribut RUNAT) des objets communs à toute l’application (attribut SCOPE) .

Exemple pour utiliser les constantes et types de données de la bibliothèque ADO d'accès aux bases de données: <!--METADATA TYPE="typelib"

FILE="c:\program files\fichiers communs\system\ado\msado15.dll" -->

Exemple d'instanciation d'un seul objet utilisable par toutes les pages de l'application :

<OBJECT RUNAT=Server SCOPE=Application ID=cpt PROGID="MSWC.PageCounter">

</OBJECT>

Une application ASP doit avoir un fichier global.asa à sa racine.

Celui ci est lu et exécuté au démarrage de l'apploication; à cet effet il définit deux procédures événementielles - Application\_OnStart() et Application\_OnEnd() – qui permettent d'effectuer des actions spécifiques au démarrage ou à l'arrêt de l'application.

Pareillement le suivi de session de IIS permet de gérer deux procédures événementielles au début d'une session (première arrivée d'un intrenaute sur le site) et à la fin d'une session (suite à un timeout d' inactivité de l'internaute). Ces procédures - Session\_OnStart() et Application\_OnEnd() - sont également définies dans le fichier global.asa

On notera que l ‘application doit redémarrer sur le serveur dès qu’une modification est apportée à son fichier global.asa

* **D'ASP vers ASP.NET**

Toute application ASP tourne sur un environnement .NET, et les règles de programmations énoncées plus haut sont transposables à des fichiers aspx d'une application ASP.NET

Cependant pour bien développer en AS¨P.NET, il faut changer ceratines habitudes liées aux anciennes technologies serveur, quelles qu'elles soient. (par exemple, penser à séparer le code de la présentation de la page, introduire des classes métiers, traiter des événements côté clients ou côté serveur, etc …)

* **Principes d' ASP.NET**

Une application ASP.NET est constituée d'un ensemble de fichiers rangés dans un *répertoire virtuel* du serveur Web.

1°) créer un répertoire physique sur c:\inetpub\wwwroot

2°) créer un répertoire virtuel sur la console MMC de IIS, rattaché à ce répertoire physique Noter qu'un répertoire virtuel peut être créé également à partir de l'explorateur de fichiers, depuis l'onglet "Partage Web" de la fenêtre des propriétés d'un répertoire.

3°) créer ou recopier les fichiers d'extensions **.aspx**

4°) possibilité de rajouter dans la racine de l'application un fichier XML **web.config** pour la configuration de l'application (par défaut celui de la racine du site sera pris)

5°) possibilité également de rajouter dans la racine de l'application un fichier **global.asax** (fichier d'application ASP.NET ) pour la gestion de l'application et des sessions.

Le navigateur invoquera alors la page en utilisant le nom du serveur, suivi du nom du répertoire virtuel, suivi d'un éventuel chemin puis du nom du fichier d'extension .aspx

Exemple: utiliser l'URL http://apinet/appliASP-1/premier.aspx pour le fichier de nom physique

C:\Inetpub\wwwroot\Demarrage-ASP-NET\rep-appliASP-1\premier.aspx

* **Mise en œuvre des différentes structures d’une page ASP.NET; exemples**

Créer sous C:\inetpub\wwwroot le répertoire physique sous lequel ranger les fichiers .aspx de l'application, puis le répertoire virtuel (à partir de l'explorateur de fichiers ou de la console MMC de IIS).

Attention: le nom donné au répertoire virtuel est indifférent, mais dans ce cas les URL des pages devront être toujours renseignées avec ce nom virtuel et non avec le nom physique du répertoire; sinon l'application ne fonctionnera pas correctement, notament pour ce qui est du fichier global.asax

**Les exemples suivants – écrits sous notepad - mettent en œuvre:**

1. les différentes balises que l'on rencontre dans une page ASP.NET et l'utilisation des objets intrinsèques issus de la technologie ASP:

**<SCRIPT runat="server"> …. </SCRIPT>** qui définit les blocs de déclarations de variables et de fonctions, et qui peut inclure des procédures exécutées au premier chargement de la page; ce code est compilés à la première invocation de la page

**<% … %>** qui contient du code en ligne, code de "rendu", exécuté à chaque invocation de la page

**<%= … %>** qui contient une expression en ligne, injectée sur la page à chaque affichage

**<@Page …>, <@Import ….>,** etc … différentes directives qui paramètrent la page

**<!---#include …. -->** pour inclure des fichiers externes

**<%--** bla bla …. **--%>** pour mettre des commentaires

Les balises d'instanciation d'objets, les balises de contrôles HTML exécutés côté serveur , les balises de contrôles serveurs spécifiques et les balises de liaisons de données seront vues ultérieurement.

1. Une première utilisation du fichier d'application global.asax et du fichier de configuration web.config (attention à redémarrer l'application en cas de modification de ces fichiers)
2. Quelques particularités du C#

##### C#

Puisqu'un type simple représente l'alias d'un type struct, tous les types simples possèdent des membres. Par exemple, int possède les membres déclarés dans System.Int32 ainsi que les membres hérités de System.Object et les instructions suivantes sont autorisées :

int i = int.MaxValue; // System.Int32.MaxValue constant

string s = i.ToString(); // System.Int32.ToString() instance method

string t = 123.ToString(); // System.Int32.ToString() instance method

***exemples BLJ***

1. **dans Demarrage-ASP-NET\rep-appliASP-1 (répertoire virtuel /appliASP-1/)**

fichiers premier.aspx, deuxieme.aspx, troisieme.aspx et quatrieme.aspx

1. **dans Demarrage-ASP-NET\rep-appliASP-2 (répertoire virtuel /appliASP-2/)**

fichiers cinquieme.aspx, sixieme.aspx, septieme.aspx, huitieme.aspx et neuvieme.aspx

pour traiter des erreurs d'exécution

dixieme.aspx pour l'inclusion de fichiers et onzieme.aspx pour le fichier global.asa

***premier.aspx*** la directive **@Page** pour le choix du langage,

la balise de code en ligne **<% %>** et la balise d'expression en ligne '**<%= %>**

**<%@ Page Language="C#"%>**

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

**<%** for (int i=0; i <7; i++) { **%>**

<font size="**<%=**i**%>**"> Instruction de boucle dans un bloc de code en ligne entre balises &lt;% et %&gt;</font> <br>

**<%** }**%>**

</center>

</body>

</html>

***deuxieme.aspx*** le bloc de code déclaratif **<SCRIPT>** avec sa méthode **Page\_Load**

l'objet **Response** et sa méthode Write

<%@ Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

**<SCRIPT language="C#" runat="server">**

String ch ="";

**void Page\_Load(Object sender, EventArgs e)**

{

for (int i=0; i <7; i++) {

ch += "<font size='" + i + "'> Utilisation d'un bloc de code de déclaration entre balises

&lt;SCRIPT&gt; et d'une expression en ligne &lt;%= %&gt;</font> <br>"; }

}

**</SCRIPT>**

<body>

<center>

**<% Response.Write(ch); %>**

</center>

</body>

</html>

***troisieme.aspx*** mauvaise utilisation du bloc de code par l'instruction Response.Write exécutée au Load de la page, donc avant toute sortie du code HTML !

(regarder le source généré sur le navigateur du client)

<%@ Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center><B> Avant la balise de script<BR>

**<SCRIPT language="C#" runat="server">**

String ch ="";

void Page\_Load(Object sender, EventArgs e)

{

for (int i=0; i <7; i++) {

ch += "<font size='" + i + "'> <U>Mauvaise</U> utilisation d'un bloc de code de déclaration entre balises

&lt;SCRIPT&gt;</font> <br>"; }

**Response.Write(ch); // mauvais choix!!! ici**

}

**</SCRIPT>**

Aprèsla balise de script<BR> </center></B>

</body>

</html>

***quatrieme.aspx*** récupération du paramètre en fin d'URL de requête par l'objet **Request**

conversions de type chaîne – numérique en C#

<%@ Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "<font size='5'> Utilisation des paramètres de la requête HTTP et conversions de types en C#</font> <br>"%>

<BR><BR>

<% int nb=**Int32.Parse**(**Request.QueryString**["toto"]);

nb \*= nb;

Response.Write("Le carré de "+Request.QueryString["toto"]+ " vaut " + nb); // conversion implicite de nb en chaîne

%>

<BR><BR>

<%= nb %>

<BR><BR>

<% string stRes= nb.**ToString();** // conversion explicite nécessaire

Response.Write (stRes);

%>

</center>

</body>

</html>

***cinquieme.aspx*** division sans problème (paramètre toto différent de 0)

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "Gestion des exceptions : division par un nombre différent de 0, pas de problème" %>

<% int nb=Int32.Parse(**Request.QueryString["toto"]);** int val=0;%>

<% val=100/nb;%>

<%= "<BR><font color='red'>Résultat de la division de 100 par le paramètre toto: "+ val.ToString() +"</font>" %>

</center>

</body>

</html>

***sixieme.aspx*** division avec problème (mettre le paramètre toto à la valeur 0)

l' exception est non gérée dans l'aspx et le fichier web.config ne prévoit pas de gestion des erreurs (mode=Off pour la balise <customErrors>)

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "Gestion des exceptions : division par 0, pas de capture de l'exception" %>

<% int nb=Int32.Parse(**Request.QueryString["toto"]);** int val=0;%>

<% val=100/nb;%>

<%= "<BR><font color='red'>Résultat de la division de 100 par le paramètre toto: "+ val.ToString() +"</font>" %>

</center>

</body>

</html>

Contenu de web.config:

<configuration>

<system.web>

<customErrors defaultRedirect=" …." **mode="Off"** >

</customErrors>

</system.web>

</configuration>

***septieme.aspx*** division avec problème (mettre le paramètre toto à la valeur 0)

l' exception est non gérée dans l'aspx et le fichier web.config prévoit une gestion des erreurs (mode=On pour la balise <customErrors>)

la directive @Page renseigne sur le fichier d'erreur à renvoyer

**<%@Page Language="C#" ErrorPage="maPageErreur.htm"%>**

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "Gestion des exceptions : division par 0, pas de capture de l'exception, mais fichier d'erreur perso

et mode=On dans web.config"%>

<% int nb=Int32.Parse(Request.QueryString["toto"]); int val=0;%>

<% val=100/nb;%>

<%= "<BR><font color='red'>Résultat de la division de 100 par le paramètre toto: "+ val.ToString() +"</font>" %>

</center>

</body>

</html>

Contenu de web.config:

<configuration>

<system.web>

<customErrors defaultRedirect=" …." **mode="On"** >

</customErrors>

</system.web>

</configuration>

***huitieme.aspx*** division avec problème (mettre le paramètre toto à la valeur 0)

l' exception est non gérée dans l'aspx et le fichier web.config prévoit une gestion des erreurs (mode=On et defaultRedirect renseigné pour la balise <customErrors>)

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "Gestion des exceptions : division par 0, pas de capture de l'exception, mais fichier d'erreur prévu dans web.config

et mode=On dans web.config" %>

<% int nb=Int32.Parse(Request.QueryString["toto"]); int val=0;%>

<% val=100/nb;%>

<%= "<BR><font color='red'>Résultat de la division de 100 par le paramètre toto: "+ val.ToString() +"</font>" %>

</center> </body>

</html>

Contenu de web.config:

<configuration>

<system.web>

<customErrors **defaultRedirect="http://localhost/Demarrage-ASP-NET/commonASP/errpage.aspx" mode="On**">

</customErrors>

</system.web>

</configuration>

***neuvieme.apsx*** division avec problème (mettre le paramètre toto à la valeur 0)

cette fois l' exception est gérée en C# dans l'aspx, quelle que soit la configuration dans le fichier web.config

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "Gestion des exceptions : division par 0, avec capture de l'exception " %>

<% int nb=Int32.Parse(Request.QueryString["toto"]); int val=0;%>

<% **try** {val=100/nb;%>

<%= "<BR><font color='red'>Résultat de la division de 100 par le paramètre toto: "+ val.ToString() +"</font>" %>

<%} **catch (Exception e)** {Response.Write ("une exception est levée, de message : \""+**e.Message**+"\"");} %>

</center>

</html>

***dixieme.apsx*** les commentaires et les inclusions de fichiers externes

Le fichier Pied.inc à inclure:

<% Response.Write("<BR><HR><BR><B> ASP.NET, c'est le pied ! </B>"); %>

Le fichier qui inclue:

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "<font size='5'> Les commentaires et les inclusions de fichiers externes</font> <br><br>"%>

**<%-- Je vous commente le fait suivant: j'inclus un fichier externe --%>**

par #include file="C:\Inetpub\wwwroot\Demarrage-ASP-NET\commonASP\pied.inc"

**<!-- #include file="C:\Inetpub\wwwroot\Demarrage-ASP-NET\commonASP\pied.inc" -->**

<BR><BR>puis par #include virtual="../Demarrage-ASP-NET/commonASP/pied.inc"

**<!-- #include virtual="../Demarrage-ASP-NET/commonASP/pied.inc" -->**

</center>

</body>

</html>

***onzieme.aspx*** le fichier global.asax et les variables d'application; classes pour la date et l'heure

<html>

<%@Page Language="C#"%>

<html>

<head>

<title>introduction aux asp.net </title>

<link rel="stylesheet"href="intro.css">

</head>

<body>

<center>

<%= "<font size='5'> Le fichier global.asax et les variables d'application</font> <br>"%>

<%-- Utilisation d'un variable créée au démarrage de l'application grâce au fichier global.asax--%>

<%= "Cette application 'appliASP-2' a démarré le " + **Application["Date"]** + " à " + **Application["Heure"]** %>

</center>

</body>

</html>

Le fichier global.asax contient la définition de la procédure de démarrage de l'application

<Script language="C#" runat="server">

public void **Application\_OnStart()** {

// Application start-up code goes here.

DateTime dt=DateTime.Now;

string sd = dt.ToLongDateString(); // ou : string sd = dt.DayOfWeek+" "+dt.Day+" "+dt.Month+" "+dt.Year;

**Application["Date"]** = sd;

string st = dt.ToShortTimeString();

**Application["Heure"]** = st;

}

</script>

#### Remarque

#### Tout changement du fichier global.asax ou du fichier web.config doit être suivi d'un redémarrage de l'application web concernée.

#### Ce redémarrage est fait automatiquement lorsque l'on travaille avec l'IDE Visual Studio. Sinon il devra être fait "à la main".

#### Rappel

#### Le fichier global.asax et le bon fonctionnement de l'application imposent la création d'un répertoire virtuel sur le répertoire physique où sont stockés les fichiers de l'application.

#### CREATION D'UNE APPLICATION AU MOYEN D'UN PROJET SOUS VISUAL STUDIO .NET

Visual Studio a été entièrement remanié de manière à standardiser les méthodes de développement à destination des deux environnements qui cohabitent de plus en plus étroitement aujourd'hui :

* Les applications Windows.
* Les applications Internet.

Dans les deux cas, une application se compose de fenêtres nommées **Forms**. Pour les applications Windows, il s'agit de **WinForms** tandis qu'il s'agit de **WebForms** pour les applications dédiées au Web. Mais le développement est similaire dans les deux cas.

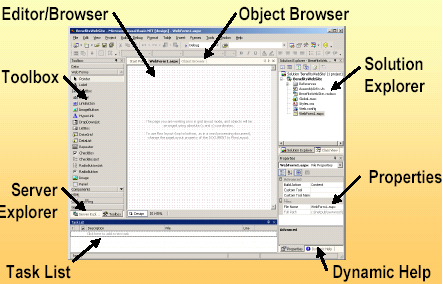
Un projet Web peut être une application Web ASP.NET standard ou un service Web XML destiné à être consommé par d'autres applications. Le serveur Web qui héberge le projet peut être installé sur un ordinateur local ou distant

* **Principes**

Un projet ASP.NET est créé sous l'environnement Visual Studio .NET qui se charge alors de créer les répertoires physiques et virtuels.

1°) créer un nouveau projet par le menu fichier, en choisissant le type **"Projets Visual C#"** et le modèle **"Applications Web ASP.NET"**

Choisir comme emplacement une URL du serveur web à laquelle on rajoute un nom pour le projet: ex: http://localhost/Demarrage-ASP-NET/WebApplication1



2°) on constate:

qu'un répertoire physique et différents fichiers sont créé sur le disque du serveur :

### C:\Inetpub\wwwroot\Demarrage-ASP-NET\WebApplication1

et qu'un répertoire virtuel est créé dans la console MMC de IIS:

WebApplication1 rangé dans le folder "Site Web par défaut / Demarrage-ASP-NET"

et qu'un répertoire physique est créé pour stocker le fichier texte " WebApplication1.sln" de définition de la solution du projet:

C:\Documents and Settings\blj\Mes documents\Projets Visual Studio\WebApplication1

3°) modifier le fichier WebForm1.aspx en insérant une phrase HTML du type "Hello le premier projet ASP.NET" et **surtout penser à générer le projet par le menu "générer"** - faute de quoi les fichiers **.cs** de codes sources des classes du projet ne sont pas compilés.

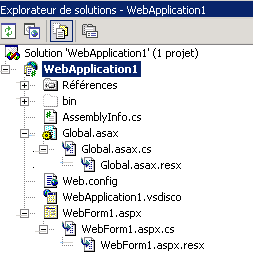
4°) Tester l'application créée à l'URL

http://apinet/Demarrage-ASP-NET/WebApplication1/WebForm1.aspx

* **Les fichiers du projet**

### Ces fichiers sont localisés sous la racine du serveur, et sont visibles dans la fenêtre de l'explorateur de projet

Des pages Web Forms permettent de créer l'interface utilisateur d'une application Web ASP.NET



**1°)** Une page "WebForm1" est automatiquement créée, dont l'URL est

**http://apinet/Demarrage-ASP-NET/WebApplication1/WebForm1.aspx**

Cette page est matérialisée par trois fichiers rangés sous

**C:\Inetpub\wwwroot\Demarrage-ASP-NET\WebApplication1**:

* **WebForm1.aspx** qui est le fichier de code HTML du formulaire web et contient les éléments visuels de la page; ce fichier est visible et modifiable en mode design ou en mode édition

**<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false" Inherits="WebApplication1.WebForm1" %>**

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html>

<head>

<title>WebForm1</title>

<meta name="GENERATOR" Content="Microsoft Visual Studio .NET 7.1">

<meta name="CODE\_LANGUAGE" Content="C#">

<meta name=vs\_defaultClientScript content="JavaScript">

<meta name=vs\_targetSchema content="http://schemas.microsoft.com/intellisense/ie5">

</head>

<body MS\_POSITIONING="GridLayout">

<H3> Hello le premier projet ASP.NET </H3>

<form id="Form1" method="post" runat="server">

</form>

</body>

</html>

* **WebForm1.aspx.cs** qui est le fichier **"code behind"** (ou code "en coulisse") qui contient la logique de programme de la page (les sources d'initialisation de la page et de gestion de ses événements). C'est écrit en source C# (extension .cs pour C Sharp) et on remarque que la page dérive une classe Page qui sera instanciée en tant qu'objet Page, mis en cache dans la mémoire du serveur:

**public class WebForm1 : System.Web.UI.Page**

On remarque également que la page .aspx hérite de la classe du code-behind : l'attribut **Inherits** de la directive **@ Page** est utilisé pour spécifier le fichier de classe à partir duquel le fichier .aspx est dérivé.

using System;

using System.Collections;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Web;

using System.Web.SessionState;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.HtmlControls;

namespace WebApplication1

{

/// <summary>

/// Description résumée de WebForm0.

/// </summary>

**public class WebForm1: System.Web.UI.Page**

{

private void Page\_Load(object sender, System.EventArgs e)

{

// Placer ici le code utilisateur pour initialiser la page

}

#region Code généré par le Concepteur Web Form

override protected void OnInit(EventArgs e)

{

//

// CODEGEN : Cet appel est requis par le Concepteur Web Form ASP.NET.

//

InitializeComponent();

base.OnInit(e);

}

/// <summary>

/// Méthode requise pour la prise en charge du concepteur - ne modifiez pas

/// le contenu de cette méthode avec l'éditeur de code.

/// </summary>

private void InitializeComponent()

{

this.Load += new System.EventHandler(this.Page\_Load);

}

#endregion

}

}

* **WebForm1.aspx.resx** qui est le fichier XML de ressources (données, images …) de la page. Ces ressources peuvent être modifiées (critères de "culture" qui modifient la langue …) sans entraîner de recompilation de la page WebForm1.aspx

Ces deux derniers fichiers sont visibles dans l'explorateur de projet en ouvrant le "folder" WebForm1.aspx ou par l'icône "tous les fichiers" de l'explorateur de projet.

* Les fichiers de classe code-behind .cs de toutes les pages du projet sont compilés dans un unique fichier **assembly** - bibliothèque de liaisons dynamiques (.dll) - généré par le projet:

**C:\Inetpub\wwwroot\Demarrage-ASP-NET\WebApplication1\bin\WebApplication1.dll**

* Le fichier de page .aspx Web Forms est également compilé, mais différemment.

Qu'est ce que l'assembly?

Qu'il s'agisse du fichier .exe d'un projet Application Window ou du fichier .dll d'un projet Web, celui ci regroupe tous les types – ou classes- du projet.

Il se nomme du nom du projet et se compose de trois parties:

- le manifeste, qui décrit les types du projet et les références sur les types contenus dans d'autres assemblys

- des métadonnées et les informations de version et de déploiement.

- le code compilé en langage intermédiaire (40% seulement de l'assembly!)

Le contenu de l'assembly se visualise grâce à l'utilitaire C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin\ **ildasm.exe** qui peut être rajouté comme outil dans Visual Studio (menu "outils" -> "outils externes") - <CTRL><M> sous ildasm montre le manifeste en détails.

On peut constater les directives du style " .assembly extern mscorlib" du manifeste qui signale à la CLR le besoin de l'assembly nommé mscorlib.dll – la principale du framework!- ; ainsi avant d'exécuter un programme, la CLR peut-elle vérifier si toutes les dll requises sont bien là.

La recherche des assemblys dll se fait

- dans le même répertoire que l'assembly du projet

- dans des sous répertoires .fr, .us etc selon les critères de culture

- dans le répertoire **GAC** (global assembly cach) généré à l'installation du framework.

Comment est géré le GAC?

C'est le répertoire dans lequel se retrouvent toutes les assemblys – ou dll- du framework.

Il s'agit de C:\WINNT\assembly dans lequel on trouve par exemple un pseudo fichier System.Web

En fait System.WEB ets lui même un répertoire : C:\WINNT\assembly\GAC\System.Web\1.0.5000.0\_\_b03f5f7f11d50a3a dans lequel on trouve cette fois le fichier System.Web.dll. On remarquerait que si l'on installait une nouvelle version du framework un deuxième répertoire numéroté 1.x.x ….. serait créé.

La particularité du framework .NET est donc ne ne pas nécessiter d'enregistrement des dll dans la base de registre, et de faire cohabiter toutes les versions sans conflit; c'est "la fin de l'enfer des dll" selon Microsoft.

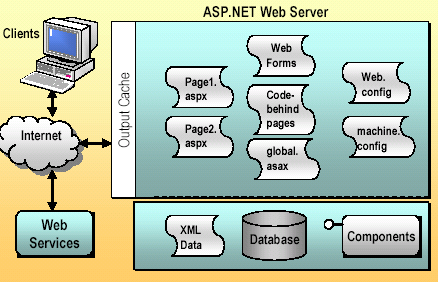
Conclusion :Bien que les pages Web Forms soient composées de deux fichiers distincts (.aspx et .aspx.cs), elles constituent une entité unique lors de l'exécution de l'application.

Remarque: Il est également possible de créer les éléments visuels et le code dans le même fichier, parfois appelé page Web Forms « à fichier unique » - ce qui a été fait dans les exemples précédents avec le notepad. Mais cela réduit la prise en charge par Visual Studio.

**2°)** Un fichier **global.asax** qui peut être utilisé en mode"design" pour ajouter des composants (d'accès à l'Active Directory, de messagerie, de surveillance du disque …) et en mode "code" pour éditer le fichier **global.asax.cs** qui contient les définitions des variables et procédures événementielles gérées au niveau de l'application et des sessions (start, stop, error). Il y a également un fichier XML **global.asax.resx** pour les ressources de l'application.

**3°)** Un fichier XML **Web.config**, qui contient les paramètres de configuration de l'application (gestion des erreurs, mode debug, authentification, statut des sessions …)

**4°)** Un fichier de code source **AssemblyInfo.cs** qui contient des définitions d'attributs (méta données) tels que la nom de la société, la culture, la version …. et qui sont globaux à l'assembly et que le développeur peut personnaliser.



* **mécanismes objet et événementiel**

On constate qu'un espace de noms ("**name space**") appelé *WebApplication1* (du nom du projet) a été implicitement créé lors de la génération du projet.

Un espace de nom est un conteneur de types que l'on peut utiliser dans le programme: des classes, des interfaces, des structures, des énumérations . Ces espaces sont organisés hiérarchiquement et fonctionnellement.

Références aux espaces de noms .NET Framework suivants :

*System* - Contient des classes fondamentales et des classes de base qui définissent les types de données référence et valeur, les événements et gestionnaires d'événements, les interfaces, les attributs et le traitement des exceptions courants.

*System.Data* - Comprend des classes qui constituent l'architecture ADO.NET, qui est la principale méthode d'accès aux données pour les applications managées.

*System.XML* - Contient des classes qui assurent une prise en charge basée sur des normes pour le traitement XML.

*System.Web* - Comprend des classes et des interfaces qui permettent la communication entre le navigateur et le serveur;au niveau inférieur, *System.Web.Mail* contient des types pour envoyer des messages, alors que Sys*tem.Web.SessionState* fournit des types qui permettent le stockage de données spécifiques à une session client particulier et que *System.Web.UI* fournit des types qui permettent de créer des contrôles et des pages qui s'affichent sous forme d'interface utilisateur.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpref/html/cpref\_start.htm

* Ainsi le fichier WebForm1.aspx.cs de code-behind de la page WebForm1 comprend-il les directives d'utilisation de types dans des espaces de noms (**directives "using"**).

De cette manière, il n'y a pas à qualifier l'utilisation d'un type dans le code source si ce type fait partie d'un de ces espaces de noms.

Par exemple la directive "using System.Web.UI;" permettra d'utiliser sa classe Control en la nommant simplement "Control" et non pas " System.Web.UI.Control".

La notation entièrement qualifiée reste utile dans le cas où deux espaces de noms présenteraient des classes de même nom.

On remarque qu'un espace de nom **WebApplication1** est créé dans le source de la page pour indiquer la portée de la classe WebForm1 qui va être créée par héritage de la classe de **System.Web.UI.Page** fournie par l'espace de nom System.Web.UI

En effet la classe Page est associée aux fichiers ayant une extension .aspx :

namespace WebApplication1

{

public class WebForm1 : System.Web.UI.Page // héritage

{

private void Page\_Load(object sender, System.EventArgs e)

{ // Placer ici le code utilisateur pour initialiser la page

}

}

}

C'est à partir de cette classe **WebApplication1.WebForm1** que sera créé en mémoire un objet Page lors de la demande de la page WebForm1.aspx par le serveur suite à une requête utilisateur.

On remarque également la méthode privée **Page\_Load( )** qui permet de placer du code au chargement de la page, tout comme la procédure événementielle décrite dans les exemples de fichiers .aspx vus précédemment:

<SCRIPT language="C#" runat="server">

void Page\_Load(Object sender, EventArgs e) { ….. }

</SCRIPT>

Remarque: On constate que cette méthode Page-Load() n'est pas fournie par la classe System.Web.UI.Page dont dérive notre classe WebForm1; en effet il y a une reconnaissance spéciale des six méthodes suivantes : **Page\_Init , Page\_Load , Page\_DataBind , Page\_PreRender , Page\_Dispose et Page\_Error**

Ces **méthodes sont automatiquement connectées aux gestionnaires d'événements** pour les événements de la page standard.

* De même le fichier WebForm1.aspx de la page WebForm1 comprend la directive **<%@ Page Inherits="WebApplication1.WebForm1">** qui spécifie la classe à instancier pour exécuter la page.

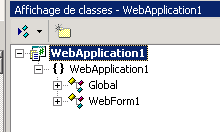
Pour ce faire il inclut la directive **<%@ Page Codebehind="WebForm1.aspx.cs">** qui renseigne sur le nom du fichier de code-behind où chercher entre autre la définition de cette classe WebApplication1.WebForm1

* Les attributs Inherits et Codebehind se retrouvent également pour la directive d'application du fichier d'application Global.asax:

**<%@ Application Codebehind="Global.asax.cs"**

**Inherits="WebApplication1.Global" %>**

Le fichier Global.asax est analysé et compilé dans la classe WebApplication1.Global qui est dérivée de la classe de base HttpApplication du Framework .NET.



Les 2 seules classes du projet, obtenues par dérivation de classes de la librairie du Framework .NET

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/vbconWebApplicationTemplate.htm

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/vbconWebApplicationProjects.htm

* **La mise au point par le débogueur**

Une première technique de débogage consiste à ajouter dans le code des appels à des fonctions de sortie sur écran ou sur fichier tetxe- telles que **printf** ou **MsgBox**. C'est la technique des "**traces**".

Mais un débogueur permettra d'observer le déroulement du programme sans en modifier son code.

Les fonctions de débogage intégrées de l'environnement de développement Visual Studio permettent d'observer à partir de sons source le comportement d'un programme au moment de son exécution.

* s'arrêter à l'emplacement d'une procédure en insérant un point d'arrêt sur l'instruction souhaitée ou en demandant un démarrage jusqu'au curseur
* inspecter les variables et les contrôles graphiques, et modifier leurs valeurs par la fenêtre Espion, la boîte de dialogue Espion express et la fenêtre Mémoire.
* redémarrer l'application jusqu'au prochain arrêt, ou en exécutant un seul pas (principal ou détaillé)

N'hésitez pas à utiliser le débogueur!!!

A signaler qu'un modèle objet du débogueur – avec ses propriétés, méthodes ….- permet d'automatiser de nombreuses fonctionnalités du débogueur.

#### CREATION D'UN PREMIER FORMULAIRE WEB

Un formulaire Web (page Web Forms) est une page ASP.NET qui contient des composants de contrôle visuels associés à un mécanisme de programmation événementiel.

* **Principes**

On a vu que l'événement **Page\_Load** survient à chaque fois que la page est demandée. Cet événement est souvent associé à des restaurations de statut de la page et à des connexions à des bases de données.

La propriété **IsPostBack** de l'objet Page permet de savoir si la page est demandée suite à une action "Post" du client ou si elle est demandée pour la première fois. Elle est très utilisée dans le code de Page\_Load.

L'événement **Page\_Unload( )** survient inversement lorsque la page a été affichée et qu'elle va être transmise au "garbage collector".

Les autres événements qui peuvent survenir dans la page sont en relation avec le traitement des formulaires par l'utilisateur (par exemple un clic sur bouton).

Si le formulaire ou le composant graphique qui reçoit un événement doit être traité par le serveur (attribut **runat="server"**), la page est à nouveau activée.

Ainsi dans les pages Web Forms, les événements associés à des contrôles serveur sont déclenchés sur le client mais gérés sur le serveur Web.

Cependant tous les événements ne sont pas traités par le serveur: imaginez que les "mouseOver" lui soient renvoyés, il y aurait un trafic incroyable sur le réseau. Les Web Forms traitent principalement des événements clic.

Les pages Web Forms sont initialisées, traitées, puis supprimées à chaque aller-retour

Ce mécanisme d'allers retours nécessite donc que le statut de la page soit conservé; d'autre part le formulaire sait distinguer grâce à un champ caché (**hidden**) s'il s'agit de son premier affichage ou d'un retour après traitement d'événement.

Les composants que l'on peut trouver dans un formulaire:

Un composant classique qui sera traité par le navigateur de l'utilisateur :

**<INPUT type="button" value="OK" id="bt" >**

Un composant classique qui sera traité par le navigateur de l'utilisateur et ensuite par le serveur web. Chaque balise HTML a une classe correspondante dans la librairie du Framework .NET (dans cet exemple, la classe HtmlInputButton de l'espace de noms **System.Web.UI.HtmlControls**).

**<INPUT type="button" value="OK" id="bt" runat="server" />**

Un composant de contrôle Web qui sera présenté dans la page par du code HTML (qui dépendra d'ailleurs du navigateur de l'utilisateur) mais qui sera traité uniquement par le serveur Web (dans cet exemple la classe Button de l'espace de noms **System.Web.UI.WebControls**).

**<asp:Button id="bt" runat="server" Text="OK"></asp:Button>**

On notera que l'espace de nom System.Windows fournit les classe de contrôles graphiques pour les applications Windows.

* **Mise en œuvre : composition graphique du formulaire**

1°) Dans l'explorateur de projet, activer le fichier WebForm1.aspx et choisir l'**onglet "Design" du concepteur WYSIWYG** pour composer graphiquement la page.

2°) Ouvrir la boîte à outils et vérifier qu'elle comporte 2 compartiments de contrôles IHM:

* les contrôles HTML classiques et que l'on peut insérer sur la page en les positionnant sur la grille (remarquer le code HTML: <**body MS\_POSITIONING="GridLayout">)**
* les contrôles Web Forms qui encapsulent des éléments d'interface utilisateur HTML communs et se positionnent pareillement; ils s'exécutent sur le serveur Web, mais présentent leur interface utilisateur au navigateur sous forme de code HTML.

Par exemple, insérer dans la page un bouton HTML puis un bouton Web Forms puis remarquer les 2 balises HTML qui ont été mises dans le formulaire:

<form id="Form1" method="post" runat="server">

<INPUT style="Z-INDEX: 101; LEFT: 191px; POSITION: absolute; TOP: 94px"

type="button" value="Button">

<asp:Button id="Button1" style="Z-INDEX: 102; LEFT: 339px;

POSITION: absolute; TOP: 112px" runat="server" Text="Button"></asp:Button>

</form>

On constate que le formulaire et le bouton Web Forms ont un attribut *runat="server"*

Tous les contrôles Web Forms commencent par la balise **<asp:** et se terminent par **</asp:** (ex: asp:Label, asp:Table, asp:RadioButton, asp:Image …)

On constate également dans le fichier de code-behind WebForm1.aspx.cs :

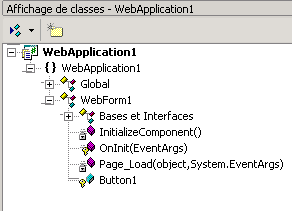
* que deux directives supplémentaires ont été mises afin d'utiliser les espaces de noms nécessaires à la gestion de ces composants:

using System.Web.UI.WebControls;

using System.Web.UI.HtmlControls;

* qu'un objet de type Button a été déclaré comme donnée membre (ou propriété) dans la classe de la page:

protected System.Web.UI.WebControls.Button Button1;



La classe WebForm1 de la page

Possède une nouvelle propriété qui est l'objet bouton

3°) Tester la page sous le navigateur

Deux boutons apparaissent identiques à l'œil, mais le source de la page affichée est le suivant:

(1)

(4)

(2)

(3)

(5)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<HTML>

<HEAD>

<title>WebForm1</title>

<meta content="Microsoft Visual Studio 7.0" name="GENERATOR">

<meta content="C#" name="CODE\_LANGUAGE">

<meta content="JavaScript" name="vs\_defaultClientScript">

<meta content="http://schemas.microsoft.com/intellisense/ie5" name="vs\_targetSchema">

</HEAD>

<body MS\_POSITIONING="GridLayout">

Hello le premier formulaire ASP.NET

<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">

<input type="hidden" name="\_\_VIEWSTATE"

value="dDwtMTc0MDc5ODg1Mzs7PjqkTgPWFH5Uc4LyOSLLUACAvwRU" />

&nbsp;

<INPUT style="Z-INDEX: 101; LEFT: 96px; POSITION: absolute; TOP: 78px" type="button"

value="Button">

<input type="submit" name="Button1" value="Button" id="Button1" style="Z-INDEX: 102;

LEFT: 296px; POSITION: absolute; TOP: 103px" />

</form>

</body>

</HTML>

On constate:

(1): que le formulaire HTML n'a plus l'attribut *runat="server"* et qu'une méthode *post* lui a été

jointe avec la page elle même comme action

(2): que le bouton HTML classique est inchangé

(3): que le bouton asp Web Forms a été traduit en bouton HTML de type *submit* afin qu'il serve à

ré invoquer la page

(4): enfin qu'une balise de type *hidden* permettra au formulaire de renvoyer lors du post une trace

pour le suivi de l'état de la page.

(5): remarquer le terminateur */>*

On vérifie que si le positionnement absolu des deux boutons (grâce à la balise <body MS\_POSITIONING="GridLayout">) fonctionne bien sous I.E. et Netscape 6, il n'en est pas de même pour des versions plus anciennes.

4°) Sous l'onglet "Design" sélectionner le bouton Web Forms et modifier par la **fenêtre des propriétés** son *texte* et son *id*; cela se répercute sur les attributs de la balise <asp:Button> du fichier WebForm1.aspx :

<asp:Button id="btnTest" runat="server" Text="Test"></asp:Button>

et donc aussi sur l'affichage et le source HTML dans le navigateur.

<input type="submit" name="btnTest" value="Test" id="btnTest">

5°) Dans le fichier WebForm1.aspx, rajouter sous forme d'un script serveur une fonction qui traitera les **événements** **clic** sur le bouton Web Forms

<asp:Button id="btnTest" onClick="clic\_test" ….

par exemple changer le texte du bouton;

Rappel: cette fonction ne peut être exécutée que sur le serveur (attribut runat) puisqu'il s'agit d'un composant de l'espac de nom System.Web.UI.WebControls

<SCRIPT **runat="server">**

void clic\_test (object sender, System.EventArgs e) { btnTest.Text = "bouton traité"; }

</SCRIPT>

objet ayant déclenché l'événement objet event contenant des

informations propres à l'événement.

Tester sous navigateur le comportement de la page ainsi que son source.

* + au premier chargement la page est identique (rien dans le source qui laisse supposer un traitement au clic)
  + à l'appel suivant on constate que le texte du bouton s'est bien modifié ainsi que le source HTML qui présente cette fois la balise du bouton sous la forme:

<input type="submit" name="btnTest" value="bouton traité" id="btnTest" …

C'est donc bien le serveur qui a traité l'événement et qui a décidé que les conséquences en seraient la modification de l'attribut value qui fixe le texte de la balise HTML.

L'état de la page a été modifié, et même si celle ci est rechargée par le navigateur, elle reste dans ce dernier état.

6°) Sous l'onglet "Design" sélectionner le bouton Web Forms, puis sélectionner dans la fenêtre des propriétés du bouton l'icône "Evénements" et double cliquer sur la ligne "Click"; une procédure événementielle construite à partir du nom du bouton est générée.

On constate dans le fichier de code-behind WebForm1.aspx.cs qu'une méthode de la classe de la page est prête à gérer l'événement clic de ce bouton:

private void btnTest\_Click(object sender, System.EventArgs e) { }

et qu' un handler d'événement est renseigné par cette méthode à l'initialisation des composants de la page (noter le "+=" qui rajoute cette méthode au handler):

private void InitializeComponent()

{

this.btnTest.Click **+=** new System.EventHandler(this.btnTest\_Click);

………….

Définir le corps de la méthode de la sorte:

private void btnTest\_Click(object sender, System.EventArgs e)

{ btnTest.Text += " deux fois"; } // on rajoute du texte sur le texte du bouton

surtout **penser à régénérer le projet**

puis tester sous navigateur le comportement de la page ainsi que son source.

* + au premier chargement la page est identique et le bouton s'appelle "Test"
  + à l'appel suivant on constate que le texte du bouton s'est bien modifié ainsi que le source HTML qui présente cette fois la balise du bouton sous la forme:

<input type="submit" name="btnTest" value="bouton activé deux fois" id="btnTest"

On peut donc en déduire que le traitement de l'événement clic s'est fait en deux étapes:

1- le handler décrit dans le script du fichier WebForm1.aspx

2- puis le handler décrit dans le code du fichier WebForm1.aspx.cs

* **une première conclusion**

En fait la séparation des traitements dans ces deux fichiers est volontaire et est un des aspects essentiels de la programmation sous le Framework .NET : **dissocier le code de présentation de la page** (qui concerne le fichier WebForm1.aspx) **de la logique de traitement** (qui concerne le fichier WebForm1.aspx.cs qui doit être régénéré "à la main").

Cette séparation facilite la maintenance et renforce la protection des pages.

L'exemple qui vient d'être traité n'a donc pas de sens!

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/vbconWebFormsEventModel.htm

D'autre part on constate que tous les contrôles qui ont l'attribut "runat=server" sont gérés par serveur, et qu'alors la valeur associée qui a été saisie dans le navigateur du client est accessible directement par leur propriété .Value, . SelectedIndexChanged, etc … , ce qui évite d'avoir à utiliser la collection des paramètres de la requête http (Request.Form) comme en asp classique.

#### LES EVENEMENTS DES CONTROLES ASP.NET

Ce chapitre est entièrement issu de la documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconwebformseventmodel.htm

*Dans des pages Web Forms, les événements associés à des contrôles serveur sont déclenchés sur le client mais gérés sur le serveur Web par l'infrastructure de page ASP.NET.*

Pour des événements déclenchés sur le client, le modèle d'événement de contrôles Web Forms nécessite que les informations sur les événements soient capturées sur le client et qu'un message d'événement soit transmis au serveur, via un protocole HTTP POST. L'infrastructure de page doit interpréter la **publication** pour déterminer quel événement s'est produit, puis appeler la méthode appropriée dans votre code sur le serveur pour gérer l'événement.

Dans des contrôles serveur Web, certains événements, notamment les événements **Click**, entraînent la publication du formulaire sur le serveur.

Les événements de modification dans des contrôles serveur HTML et des contrôles serveur Web, tels que le contrôle TextBox, sont capturés mais n'entraînent pas une publication immédiate. Ils sont plutôt mis en cache par le contrôle en attendant la prochaine publication. Ensuite, lorsque la page est de nouveau traitée sur le serveur, tous les événements en attente sont déclenchés et traités.

Vous pouvez spécifier qu'un événement de modification entraîne la publication d'un formulaire. Les contrôles serveur Web qui prennent en charge un événement de modification comportent une propriété **AutoPostBack**. Lorsque cette propriété a la valeur True, l'événement de modification du contrôle entraîne la publication immédiate du formulaire, sans attendre un événement Click (voir les exemples du chapitre suivant).

La liaison entre le message d'événement et une méthode spécifique (c'est-à-dire un gestionnaire d'événements) s'effectue à l'aide d'un **délégué d'événement**.

Dans Visual C#, vous liez des événements à des méthodes en ajoutant au fichier de code-behind un délégué de gestionnaire d'événements explicite, comme ci-dessous :

private void InitializeComponent()

{

this.Load **+=** new System.EventHandler(this.Page\_Load);

}

L'infrastructure de page ASP.NET prend également en charge une façon automatique de lier des méthodes et des événements de page. Si l'attribut **AutoEventWireup** de la directive Page a la valeur True (ou s'il n'est pas défini, puisqu'il a la valeur True par défaut), l'infrastructure de page appelle des événements de page automatiquement, en particulier les méthodes Page\_Init et Page\_Load. Dans ce cas, aucun délégué ni aucune clause Handles explicites ne sont nécessaires (voir les exemples du chapitre suivant).

L'inconvénient de l'attribut AutoEventWireup est qu'il nécessite que les gestionnaires d'événements de page possèdent des noms spécifiques prévisibles. Le choix des noms que vous pouvez attribuer à vos gestionnaires d'événements est donc limité.

L'événement **onclick** client pose un problème si vous souhaitez le gérer à la fois sur le client et le serveur. Le problème est dû au fait que tous les contrôles serveur bouton (et d'autres contrôles dont la propriété AutoPostBack a la valeur True) envoient la page au serveur.

Vous pouvez créer des gestionnaires OnClick côté client pour tous les contrôles, mais vous devez choisir la stratégie qui fonctionne avec chaque type de contrôle. Le tableau suivant récapitule les stratégies à utiliser pour les différents types de contrôles.

|  |  |
| --- | --- |
| Contrôle | Stratégie |
| HtmlInputButton, qui contient des boutons de contrôle serveur HTML dont le type est Submit, Reset ou Image. | Ajoutez un attribut onclick dans la syntaxe HTML du contrôle :  <INPUT Type="Submit" Runat="Server" Value="*caption*" onclick="clientfunction()"  ...>  Côté serveur, ces types de boutons déclenchent un événement ServerClick au lieu d'un simple événement Click. L'événement client est déclenché en premier, puis le formulaire est envoyé et l'événement serveur est géré. |
| Tous les autres contrôles HTML (ceux qui n'envoient pas de formulaire par défaut). | Ajoutez un attribut onclick dans la syntaxe HTML du contrôle, en le faisant suivre d'un point-virgule (;) :  <INPUT Type="Button" Runat="Server" Value="*caption*" onclick="clientfunction();"  ...>  Votre fonction est ainsi appelée avant le script d'envoi côté client. |
| Les contrôles serveur Web, qui contiennent des boutons (<asp:button>) et d'autres contrôles (par exemple, <asp:checkbox>). | Vous ne pouvez pas spécifier d'événement côté client dans la syntaxe HTML pour un contrôle serveur Web. Ajoutez plutôt un attribut d'événement au contrôle au moment de l'exécution dans le code serveur, comme ci-après :  Button1.Attributes.Add("onclick", "clientfunction();")  Remarque   Pour le contrôle serveur Web Button, le point-virgule n'est pas nécessaire, parce que ce contrôle envoie automatiquement la page. |

#### LES COMPOSANTS ET CONTROLES DES PAGES WEB FORMS

* **System.Web.UI.HtmlControls**

L'espace de noms **System.Web.UI.HtmlControls** est une collection de classes qui permet de créer des contrôles serveur HTML sur une page Web Forms. Les contrôles serveur HTML s'exécutent sur le serveur et sont directement mappés vers des balises HTML standard prises en charge par la plupart des navigateurs. Cela permet de contrôler par programme les éléments HTML d'une page Web Forms. Les contrôles serveur HTML doivent résider dans une balise conteneur **<form>** avec l'attribut **runat="server"**.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpref/html/frlrfSystemWebUIHtmlControls.htm

1°) Reprendre la page WebForm1.aspx du projet et lui ajouter une boîte de sélection en écrivant le code HTML suivant dans le fichier WebForm1.aspx (ou prendre une ListBox dans la boîte à outils HTML):

<form id="Form1" method="post" runat="server">

<select id="Select1" runat="server"NAME="Select1">

<option value="1" Selected> bleu</option>

<option value="2"> vert</option>

<option value="3"> rouge </option>

<option value="4"> jaune </option>

<option value="5"> noir </option>

<option value="6"> orange </option>

</select>

<asp:Button id="btnTest" onClick="clic\_test" style="Z-INDEX: 102; LEFT: 220px;

POSITION: absolute; TOP: 124px" runat="server" Text="Test"></asp:Button>

</form>

Les balises sont renseignées par l'attribut **runat="server",** ce qui a pour effet de renvoyer un traitement sur le serveur lorsque se produit un clic sur le bouton Test, le source de la page affichée par le navigateur étant le suivant:

(1)

(2)

(3)

<body MS\_POSITIONING="GridLayout">

Hello le premier formulaire ASP.NET

<HR>

Entrez votre couleur préférée

<br>

<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">

<input type="hidden" name="\_\_VIEWSTATE"

value="dDwyMDAwOTE5MTczOzs+eFqrRRAN5MwkrnVFpRqTTWjA1oQ=" />

<select name="Select1" id="Select1">

<option **selected="selected"** value="1"> bleu</option>

<option value="2"> vert</option>

<option value="3"> rouge </option>

<option value="4"> jaune </option>

<option value="5"> noir </option>

<option value="6"> orange </option>

</select>

<input type="submit" name="btnTest" value="Test" id="btnTest"

style="Z-INDEX: 102; LEFT: 220px; POSITION: absolute; TOP: 124px" />

</form> </body>

On constate:

(1): que le formulaire HTML n'a plus l'attribut *runat="server"* et qu'une méthode *post* lui a été

jointe avec la page elle même comme action

(2): que la balise HTML classique select a été générée

(3): que le bouton asp Web Forms a été traduit en bouton HTML de type *submit* afin qu'il serve à

ré invoquer la page

Choisir une couleur et effectuer le clic sur le bouton de test:

On s'aperçoit que bien que la page aie été rechargée par le serveur, elle pré sélectionne la valeur choisie précédemment (le jaune):

<select name="Select1" id="Select1">

<option value="1">bleu</option>

<option value="2">vert</option>

<option value="3">rouge</option>

<option **selected="selected"** value="4">jaune</option>

<option value="5">noir</option>

<option value="6">orange</option>

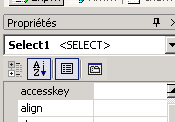
</select>

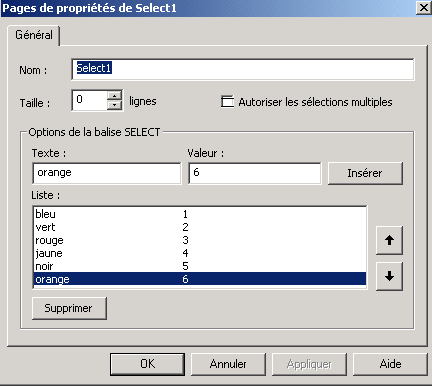
L'état de la page a été modifié, et même si celle ci est rechargée par le navigateur, elle reste dans ce dernier état.

2°) Effectuer la génération du projet et constater qu'un objet a été déclaré dans la classe WebForm1 du fichier WebForm1.aspx.cs:

protected **System.Web.UI.HtmlControls.HtmlSelect** Select1;

3°) Sélectionner le composant sous l'onglet Design qui permet de composer la page; sa fenêtre de propriétés permet alors d'ouvrir une page des propriétés:





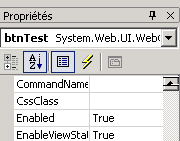
On remarquera que le composant HTML

select n'offre pas d'icône pour gérer

des événements comme l'a fait le

composant asp bouton (qui n'a pas de

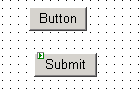
page de propriété!)



* **une première conclusion**

Les composants insérés dans le formulaire à l'aide de la boîte à outils HTML sont des composants classiques, dont les événements seront traités par le client, mais qui peuvent être initialisés par le serveur au chargement de la page du fait que des objets des classes de l'espace System.Web.UI.HtmlControls leur sont associés.

Attention, pour que ces composants HTML soient accessibles côté serveur, il faut qu'ils soient identifiés (ce nom sera la variable objet) et qu'ils aient l'attribut runat="server".



On peut remarquer qu'une flèche verte signale dans le composeur de la page qu'un objet du code-behind est associé au composant Submit

Cet exemple:

<form id="WebForm1" method="post" runat="server">

<INPUT runat="server" type="checkbox" id="check1">

<INPUT runat="server" id="radio1">

<SELECT runat="server" id="select1">

<OPTION selected>volaille</OPTION>

</SELECT>

<INPUT runat="server" type="button" value="Button" id="bouton1">

<INPUT runat="server" type="submit" value="Submit" id="submit1">

</form>

pourrait être traité dans le code-behind par les déclarations suivantes:

public class WebForm1 : System.Web.UI.Page

{

protected System.Web.UI.HtmlControls.HtmlInputCheckBox check1;

protected System.Web.UI.HtmlControls.HtmlInputRadioButton radio1;

protected System.Web.UI.HtmlControls.HtmlSelect select1;

protected System.Web.UI.HtmlControls.HtmlInputButton bouton1;

protected System.Web.UI.HtmlControls.HtmlInputButton submit1;

et les instructions suivantes :

check1.Checked=true;

radio1.Checked=true;

select1.Items.Add("poisson");

select1.Items.Add("viande");

bouton1.Value = "vu au chargement"; ………….

**Remarques:** Il se peut que le contrôle mis dans le formulaire .aspx ne soit pas déclaré dans la page de code-behind (.cs). dans ce cas, vérifier qu'il a bien l'attribut runat="server" et forcer la "**synchronisation de la structure du document** " dans le menu contextuel (bouton droit souris) du formulaire.

Ou simplement faire "**exécuter en tant que contrôle serveur**" du même menu contextuel sur le contrôle.

* **System.Web.UI.WebControls**

L'espace de noms **System.Web.UI.WebControls** est une collection de classes qui permet de créer des contrôles serveur Web sur une page Web. Les contrôles serveur Web sont exécutés sur le serveur et incluent des contrôles de formulaire tels que des boutons et des zones de texte. Ils incluent également des contrôles ayant un but spécial, par exemple un calendrier. Comme les contrôles serveur Web s'exécutent sur le serveur, il est possible de contrôler ces éléments par programme. Les contrôles serveur Web sont plus abstraits que les contrôles serveur HTML. Leur modèle objet ne reflète pas nécessairement la syntaxe HTML.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpref/html/frlrfSystemWebUIWebControls.htm

1°) Créer dans le projet une deuxième page de formulaire (WebForm2).

Insérer un bouton image et une liste box dans le formulaire de la page grâce à la boîte à outils Web Forms et générer le projet.

Haut du formulaire

Bas du formulaire

Haut du formulaire

Bas du formulaire

Haut du formulaire

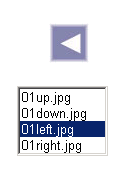
Bas du formulaire

<form id="WebForm2" method="post" runat="server">

<asp:ListBox id="ListBox1" runat="server"></asp:ListBox>

<asp:ImageButton id="ImageButton1" runat="server"></asp:ImageButton>

</form>



La liste contiendra des noms d'images et l'image sélectionnée remplira le bouton image.

2°) Dans le code-behind rajouter les traitements puis regénérer le projet :

protected System.Web.UI.WebControls.ImageButton ImageButton1;

protected System.Web.UI.WebControls.ListBox ListBox1;

private void Page\_Load(object sender, System.EventArgs e)

{

ListBox1.AutoPostBack=true;

if (!IsPostBack) // si c'est le premier chargement chez le client

{

ListBox1.Items.Add("01up.jpg");

ListBox1.SelectedIndex = 0;

ListBox1.Items.Add("01down.jpg");

ListBox1.Items.Add("01left.jpg");

ListBox1.Items.Add("01right.jpg");

ImageButton1.ImageUrl = "./images/01up.jpg";

}

}

private void ListBox1\_SelectedIndexChanged(object sender, System.EventArgs e)

{

ImageButton1.ImageUrl= "./images/" + ListBox1.Items[ListBox1.SelectedIndex].Value;

}

* Constater que l'image du bouton change lorsqu'on change la sélection: cela n'est rendu possible que par la propriété **AutoPostBack** à true qui indique qu'une publication sur le serveur se produit automatiquement lorsque l'utilisateur modifie la sélection de la liste.

Cela se traduit par l'ajout d'un gestionnaire de l'événement onChange sur la balise select dans le navigateur du client.

* Constater également que sans le test de la propriété **IsPostBack** de la page, la liste se trouve augmentée de 4 item à chaque appel de la page.

Le source de la page affichée par le navigateur étant le suivant:

<body MS\_POSITIONING="GridLayout">

<form name="WebForm2" method="post" action="WebForm2.aspx" id="WebForm2">

<input type="hidden" name="\_\_EVENTTARGET" value="" />

<input type="hidden" name="\_\_EVENTARGUMENT" value="" />

<input type="hidden" name="\_\_VIEWSTATE" value="dDwxMTYwMTc2MDg5O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+O2k8Mz47PjtsPHQ8dDxwPHA8bDxBdXRv UG9zdEJhY2s7PjtsPG88dD47Pj47PjtwPGw8aTwwPjtpPDE+O2k8Mj47aTwzPjs+O2w8cDwwMXVwLmpwZzswMXVwLmpwZz47cDwwMWRvd24uanBnOzAxZG93bi5qcGc+O3A8MDFsZWZ0LmpwZzswMWxlZnQuanBnPjtwPDAxcmlnaHQuanBnOzAxcmlnaHQuanBnPjs+Pjs+Ozs+O3Q8cDxwPGw8SW1hZ2VVcmw7PjtsPC4vaW1hZ2VzLzAxdXAuanBnOz4+Oz47Oz47Pj47Pj47bDxJbWFnZUJ1dHRvbjE7Pj4zIocAwsSwHsyL5d6UgBjRR7Ot1w==" />

<script language="javascript">

<!--

function **\_\_doPostBack**(eventTarget, eventArgument) {

var theform = document.WebForm2;

theform.\_\_EVENTTARGET.value = eventTarget;

theform.\_\_EVENTARGUMENT.value = eventArgument;

theform.submit();

}

// -->

</script>

<select name="ListBox1" id="ListBox1" size="4" **onchange="\_\_doPostBack('ListBox1','')"**

language="javascript" style="Z-INDEX: 101; LEFT: 53px; POSITION: absolute; TOP: 120px">

<option selected="selected" value="01up.jpg">01up.jpg</option>

<option value="01down.jpg">01down.jpg</option>

<option value="01left.jpg">01left.jpg</option>

<option value="01right.jpg">01right.jpg</option>

</select>

<input type="image" name="ImageButton1" id="ImageButton1"

src="/Demarrage-ASP-NET/WebApplication1/images/01up.jpg" border="0"

style="Z-INDEX: 102; LEFT: 85px; POSITION: absolute; TOP: 57px" />

</form>

</body>

* **une deuxième conclusion**

Les composants insérés dans le formulaire à l'aide de la boîte à outils Web Forms seront traduits sur le navigateur du client par des balises HTML classiques, auxquelles sont associés des gestionnaires d'événements si l'on désire que ceux ci soient traités au niveau du serveur.

Par exemple le gestionnaire \_\_doPostBack de l'événement onchange de la balise select, qui se contente de signaler au serveur par un champ caché que c'est cette balise qui lui a déclenché la soumission du formulaire.

#### LES COMPOSANTS DE VALIDATION

* **Principes**

La validation de saisie d'un formulaire peut se produire au niveau du client (en java script) ou du serveur (en asp). Le premier cas nuit à la portabilité de la page et le deuxième cas augmente le trafic réseau par des allers retours entre le client et le serveur.

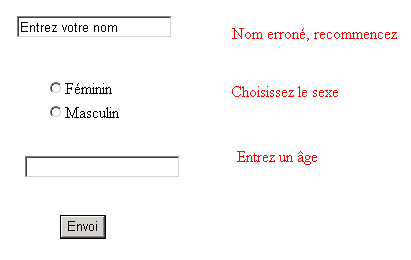
ASP.NET propose des composants qui optimisent ces validations; ils font partie de l'espace de noms System.Web.UI.WebControls.

Par exemple:

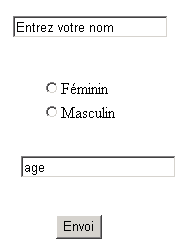
* **System.Web.UI.WebControls.CompareValidator** compare la valeur entrée par l'utilisateur dans un contrôle d'entrée avec la valeur entrée dans un autre contrôle d'entrée ou une constante.
* **System.Web.UI.WebControls.RequiredFieldValidator** convertit le contrôle d'entrée associé en champ requis.
* **System.Web.UI.WebControls.RangeValidator** vérifie si la valeur d'un contrôle d'entrée se trouve dans une plage de valeurs spécifiée.
* **System.Web.UI.WebControls.RegularExpressionValidator** valide si la valeur d'un contrôle d'entrée associé correspond au modèle spécifié par une expression régulière

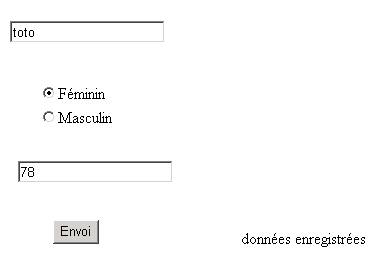
Ce dernier contrôle sera vu dans l'exemple WebForm10.aspx

* **Exercice dirigé**



On se propose de faire un formulaire de saisie du nom (obligatoirement toto), du sexe et de l'âge, qui fait les contrôles suivants :





Et qui permet l'enregistrement des données par le serveur si tous les contrôles sont satisfaits :

1°) Créer une nouvelle page de formulaire dans le projet (WebForm3) pour effectuer la saisie d'un nom précis qui sera validé par un composant "**CompareValidator**". Ce composant n'est pas visible dans le navigateur.

Fichier WebForm3.aspx :

<asp:TextBox id="TextBox1" runat="server"> </asp:TextBox>

<asp:CompareValidator id="CompareValidator1" runat="server"

ErrorMessage="CompareValidator"> </asp:CompareValidator>

et rajouter dans le script décrivant la procédure de chargement Page\_Load :

TextBox1.Text = "Entrez votre nom (toto)";

*Attention*: pour que la procédure Page\_Load(Object sender, EventArgs e) soit active, il faut mettre **AutoEventWireup** à "true" dans la directive de Page du fichier aspx

Fichier WebForm3.aspx.cs :

protected System.Web.UI.WebControls.TextBox TextBox1;

protected System.Web.UI.WebControls.CompareValidator CompareValidator1;

et rajouter dans la procédure de chargement:

CompareValidator1.ValueToCompare = "toto";

CompareValidator1.ControlToValidate = "TextBox1";

CompareValidator1.Operator = ValidationCompareOperator.Equal;

CompareValidator1.ErrorMessage = "Nom erroné, recommencez";

2°) Ajouter 2 boutons radio pour le choix du sexe dans le formulaire et forcer le choix avant l'envoi du formulaire par un bouton de soumission

Utiliser un " **RequiredFieldValidator** ". Ce composant n'est pas visible dans le navigateur.

Les radios boutons sont générés à partir de **System.Web.UI.WebControls.RadioButtonList** sur laquelle on peut ajouter des items (boutons) par sa fenêtre des propriétés.

Fichier WebForm3.aspx :

<asp:RadioButtonList id="RadioButtonList1" runat="server">

<asp:ListItem Value="F">F&#233;minin</asp:ListItem>

<asp:ListItem Value="M">Masculin</asp:ListItem>

</asp:RadioButtonList>

<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"

ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator>

Fichier WebForm3.aspx.cs :

protected System.Web.UI.WebControls.RadioButtonList RadioButtonList1;

protected System.Web.UI.WebControls.RequiredFieldValidator RequiredFieldValidator1;

et à rajouter dans la procédure de chargement:

RequiredFieldValidator1.ControlToValidate = "RadioButtonList1";

RequiredFieldValidator1.ErrorMessage = "Choisissez le sexe";

Des contrôles RequiredFieldValidator devront également être mis par la suite sur les zones de saisie du nom et de l'âge.

3°) Ajouter une zone de texte pour la saisie de l'âge, forcer le saisie et contrôler la valeur avant l'envoi du formulaire.

Utiliser un " **RangeValidator** ". Ce composant n'est pas visible dans le navigateur.

Fichier WebForm3.aspx :

<asp:RequiredFieldValidator id="RequiredFieldValidator2" runat="server"

ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator>

<asp:RangeValidator id="RangeValidator1" runat="server"

ErrorMessage="RangeValidator"></asp:RangeValidator>

<asp:TextBox id="TextBox2" runat="server">age</asp:TextBox>

Remarquer : "age"

Fichier WebForm3.aspx.cs :

protected System.Web.UI.WebControls.TextBox TextBox2;

protected System.Web.UI.WebControls.RequiredFieldValidator RequiredFieldValidator2;

protected System.Web.UI.WebControls.RangeValidator RangeValidator1;

et à rajouter dans la procédure de chargement:

RequiredFieldValidator2.ControlToValidate = "TextBox2";

RequiredFieldValidator2.ErrorMessage = "Entrez un âge";

RangeValidator1.ControlToValidate = "TextBox2";

RangeValidator1.MinimumValue = "1";

RangeValidator1.MaximumValue = "120";

RangeValidator1.Type = ValidationDataType.Integer;

RangeValidator1.ErrorMessage = "Soyez raisonnable quant à l'âge!";

4°) Ajouter un contrôle "label" qui mettra un message lorsque toutes les saisies se seront enfin bien passées et que les données auront été enregistrées (fonction en code-behind), et un bouton d'envoi des données

Fichier WebForm3.aspx :

Remarquer : vide

<asp:Label id="Label1" runat="server"> </asp:Label>

<asp:button id="Button1" runat="server" Text="Envoi"></asp:button>

Fichier WebForm3.aspx.cs :

protected System.Web.UI.WebControls.Button Button1;

protected System.Web.UI.WebControls.Label Label1;

private void InitializeComponent()

{

this.Button1.Click += new System.EventHandler(this.Button1\_Click);

this.Load += new System.EventHandler(this.Page\_Load);

}

private void Button1\_Click(object sender, System.EventArgs e)

{ // enregistre les données dans une base de données par exemple

Label1.Text = "données acceptées et enregistrées dans la base";

// ou autre message au besoin

}

#### UTILISATION D'UN ESPACE DE NOM : System.IO

* **Principes**

On pourrait envisager de stocker le nombre d'enregistrements effectués grâce au formulaire précédent dans un fichier sauvegardé sur le serveur. Ce traitement spécifique sera fait dans le fichier code-behind de la page, suit à l'événement Button1\_Click .

Pour ce faire, il faut utiliser une classe de l'espace de nom ***System.IO***

L'espace de noms System.IO contient des types permettant les opérations de lecture et d'écriture synchrones et asynchrones sur les fichiers et flux de données.

La classe ***File*** fournit des méthodes "static" pour créer, copier, supprimer, déplacer et ouvrir des fichiers et facilite la création d'objets FileStream. La classe ***FileInfo*** fournit des méthodes d'instance identiques.

La classe ***Directory*** fournit des méthodes "static" pour la création, le déplacement et l'énumération dans les répertoires et les sous-répertoires. La classe ***DirectoryInfo*** fournit des méthodes d'instance identiques.

La classe ***Path*** fournit des méthodes "static" pour la manipulation des chaînes qreprésentant des noms de fichiers et de répertoires.

La classe abstraite de base *Stream* prend en charge la lecture et l'écriture d'octets; elle n'est bien sur pas utilisable telle quelle.

La classe ***FileStream*** en dérive et prend en charge l'accès aléatoire aux fichiers, avec plusieurs modes d'ouvertures et des accès séquentiels et directs.

Ce sera celle que l'on utilisera pour stocker la valeur binaire du compteur d'événements:

* **Mise en oeuvre**

Les noms du répertoire et du fichier seront déclarés dans la classe du formulaire:

private string nomFichier = "Compteur.data";

private string nomRep = "RepData";

et une fonction permettra d'incrémenter le contenu du fichier:

private int Comptabilise ()

{

// donne le chemin physique de la page

string pathAppli = Page.MapPath(".");

nomFichier = pathAppli + "\\" + nomRep + "\\" + nomFichier;

FileStream fic = new FileStream(nomFichier, FileMode.Open); // ouverture du fichier

int cpt=fic.ReadByte(); // lecture de l'octet

cpt++;

fic.Seek(0, SeekOrigin.Begin);

fic.WriteByte((byte)cpt); // écriture après casting de l'entier en octet

fic.Close();

return cpt;

}

##### C#

Casting implicite entre types compatibles

Ex: byte -> int

Casting explicite entre types compatibles

Ex: int -> byte par : varB=(byte)varI; // tronque les bits forts

Il faudra préalablement créer le répertoire et le fichier - s'ils n'existent pas – lors du premier démarrage de la page, en rajoutant dans l'événement de chargement la fonction suivante :

private void Init\_Fichier()

{

// donne le chemin physique de la page

pathAppli = **Page.MapPath**(".");

// création du répertoire s'il n'existe pas

nomRep = pathAppli + "//" + nomRep;

if (!**Directory**.**Exists** (nomRep))

{

Directory.CreateDirectory(nomRep);

}

//création du fichier s'il n'existe pas

nomFichier = nomRep + "//" + nomFichier;

if (!**File**.**Exists**( nomFichier))

{

**FileStream fic = new FileStream(nomFichier, FileMode.CreateNew);**

fic.WriteByte(0);

fic.Close();

}

}

ATTENTION:

Le message d'erreur suivant apparaît si la sécurité du répertoire de la page n'est pas correcte:

"Le processus ASP.NET n'est pas autorisé à accéder à la ressource demandée. Pour des raisons de sécurité, l'identité du processus ASP.NET par défaut est '{machinename}\ASPNET', qui dispose de privilèges limités."

Dans ce cas rajouter l'utilisateur ASPNET de la machine serveur en accès "modifier" pour pouvoir créer le répertoire (compte aspnet\_wp)

#### Les composants de contrôle de données

Les composants de contrôle Web Forms décrits précédemment offrent une interface à l'utilisateur et peuvent pour la plupart associer leurs propriétés à des données : c'est le mécanisme du **Data Binding**.

* **Principes du Data Binding**

C'est l'association entre une propriété de page ou de composant de contrôle et une source de données qui peut être une propriété d'un composant, le retour d'exécution d'une méthode, une collection, une expression

Toutes les associations de la page se font lors de l'appel à la méthode DataBind() de la page; cette méthode fonctionne en cascade sur les enfants du conteneur auquel elle s'applique.

Par exemple au chargement de la page, le code C# peut assurer la liaison de la sorte:

private void Page\_Load(object sender, System.EventArgs e)

{

**Page.DataBind();**

}

La liaison est repérée par la syntaxe **<%# *expression* %>** utilisée en substitution d'un texte dans le contrôle à lier (pour sa value, ses item ….)

* **Un premier exemple : Data Binding sur une propriété d'un composant de contrôle**

Créer un nouveau formulaire WebForm4.aspx contenant un Label et une DropDownList

Notes: Le contrôle serveur Web **DropDownList** permet aux utilisateurs de sélectionner un ou plusieurs éléments dans une liste prédéfinie. Il diffère du contrôle serveur Web **ListBox** par le fait que la liste des éléments reste masquée jusqu'à ce que l'utilisateur clique sur le bouton de déroulement. De plus, le contrôle DropDownList ne prend pas en charge la sélection multiple, contrairement au contrôle ListBox.

Le label a sa propriété "Text" qui est le reflet de la propriété "Value" de l'item sélectionnée dans la liste déroulante. L'association doit être mise à jour à chaque changement de sélection dans la liste déroulante (il faut donc traiter son événement qui doit être explicitement ramené au serveur)

private void Page\_Load(object sender, System.EventArgs e)

{

if (! Page.IsPostBack) // si c'est le premier chargement chez le client

{

Label1.Text="pas lié"; // texte libre au chargement

DropDownList1.AutoPostBack=true; // pour récupérer les evts au serveur

}

}

private void DropDownList1\_SelectedIndexChanged(object sender, System.EventArgs e)

{

Page.DataBind(); // réalise toutes les associations dans la page

}

<form id="WebForm4" method="post" runat="server">

Quel est votre département?

<asp:DropDownListid="DropDownList1" runat="server">

<asp:ListItem Value="inconnu">?</asp:ListItem>

<asp:ListItem Value="Paris">75</asp:ListItem>

<asp:ListItem Value="les Hauts de Seine">92</asp:ListItem>

<asp:ListItem Value="la Seine St Denis">93</asp:ListItem>

<asp:ListItem Value="le Val de Marne">94</asp:ListItem>

<asp:ListItem Value="le Val d'Oise">95</asp:ListItem>

</asp:DropDownList>

Votre département est

<asp:Label **text="<%# DropDownList1.SelectedItem.Value %>"**  runat="server">

</asp:Label>

</form>

* **Un deuxième exemple : Data Binding sur une fonction**

On reprend l'exemple précédent auquel on rajoute la définition d'une fonction de mise à l'heure:

protected string MiseALHeure()

La fonction doit être de visibilité "protected" pour être utilisable par la balise asp:Label dans le fichier .aspx

{

DateTime dt=DateTime.Now;

String sd = dt.ToLongDateString();

String st = dt.ToShortTimeString();

return sd.ToString() +" "+st.ToString();

}

et un label qui sera lié au résultat d'exécution de cette fonction.

<asp:Label **text="<%# MiseALHeure() %>"** id="Label2" runat="server"></asp:Label>

On remarque que la liaison (donc l'heure) est mise à jour par l'effet cascade de la méthode Page.DataBind(); à chaque événement de changement de la liste déroulante.

* **Un troisième exemple : l'utilisation des composants de contrôle DataList et DataGrid**

L'espace de noms System.Web.UI.WebControls propose des classes de composants de contrôle adaptées pour l'affichage des données (DataGrid, DataList, ButtonColumn etc).

DataList sert à afficher une liste associée à des données, en utilisant des modèles.

DataGrid sert à afficher des données liées sous forme de tableau et permet de les sélectionner, trier et modifier.

1°) Le composant de contrôle **DataList**

Mettre dans le formulaire un contrôle DataList.

Avec le bouton droit sur le composant, prendre le menu "mise en forme automatique" pour choisir une présentation et prendre le menu "Générateur de propriétés" pour choisir la disposition de la liste et les options d'en tête et pied de liste.

Le DataList est constitué d'une en tête, suivie des items, avec un item particulier lorsqu'il est sélectionné et terminée par un pied. Chacune de ces 4 entités est personnalisée par un style au moyen des balises suivantes:

<**asp:DataList** id="DataList1" runat="server" BorderColor="White" BorderStyle="Ridge"

CellSpacing="1" BackColor="White" CellPadding="3" BorderWidth="2px">

<SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#9471DE">

</SelectedItemStyle>

<ItemStyle ForeColor="Black" BackColor="#DEDFDE"></ItemStyle>

<FooterStyle ForeColor="Black" BackColor="#C6C3C6"></FooterStyle>

<HeaderStyle Font-Bold="True" ForeColor="#E7E7FF" BackColor="#4A3C8C">

</HeaderStyle>

<**/asp:DataList**>

La source des valeurs utilisée pour remplir les items dans le contrôle DataList se définit par sa propriété **.DataSource.** Dans notre exemple, la source des données sera la collection des items de la DropDownList et elle sera renseignée au chargement de la page. La liaison sera exécutée par la méthode **.DataBind()**

private void Page\_Load(object sender, System.EventArgs e)

{

DataList1.DataSource = DropDownList1.Items; // définit la source de données DataList1.DataBind(); // réalise l'association à la source de données

}

Le contenu du contrôle DataList est défini à l'aide de modèles, tels que IdemTemplate qui donne le contenu et la disposition des items de la liste, HeaderTemplate pour l'en tête de la liste, SelectedItemtemplate pour l'élément sélectionné (par défaut il est équivallent à ItemTemplate) et FooterTemplate pour le pied de liste.

Seul le modèle **ItemTemplate** est obligatoire et défini par une balise **<ItemTemplate>.**

<ItemTemplate>

<%# Container.DataItem%> **:**

<%#((ListItem)Container.DataItem).Value%>

</ItemTemplate>

Il faut spécifier les enregistrements à lier, grâce à deux objets que l'on peut utiliser dans une expression de liaison de données (<%# …. %>):

- L'objet **Container** qui fait référence au contrôle parent (la DataList ici) qui, à son tour, est lié à la source de données.

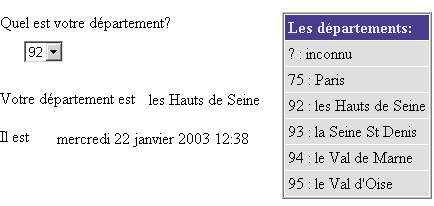
- L'objet **DataItem** qui fait référence à l'enregistrement en cours de traitement par le contrôle parent.

Ainsi: **<%# Container.DataItem%>** signifie que le modèle est basé sur le champ par défaut du conteneur de liaison de données , donc la propriété Text de la DropDownList

et **<%#((ListItem)Container.DataItem).Value%>** signifie que l'on rajoute au modèle un champ explicite qui sera la propriété Value de la DropDownList; remarquer que la DataItem n'est pas typé et qu'il faut le caster pour notre cas en un objet de la classe ListItem qui est la classe des options de la DropDownList afin de lui appliquer ensuite la propriété Value!

et **<%# DataBinder.Eval(Container.DataItem, "Value")%>** donne le même résultat mais en utilisant la méthode .Eval de la classe DataBinder qui reçoit en deuxième paramètre le nom de la propriété qui nous intéresse.

On remarque que le nombre d'items générés dans la DataList correspond au nombre d'enregistrements de sa source de données.



2°) Le composant de contrôle **DataGrid**

On utilise le DataGrid pour afficher les champs d'une source de données sous forme de colonnes dans un tableau. Chaque ligne représente un enregistrement de la source de données. Le contrôle DataGrid prend en charge la sélection, la modification, la suppression, la pagination et le tri.

Mettre dans le formulaire un contrôle DataGrid

Avec le bouton droit sur le composant, prendre le menu "mise en forme automatique" pour choisir une présentation et prendre le menu "Générateur de propriétés" pour choisir la disposition de la grille et les options d'en tête et pied de grille.

On obtient à nouveau des balises qui définissent le style des différentes entités de la grille:

<**asp:DataGrid** id="DataGrid1" runat="server" BorderWidth="20px" CellPadding="4"

BackColor="White" BorderStyle="None" BorderColor="#CC0000"

ForeColor="Black" GridLines="Vertical" AutoGenerateColumns="True">

<SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#CC3333">

</SelectedItemStyle>

<HeaderStyle Font-Size="Larger" Font-Names="Comic Sans MS" Font-Italic="True"

Font-Bold="True" ForeColor="White" BackColor="#333333">

</HeaderStyle>

<FooterStyle ForeColor="Black" BackColor="#CCCC99">

</FooterStyle>

<PagerStyle HorizontalAlign="Right" ForeColor="Black" BackColor="White">

</PagerStyle>

<**/asp:DataGrid**>

La source des valeurs utilisée pour remplir les colonnes dans le contrôle DataGrid se définit par sa propriété **.DataSource.** Dans notre exemple, la source des données sera la collection des items de la DropDownList et elle sera renseignée au chargement de la page. La liaison sera exécutée par la méthode **.DataBind()**

private void Page\_Load(object sender, System.EventArgs e)

{

DataGrid1.DataSource = DropDownList1.Items; // définit la source de données

DataGrid1.DataBind(); // réalise l'association à la source de données

}

Puisque la source de données est renseignée, la DataGrid propose alors par défaut les champs disponibles dans la source (attribut **AutoGenerateColumns="False"** de la balise):



Remarquer

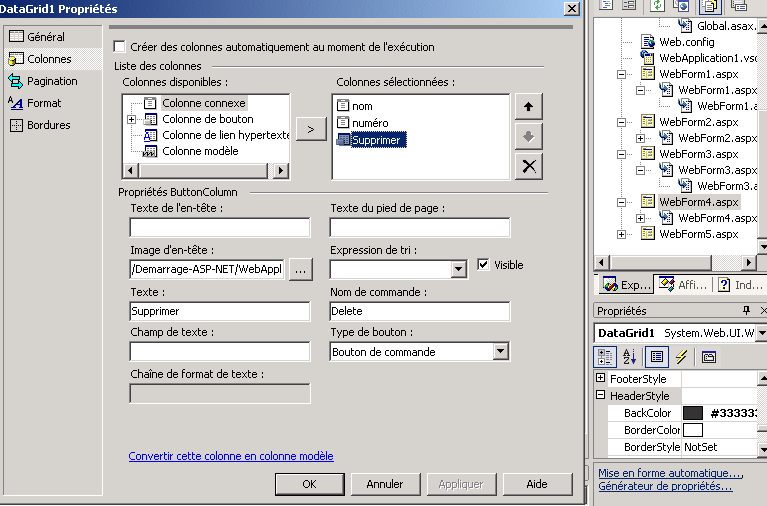
Cette option peut être retirée en décochant la case "Créer des colonnes automatiquement au moment

de l'exécution" de l'onglet "Colonnes" de la page

des propriétés ouverte depuis la fenêtre des propriétés du contrôle DataGrid ou en passant l'attribut AutoGenerateColumns à "False"

Dans ce cas on peut choisir les champs que l'on souhaite dans la source de données. Des colonnes peuvent être définies par la page des propriétés de la fenêtre des propriétés de la DataGrid.

Par exemple:



Il faut alors déterminer les colonnes que l'on souhaite ajouter à la collection *Columns* du DataGrid et qui peuvent être de différents types:

- des colonnes liées à un champ de la source de données, appelées colonnes connexes (**BoundColumn**)

- des colonnes qui affichent des boutons d'ajout, de retrait …(**ButtonColumn**)

- des colonnes qui donnent des liens (**HyperLinkColumn**)

etc …

Ce qui rajoutera dans la balise <asp:DataGrid**>** le source aspx suivant:

**<Columns>**

<asp:BoundColumn DataField="Text" HeaderText="nom"></asp:BoundColumn>

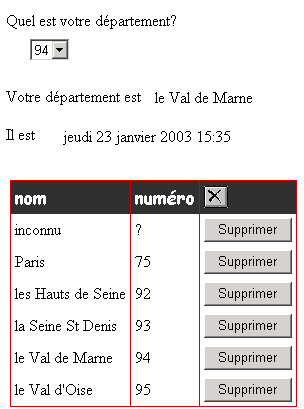
<asp:BoundColumn DataField="Value" HeaderText="num&#233;ro"></asp:BoundColumn>

<asp:ButtonColumn Text="Supprimer" ButtonType="PushButton" CommandName="Delete"

HeaderImageUrl="/Demarrage-ASP-NET/WebApplication1/images/DELETE.BMP"

</asp:ButtonColumn>

**</Columns>**



et présentera la page de la sorte:

On constate que sous le navigateur les

boutons des suppressions sont autant

de bouton de soumission (ou publica-

tion) du formulaire.

Il suffira alors de traiter les événements

côté serveur pour par exemple détruire

des enregistrements d'une base de

données.

#### Association des composants de contrôle avec

#### des bases de données

Tous les composants de contrôle que nous venons d'utiliser peuvent être liés aux données issues de bases de données.

* **Principes**

Trois étapes sont nécessaires:

1. Connexion à la base de données
2. Sélection et extraction des données
3. Association du composant à la donnée appropriée

et les deux espace **System.Data** et **System.Data.OleDb** doivent être importés:

using System.Data;

using System.Data.OleDb;

1. La connexion à la base de données

Elle est réalisée par un objet de la classe **System.Data.OleDb.OleDbConnection** renseigné par une chaîne de connexion ou par un DSN (Nom de Source de Données) préparé sur la machine serveur (par l'outil d'administration nommé "sources de données ODBC").

Une fois créée, la connexion doit être ouverte - méthode **.Open()** – et fermée à la fin de son utilisation – méthode **.Close()**

OleDbConnection connexDB;

connexDB = new OleDbConnection("Provider=SQLOLEDB; server=APISERVEURBLEU;

Initial Catalog = PUBS; User Id=stage; Password=stage;");

try { connexDB.Open(); }

catch(Exception ex) { …. }

1. La sélection et l'extraction des données

Ceci est réalisé par 3 objets: un objet de la classe **System.Data.OleDb.OleDbCommand** pour donner la chaîne requête SQL , un objet de la classe **System.Data.OleDb.OleDbDataAdapter** et un objet de la classe **System.Data.DataSet**.

L'objet OleDbDataAdapter joue le rôle de pont entre le cache en mémoire des données dans le DataSet, et la source de données; il fournit ce pont en utilisant sa méthode **.Fill** pour charger les données à partir de la source de données dans le DataSet.

OleDbCommand commDB;

OleDbDataAdapter adaptDB;

DataSet ds;

commDB = new OleDbCommand ("select \* from authors", connexDB);

adaptDB = new OleDbDataAdapter(commDB);

ds = new DataSet();

try { adaptDB.Fill(ds); }

catch(Exception ex) { …. }

1. L' association des composants aux données

Elle se fait en définissant la propriété .DataSource des contrôles et si besoin les propriétés des contrôles qui sélectionnent les champs de la source de données:

ListBox1.DataSource = ds; pour une ListBox, ou un DropDownList

ListBox1.DataTextField="au\_lname";

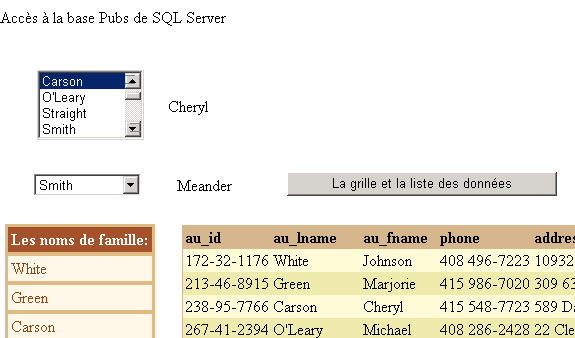
ListBox1.DataValueField="au\_fname";

DataGrid1.DataSource = ds; pour une DataGrid

DataList1.DataSource = ds; pour une ListGrid

Puis en réalisant la liaison par la méthode **.Bind()** de chaque contrôle ou globalement de tous les contrôles liés la page (Page.DataBind();)

* **Exemples:**



Dans cet exemple WebForm5, les deux listes sont liées à la source de données (Base SQL Server) au chargement de la page.

Les deux labels sont liés aux deux listes et mis à jours à chaque changement de sélection dans la liste.

La ListGrid de gauche est liée à la même source de donnée, mais à la demande du bouton. Son modèle de données est basé sur un des champs de la source de données.

La DataGrid de droite est également liée à la source de donnée à la demande du bouton. Elle présente par défaut tous les champs de la source de données.

**Source de l'exemple –> fichier WebForm5.aspx :**

<body MS\_POSITIONING="GridLayout">

Accès à la base Pubs de SQL Server

<form id="WebForm5" method="post" runat="server">

**<asp:listbox** id="ListBox1" runat="server" > </asp:listbox>

**<asp:dropdownlist** id="DropDownList1" runat="server" > </asp:dropdownlist>

**<asp:label** id= runat="server" Text="<%# ListBox1.SelectedItem.Value %>">label 1</asp:label>

**<asp:label** id= runat="server" Text="<%# DropDownList1.SelectedItem.Value %>">Label 2 </asp:label>

**<asp:button** id="Button1" runat="server" Text="La grille et la liste des données"> </asp:button>

**<asp:DataGrid** id="DataGrid1 runat="server" BorderColor="Tan" BorderWidth="1px"

BackColor="LightGoldenrodYellow" CellPadding="2" GridLines="None" ForeColor="Black">

<SelectedItemStyle ForeColor="GhostWhite" BackColor="DarkSlateBlue"></SelectedItemStyle>

<AlternatingItemStyle BackColor="PaleGoldenrod"></AlternatingItemStyle>

<HeaderStyle Font-Bold="True" BackColor="Tan"></HeaderStyle>

<FooterStyle BackColor="Tan"></FooterStyle>

<PagerStyle HorizontalAlign="Center" ForeColor="DarkSlateBlue" BackColor="PaleGoldenrod">

</PagerStyle>

</asp:DataGrid>

**<asp:Label** id="Labelerrrunat="server"></asp:Label>

**<asp:DataList** id="DataList1" runat="server" BorderColor="#DEBA84" BorderWidth="1px"

BackColor="#DEBA84" CellPadding="3" GridLines="Both" BorderStyle="None" CellSpacing="2">

<SelectedItemStyle Font-Bold="True" ForeColor="White" BackColor="#738A9C">

</SelectedItemStyle>

<ItemStyle ForeColor="#8C4510" BackColor="#FFF7E7"></ItemStyle>

<HeaderTemplate>

Les noms de famille:

</HeaderTemplate>

<ItemTemplate>

**<%# DataBinder.Eval(Container.DataItem, "au\_lname") %>**

</ItemTemplate>

<FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5"></FooterStyle>

<HeaderStyle Font-Bold="True" ForeColor="White" BackColor="#A55129"></HeaderStyle>

</asp:DataList>

</form>

</body>

**Source de l'exemple –> fichier WebForm5.cs :**

using System;

…..

Importer ces 2 espaces de noms

using System.Web.UI.HtmlControls;

using System.Data;

using System.Data.OleDb;

namespace WebApplication1

{

public class WebForm5 : System.Web.UI.Page

{

protected System.Web.UI.WebControls.ListBox ListBox1;

protected System.Web.UI.WebControls.DropDownList DropDownList1;

protected System.Web.UI.WebControls.Label Label2;

protected System.Web.UI.WebControls.Button Button1;

protected System.Web.UI.WebControls.DataGrid DataGrid1;

protected System.Web.UI.WebControls.Label Labelerr;

protected System.Web.UI.WebControls.DataList DataList1;

protected System.Web.UI.WebControls.Label Label1;

private DataSet CreerDataSet()

{

OleDbConnection connexDB;

OleDbCommand commDB;

OleDbDataAdapter adaptDB;

DataSet leds;

**connexDB = new OleDbConnection**

**("Provider=SQLOLEDB;server=APISERVEUR;**

**Initial Catalog=PUBS;User Id=stage;Password=stage;");**

**try {connexDB.Open();}**

**catch(Exception ex) {Labelerr.Text=ex.Message;return null;}**

**commDB = new OleDbCommand("select \* from authors", connexDB);**

**adaptDB = new OleDbDataAdapter(commDB);**

**leds = new DataSet();**

**try {adaptDB.Fill(leds);}**

**catch(Exception ex) {Labelerr.Text=ex.Message;return null;}**

**connexDB.Close();**

return leds;

}

private void Page\_Load(object sender, System.EventArgs e)

{

if (! Page.IsPostBack) // si c'est le premier chargement chez le client

{

DataSet ds = CreerDataSet();

ListBox1.SelectedIndex = 0;

ListBox1.DataSource = ds;

ListBox1.DataTextField="au\_lname";

ListBox1.DataValueField="au\_fname";

ListBox1.DataBind();

ListBox1.AutoPostBack=true; // pour récupérer les evts au serveur

DropDownList1.DataSource = ds;

DropDownList1.DataTextField="au\_lname";

DropDownList1. DataValueField="au\_fname";

DropDownList1.DataBind();

DropDownList1.AutoPostBack=true; // pour récupérer les evts au serveur

}

}

**………..**

private void ListBox1\_SelectedIndexChanged(object sender, System.EventArgs e)

{

Label1.DataBind();

}

private void DropDownList1\_SelectedIndexChanged(object sender, System.EventArgs e)

{

Label2.DataBind();

}

private void Button1\_Click(object sender, System.EventArgs e)

{

DataSet ds = CreerDataSet();

DataGrid1.DataSource = ds;

DataGrid1.DataBind();

DataList1.DataSource = ds;

DataList1.DataBind();

}

}

}

#### L'outil de "Connexions de données" de Visual Studio

Cet outil permet de créer des connexions sur des bases de données et ainsi d'explorer les objets de la base depuis Visual Studio – palie à l'absence d'outil client spécifique de la base de données.

Cet outil est proposé dans la fenêtre "**Explorateur de serveur**", à côté d' autres outils de gestion du serveur (tels que l'observateur d'événements, le gestionnaire des services, etc …). Cette fenêtre doit éventuellement être obtenue à partir du menu d'affichage.

Par le menu du bouton droit sur "**Connexions de données**" on peut demander à ajouter une nouvelle connexion, et alors renseigner sur le fournisseur (Ole DB pour SQL Server par exemple), et donner les noms du serveur SQL (APISERVEUR) et de la base (PUBLI), ainsi qu'un utilisateur (STAGE) et son mot de passe.

1°) Lorsque la connexion est établie, on visualise alors pour la base demandée les Tables, Vues, Procédures Stockées et Fonctions.

En fonction des autorisations accordées à l'utilisateur renseigné lors de la connexion, il est possible de créer des nouveux objets (des procédures par exemple), ou de les modifier (rajouter des enregistrements dans une table par exemple).

2°) D'autre part la connexion est un objet nommé (dans notre exemple APISERVEUR.publi.stage) et possède donc des propriétés que l'on voit dans la "fenêtre des propriétés"; entre autre la propriété **.*ConnectString*** qui donne la chaîne que l'on peut passer en paramètre de connexion dans nos applications aspx d'accès aux données.

Exemple:

*"Provider=SQLOLEDB.1;Persist Security Info=False;User ID=stage;Initial Catalog=publi;Data Source=APISERVEUR;Use Procedure for Prepare=1;Auto Translate=True;Packet Size=4096; Workstation ID=APIBLJ;Use Encryption for Data=False;Tag with column collation when possible=False"*

**ATTENTION**: lorsqu'une connexion est ainsi établie, la base peut être verrouillée en accès exclusif (c'est le cas par défaut pour les connexions sur des bases accès); on voit alors la création d'un fichier .ldb

#### PRESENTATION D' ADO.NET

Le .NET Framework propose une nouvelle technologie d'accès aux données nommée ADO.NET.

Afin d'utiliser ADO.NET, il faut importer les espaces de noms **System.Data** et **System.Data.OleDb**.

Si la source de données est une base de données SQL Server, vous pouvez importer l'espace de noms **System.Data.SqlClient** au lieu de System.Data.OleDb.

Les composants de ADO.NET ont été conçus de façon à distinguer *l'accès aux données -* par le fournisseur de données .NET, ensemble de composants comprenant les objets **Connection**, **Command**, **DataReader** et **DataAdapter** - de *la manipulation de données* - par l'objet **DataSet**

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconadonetarchitecture.htm

* **Principes**

Trois étapes sont nécessaires pour accéder aux données d'une source de données:

1. La connexion à la source de données
2. L'exécution d'une commande
3. La récupération du résultat de cette commande
4. ***La connexion à la base de données***

On a vu précédemment qu'elle était réalisée par un objet de la classe **System.Data.OleDb.OleDbConnection** – ou **System.Data.SqlClient.SqlConnection -** renseigné par une chaîne de connexion ou par un DSN (Nom de Source de Données) préparé sur la machine serveur. Cet objet va être ouvert puis fermé.

1. ***L'exécution de la commande***

Elle est réalisée par un objet de la classe **System.Data.OleDb.OleDbCommand** – ou **System.Data.SqlClient.SqlCommand –** relié à la connexion, soit lors de son instanciation, soit en affectant sa propriété **.Connection**

Cette commande peut être de trois types différents:

sur une procédure stockée -> " *StoredProcedure* "

sur une table entière -> " *TableDirect* " qui renseigne la propriété **.CommandType**

sur une requête (par défaut) -> " *Text* "

et son énoncé – nom de procédure, de table ou chaîne de requête - est alors renseigné par la propriété **.CommandText** - ou encore lors de son instanciation lorsqu'il s'agit d'une requête.

Plusieurs possibilités se présentent ensuite pour exécuter la commande:

a -soit par sa méthode **.ExecuteNonQuery()** pour des instructions UPDATE, INSERT ou DELETE, les valeurs de retour représentant alors le nombre de lignes affectées par la commande

b - soit par sa méthode **.ExecuteScalar()** qui exécute la requête et rend une valeur unitaire : la première colonne de la première ligne du jeu de résultats retourné par la requête.

c - soit par sa méthode **.ExecuteReader()** qui exécute la requête et rend un objet de la classe **System.Data.OleDb.OleDbDataReader** – ou **System.Data.SqlClient.SqlDataReader** – qui permettra de lire les données sous forme d'un flux séquentiel.

d – soit enfin en la passant en paramètre lors de l'instanciation d'un objet de la classe **System.Data.OleDb.OleDbDataAdapter** – ou **System.Data.SqlClient.SqlDataAdapter** – qui permettra de remplir un espace mémoire (le DataSet).

1. ***La récupération du résultat de cette commande***

Le jeu d'enregistrements récupéré suite à l'exécution d'un "select" d'une requête ou procédure stockée s'exploite de 2 façons:

* + Soit en **mode connecté**, ligne par ligne, grâce à un objet **DataReader** permettant un accès en lecture seule et en mode séquentiel, comme sur un flux classique. Ce mode a l'avantage de décharger la mémoire puisqu'il ne stocke qu'une ligne à la fois, mais il oblige à garder la connexion ouverte tout le long de son exploitation.
  + Soit en **mode déconnecté,** grâce à un objet **DataSet** qui est un cache en mémoire des données extraites de la source de données, et qui facilite le travail avec des tables provenant de données de sources disparates.. Ce mode de connexion est rapide et permet de fermer la connexion, donc de libérer la base.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/ cpconcreatingdatatables.htm

* **Exemple d'accès en mode déconnecté : le DataSet**

Un **DataSet** (ou **groupe de données**) est un cache en mémoire des données extraites d'une source de données.

Les données des tables sont locales à l'application, et peuvent être obtenues à partir d'une source de données telle que Microsoft® SQL Server, à l'aide d'un **DataAdapter** (pont entre le DataSet et la source de données) qui le remplit. Conseil :  Essayez toujours de limiter au maximum la taille d'un groupe de données en le remplissant uniquement avec les enregistrements dont vous avez besoin.

Un DataSet est constitué d'une *collection de tables de données relationnelles en mémoire* (objets **DataTable**), de relations et de contraintes.



Le schéma, ou structure, d'une table est représenté par des colonnes, qui peuvent mapper aux colonnes d'une source de données, et par des contraintes; il est défini à l'aide d'objets **DataColumn**, ainsi que d'objets **ForeignKeyConstraint** et **UniqueConstraint**.

Outre un schéma, une table doit également avoir des lignes pour contenir et trier les données.

L'objet **DataRow** représente les données en cours contenues dans une ligne de la table.

En conclusion, un DataSet est composé d'une collection de tables (sa propriété **.Tables**) , chaque table étant elle même composée d'une collection de colonnes pour son schéma (propriété **.Columns**) et de lignes pour son contenu (propriété **.Rows**).

Exemple WebForm6 :

Remplissage du DataSet:

private DataSet ds;

private DataSet CreerDataSet()

{

OleDbConnection connexDB;

OleDbCommand commDB;

OleDbDataAdapter adaptDB;

DataSet leds;

connexDB = new OleDbConnection ("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBS;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = "select \* from authors";

**adaptDB = new OleDbDataAdapter(commDB);**

**leds = new DataSet();**

try {**adaptDB.Fill(leds, "les auteurs");}** catch(Exception ex) {throw(ex);} // remplir une table

commDB.CommandText = "select \* from titles";

try {**adaptDB.Fill(leds, "les titres");}** catch(Exception ex) {throw(ex);} // remplir une deuxième table

connexDB.Close();

return leds;

}

Dans cette fonction on constate:

* Que l'objet Command peut être utilisé plusieurs fois sur des énoncés différents
* Que l'objet DataAdapter, une fois lié à l'objet Command, permet de remplir plusieurs fois le DataSet, en nommant chaque fois le jeu d'enregistrements – ou table – du DataSet

Exploitation du DataSet:

protected String LireTable (String laquelle)

{

DataTable table = ds.Tables[laquelle];

int nbCols = **table.Columns.Count**;

String chaine="";

chaine += "Schéma de la table \"" + laquelle + "\": <BLOCKQUOTE>";

for (int i=0; i<nbCols; i++)

{

chaine += **table.Columns[i].**Ordinal + " -> \"";

chaine += table.Columns[i].ColumnName +"\" de type ";

chaine += table.Columns[i].DataType + "<BR>";

}

chaine += "</BLOCKQUOTE> Contenu de la table \"" + laquelle + "\":<BR><BR>";

for (int i=0; i<**table.Rows.Count**; i++)

{

for (int j=0; j<nbCols; j++)

chaine += **table.Rows[i][j]** + " | ";

chaine += "<BR>";

}

chaine += "<HR>";

return chaine;

}

Dans cette fonction on constate:

* Que l'objet Table possède une collection des colonnes qui renseigne sur son schéma
* Que l'objet Table possède une collection de lignes qui contient ses données

Et dans le source WebForm6.aspx :

<%Response.Write(LireTable("les auteurs"));%>

<%Response.Write(LireTable("les titres")); %>

* **Exemple d'accès en mode connecté : le DataReader**

Si vous souhaitez simplement afficher des données sur la page, la création d'un DataSet, son remplissage, puis la liaison des contrôles peut représenter une charge mémoire inutile, puisque le DataSet sera immédiatement supprimé. Il est souvent plus efficace d'utiliser un **lecteur de données** (**DataReader**) pour extraire les données et y lier des contrôles au moment de l'exécution.

Un DataReader est un flux en lecture seule sur une source de données. Sa méthode **.Read()** permet de l'avancer ligne par ligne en rendant *true* tant qu'il reste un enregistrement;

L'enregistrement courant du DataReader contient les colonnes de données, accessibles soit par indexation directe, soit par des méthodes spécifiques telles que .GetString() ou .GetInt32() qui reçoivent alors l'index de la colonne voulue.

Ensuite le DataReader doit être fermé.

Exemple WebForm7 :

Ouverture du DataReader:

private OleDbDataReader rd;

private OleDbDataReader CreerDataReader()

{

OleDbConnection connexDB;

OleDbCommand commDB;

OleDbDataReader readerDB;

connexDB = new OleDbConnection("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBS;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = "select title, price, royalty from titles";

try {**readerDB = commDB.ExecuteReader(CommandBehavior.CloseConnection);**}

catch(Exception ex) {throw(ex);}

return readerDB;

}

Exploitation du DataReader:

protected String LireReader()

{

int nbLignes = 0;

String chaine = "Liste des titres et de leurs prix: <BLOCKQUOTE>";

**while (rd.Read())**

{

nbLignes ++;

chaine += **rd.GetString(0)** + " -> ";

chaine += **rd[1]** + " $";

chaine += " \*\*\*\* royalty: " + rd[2];

chaine += "<BR>";

}

**rd.Close();**

chaine += "</BLOCKQUOTE> Soit un total de " + nbLignes + " titres";

return chaine;

}

Et dans le source WebForm7.aspx :

<%Response.Write(LireReader());%>

* **Les DataView**

Un **DataView** permet de créer différentes vues des données stockées dans un DataTable. Il propose une vue dynamique des données dont le contenu, l'ordre de tri et l'appartenance reflètent les changements apportés au DataTable sous-jacent à mesure qu'ils sont effectués.

On peut utiliser un **DataViewManager** pour gérer les paramètres de vue pour toutes les tables d'un DataSet.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconcreatingusingdataviews.htm

Il existe deux façons de créer un DataView.:

* par le constructeur de DataView qui peut être vide, mais peut également accepter soit un DataTable comme argument unique, soit un DataTable avec des critères de filtre (**RowFilter**) et de tri (**Sort**), et un filtre d'état de ligne (**RowStateFilter**)
* en créant une référence à la propriété **.DefaultView** du DataTable

Exemple WebForm8 :

Création du DataSet et des DataView

private DataSet CreerDataSet()

{

OleDbConnection connexDB;

OleDbCommand commDB;

OleDbDataAdapter adaptDB;

DataSet leds;

connexDB = new OleDbConnection("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBS;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = "select \* from authors";

adaptDB = new OleDbDataAdapter(commDB);

leds = new DataSet();

try {adaptDB.Fill(leds, "les auteurs");}

catch(Exception ex) {throw(ex);} // remplir une première table

connexDB.Close();

return leds;

}

private DataView CreerDataView()

{

DataView ledv = new DataView(ds.Tables["les auteurs"]);

return ledv;

}

private void Page\_Load(object sender, System.EventArgs e)

{

ds= CreerDataSet();

dv1= CreerDataView();

dv2= CreerDataView();

}

Visualisation des DataView dans le source WebForm8.aspx:

Dans cet exemple on va utiliser un web contrôle du type **Repeater**, qui a comme intérêt de se lier à une source de données comme le DataList et d'avoir des modèles d'en tête et de contenu, mais aussi de pouvoir intégrer du code HTML (tel que les balises de tableau) mélangé à la déclaration de son modèle.

<asp:Repeater id="Repeater1" runat="server">

<HeaderTemplate>

<table border="1">

<tr><td><b>Nom</b></td><td><b>Prenom</b></td></tr>

</HeaderTemplate>

<ItemTemplate>

<tr>

<td><%# DataBinder.Eval(Container.DataItem, "au\_lname") %></td>

<td><%# DataBinder.Eval(Container.DataItem, "au\_fname") %></td>

</tr>

</ItemTemplate>

<FooterTemplate>

</table>

</FooterTemplate>

</asp:Repeater>

et encore:

<asp:Repeater id="Repeater2" runat="server">

<HeaderTemplate>

Les états : <br>

</HeaderTemplate>

<ItemTemplate>

<%# DataBinder.Eval(Container.DataItem, "city") %>

<b><i>(<%# DataBinder.Eval(Container.DataItem, "state") %>)</b></i>

</ItemTemplate>

<SeparatorTemplate>

**,**

</SeparatorTemplate>

</asp:Repeater>

Modifications des DataView:

Sur des boutons d'action, on peut poser des tris et des filtres tels que:

*Tri:*  dv1.Sort="au\_fname ASC";

puis Repeater1.DataBind();

*Filtre*: dv2.RowFilter=""; // plus de filtre

ou

dv2.RowFilter="state='CA'";

puis Repeater2.DataBind();

* **Les commandes sur les procédures stockées**

L'objet Command se paramètre lorsqu'il réfère à une procédure stockée elle même paramétrée.

Par exemple la base Pubs de SQL Server possède une procédure créée de la sorte:

CREATE PROCEDURE byroyalty @percentage int

AS

select au\_id from titleauthor

where titleauthor.royaltyper = @percentage

Exemple WebForm9 :

Dans ce cas la connexion et la création du DataReader se déroulent comme précédemment, avec l'objet Command renseigné pour la procédure:

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType **= CommandType.StoredProcedure;**

commDB.CommandText = "byroyalty";

**OleDbParameter param = commDB.Parameters.Add("@percentage", OleDbType.Integer, 4);**

**param.Value = "50";**

Visualisation des résultats dans le source WebForm9.aspx

Dans cet exemple on va utiliser un web contrôle du type **Table**

<**asp:Table** id="Table1" BorderColor="#009933" ForeColor="#339900" BorderStyle="Double" runat="server">

<**asp:TableRow**>

<**asp:TableCell** Font-Underline="True"> ID auteurs à 50%

</asp:TableCell>

</asp:TableRow>

</asp:Table>

qui va être rempli avec le DataReader, résultat de l'exécution de la commande:

while (rd.Read())

{



**TableRow** ligne= new TableRow();

**TableCell** cellule = new TableCell();

cellule.Text = **rd["au\_id"].ToString();**

ligne.Cells.Add(cellule);

Table1.Rows.Add(ligne);

}

rd.Close(); Ce qui donnera à l'affichage:

* **Les commandes qui retournent une valeur scalaire**

La méthode **.ExecuteScalar()** de l'objet Command extrait une valeur unique, la première colonne de la première ligne du jeu de résultats (par exemple, une valeur d'agrégation) d'une source de données.

Les colonnes ou lignes supplémentaires sont ignorées. Ce type d'exécution concerne les procédures stockées ou les instructions SQL qui effectuent une recherche, calculent une valeur d'agrégation ou ont pour résultat une valeur unique.

Exemple WebForm9:

int CalculerMoyenne()

{

OleDbConnection connexDB;

OleDbCommand commDB;

Int32 val;

connexDB = new OleDbConnection("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBS;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = **"SELECT avg(royaltyper) FROM titleauthor";**

try {**val = (Int32)commDB.ExecuteScalar();}** catch(Exception ex) {throw(ex);}

return val;

}

* **Conclusion**

###### Nous avons vu les principales classes d'ADO.NET:

|  |  |
| --- | --- |
| **Objet** | **Description** |
| **Connection** | Établit une connexion à une source de données spécifique. |
| **Command** | Exécute une commande sur une source de données. |
| **DataReader** | Lit un flux de données en avant seulement et en lecture seule à partir d'une source de données. |
| **DataAdapter** | Remplit un DataSet et répercute les mises à jour dans la source de données. |

Mais ADO.NET propose également:

|  |  |
| --- | --- |
| **Objet** | **Description** |
| **Transaction** | Vous permet d'inscrire des commandes dans des transactions au niveau de la source de données. |
| **CommandBuilder** | Objet d'assistance qui générera automatiquement les propriétés de commande d'un DataAdapter ou dérivera les informations sur les paramètres à partir d'une procédure stockée et remplira la collection Parameters d'un objet Command. |
| **Parameter** | Définit les paramètres des valeurs d'entrée, de sortie et de retour pour les commandes et les procédures stockées. |
| **Exception** | Retourné en cas d'erreur au niveau de la source de données. En cas d'erreur côté client, les fournisseurs de données .NET lèvent une exception .NET Framework. |
| **Error** | Expose les informations provenant d'un avertissement ou d'une erreur retournés par une source de données. |
| **ClientPermission** | Fourni pour les attributs de sécurité d'accès du code du fournisseur de données .NET. |

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconusingadonetproviderstoaccessdata.htm

#### LA MODIFICATION DES DONNEES DANS LA BASE

Les exemples précédents ont présenté divers modes (connecté ou non) d'accès aux données afin de les publier sur une page aspx dans divers contrôles d'affichages – DataDrid, ListGrid, Repeater liés aux données, Table, …

ADO.NET permet également de modifier les données de la base; deux possibilités se présentent :

1°) par l'intermédiaire d'un **groupe de données** (DataSet) chargé par un adaptateur de données. Les manipulations (ajout, retrait ou mise à jour d'enregistrements) sont alors apportées dans le cache mémoire du DataSet et ensuite on utilise l'adaptateur pour reporter ces modifications dans la base.

2°) en accédant directement dans la base de données par une **commande de données**, configurée avec une instruction SQL ou le nom d'une procédure stockée, et ensuite exécutée par sa méthode **.ExecuteNonQuery()** qui retourne le nombre de lignes affectées. C'est la seule méthode lorsqu'il s'agit de modifier la structure de la base, par exemple créer des éléments nouveaux tels que des tables.

La première méthode sera pleinement intéressante dans des applications Windows traditionnelles (WinForms), où le client pourra travailler en mode déconnecté de la base, effectuer tranquillement les manipulations sur le DataSet et ensuite mettre à jour la base.

Dans les applications Web (WebForms), il faut se rappeler que le DataSet est perdu entre les allers retours de la page, et que de ce fait il est plus intéressant d'utiliser les objets de commandes pour modifier la base.

Notes:

Les événements provenant des **contrôles imbriqués** (tels que **Repeater**, **DataList** et **DataGrid**) sont propagés, c'est-à-dire qu'ils sont envoyés au conteneur, au lieu d'être déclenchés individuellement par chaque bouton. Le conteneur déclenche à son tour un **événement générique intitulé ItemCommand** avec les paramètres vous permettant de découvrir le contrôle qui a déclenché l'événement d'origine. En répondant à ce seul événement, vous évitez de devoir écrire des gestionnaires d'événements pour les contrôles enfants.

L'événement ItemCommand comporte les deux arguments d'événement standard, un objet référençant la source de l'événement et un objet d'événement contenant des informations relatives à l'événement

Remarque:   Les contrôles serveur Web DataGrid et DataList prennent en charge des événements supplémentaires, tels que **EditCommand**, **DeleteCommand** et **UpdateCommand**, qui sont des événements propagés particuliers.

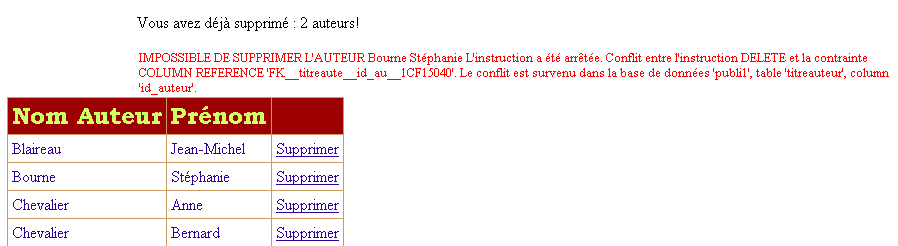
Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconwebformseventmodel.htm

Exemple WebForm10:

Dans cet exemple, la table des auteurs de la base de données "Publi" de SQL Server 2000 sera visualisée par un **contrôle DataGrid.** Ce DataGrid présentera les noms et prénoms de chaque auteur et donnera pour chacun un bouton de suppression.

Il faudra gérer les exceptions en cas d'échec à la suppression (auteur lié à une autre table …. )



.

La requête de suppression ressemblera à :

delete from auteurs where id\_auteur='444-77-8888'

La DataGrid va être liée à un DataSet comme dans les exercices précédents; elle contiendra les noms, prénoms et id des auteurs, l'id étant invisible, ainsi qu'une colonne de boutons de suppression créée à l'aide du générateur de propriétés:

<asp:DataGrid id="DataGrid1 runat="server" ….. AutoGenerateColumns="False">

<SelectedItemStyle Font-Bold="True" ForeColor="#663399" BackColor="#FFCC66"></SelectedItemStyle>

<ItemStyle ForeColor="#330099" BackColor="White"></ItemStyle>

<HeaderStyle Font-Size="Large" … BackColor="#990000"></HeaderStyle>

<FooterStyle ForeColor="#330099" BackColor="#FFFFCC"></FooterStyle>

<Columns>

<asp:**BoundColumn** **DataField="nom\_auteur"** HeaderText="Nom Auteur"></asp:BoundColumn>

<asp:**BoundColumn** **DataField="pn\_auteur"** HeaderText="Pr&#233;nom"></asp:BoundColumn>

<asp:**ButtonColumn** **Text="Supprimer" CommandName="Delete"></**asp:ButtonColumn>

<asp:**BoundColumn** **DataField="id\_auteur" Visible="False"**></asp:BoundColumn>

</Columns>

<PagerStyle HorizontalAlign="Center" ForeColor="#330099" BackColor="#FFFFCC"></PagerStyle>

</asp:DataGrid>

L' événement de suppression sera traité de la sorte:

private void **DataGrid1\_DeleteCommand**

**(object source, System.Web.UI.WebControls.DataGridCommandEventArgs e)**

{

Label2.Text="";

// rechercher l'ID de l'auteur à détruire

**string id = (string) e.Item.Cells[3].Text;**

// crée la requete sql

**string sql = "delete from auteurs where id\_auteur='"+ id +"'";**

// établit la connexion sur la base

OleDbConnection connexDB;

connexDB = new OleDbConnection("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBLI1;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

// exécute la destruction de la ligne

OleDbCommand commDB;

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = sql;

try {**commDB.ExecuteNonQuery();**}

catch(Exception ex)

{

Label2.Text = "IMPOSSIBLE DE SUPPRIMER L'AUTEUR "

**+ (string) e.Item.Cells[0].Text** +" "+ (string) e.Item.Cells[1].Text;

Label2.Text += "\n" + **ex.Message;**

return;

}

// supprime la ligne dans la grille en la réactualisant

commDB.CommandType = CommandType.Text;

commDB.CommandText = "select \* from auteurs order by nom\_auteur, pn\_auteur";

OleDbDataAdapter adaptDB;

DataSet leds;

adaptDB = new OleDbDataAdapter(commDB);

leds = new DataSet();

try {adaptDB.Fill(leds, "les auteurs");} catch(Exception ex) {throw(ex);}

DataGrid1.DataSource = leds;

DataGrid1.DataBind();

connexDB.Close();

**Hidden1.Value = (Int32.Parse(Hidden1.Value)+1).ToString();**

Label1.Text = "Vous avez déjà supprimé : " + Hidden1.Value + " auteur" + (Hidden1.Value =="1"?"":"s") + "!";

}

**Remarquer le champ hidden pour tenir à jour un compteur entre les allers retours de la page.**

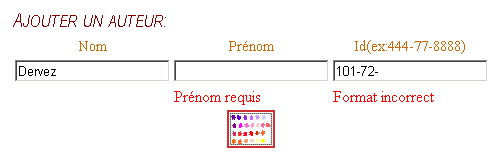
Exemple WebForm10 - suite:

La suite de l'exemple consiste à rajouter des auteurs dans la table. Les champs seront saisis dans des zones de texte puis contrôlés comme étant non vides et du format adéquat pour l'identificateur de l'auteur.

La requête d'insertion ressemblera à :

insert into auteurs(id\_auteur, nom\_auteur, pn\_auteur, contrat)

values ('445-77-8888', 'tutu', 'titi',1 )



Tous les composants (textes et validateurs) seront organisés dans une table qui sera positionnée dans la page par l'attribut style. Remarquer l'expression régulière du contrôle **RegularExpressionValidator** qui impose le format "xxx-xx-xxxx" x décimal entre 0 et 9

<table style="Z-INDEX: 103; LEFT: 467px; POSITION: absolute; TOP: 242px">

<tr style="FONT-SIZE: 20px; COLOR: maroon; FONT-STYLE: italic; FONT-FAMILY: 'Arial Narrow';

FONT-VARIANT: small-caps">Ajouter un auteur:</tr>

<tr style="FONT-SIZE: 15px; COLOR: #cc6600; TEXT-ALIGN: center">

<td><asp:Label id="Label3" runat="server">Nom</asp:Label></td> <td><asp:Label id="Label4" runat="server">Prénom</asp:Label></td>

<td><asp:Label id="Label5" runat="server">Id(ex:444-77-8888)</asp:Label></td>

</tr>

<tr>

<td><asp:TextBox id="TextBox1" runat="server"></asp:TextBox></td>

<td><asp:TextBox id="TextBox2" runat="server"></asp:TextBox></td>

<td><asp:TextBox id="TextBox3" runat="server"></asp:TextBox></td>

</tr>

<tr>

<td><asp:RequiredFieldValidator id="RequiredFieldValidator1" ControlToValidate="TextBox1"

runat="server" ErrorMessage="Nom de famille requis"></asp:RequiredFieldValidator></td>

<td><asp:RequiredFieldValidator id="RequiredFieldValidator2" ControlToValidate="TextBox2"

runat="server" ErrorMessage="Prénom requis"></asp:RequiredFieldValidator></td>

<td**><asp:RequiredFieldValidator** id="Requiredfieldvalidator3" **ControlToValidate="TextBox3"**

runat="server" **ErrorMessage="Identification requise" Display="Dynamic">**

</asp:RequiredFieldValidator>

<asp:**RegularExpressionValidator** id="Regularexpressionvalidator1" runat="server"

**ValidationExpression="\d{3}[-]\d{2}[-]\d{4}"** **ControlToValidate="TextBox3"**

**ErrorMessage="Format incorrect" Display="Dynamic"></**asp:RegularExpressionValidator></td>

</tr>

<tr>

<td colspan="3" align="middle"><**asp:ImageButton** id="Imagebutton1" BorderColor="#cc3333"

BorderStyle="Double" width="40" Height="30" runat="server" ImageUrl="Images/TINYARRW.WMF"></asp:ImageButton></td>

</tr>

</table>

L' événement du bouton image sera traité de la sorte:

private void **Imagebutton1\_Click(object sender, System.Web.UI.ImageClickEventArgs e)**

{

// crée la requete sql

**string sql = "insert into auteurs(id\_auteur, nom\_auteur, pn\_auteur, contrat) values**

**('"+ TextBox3.Text +"', '"+TextBox1.Text+"', '"+TextBox2.Text+"',1 )";**

// établit la connexion sur la base

OleDbConnection connexDB;

connexDB = new OleDbConnection("Provider=SQLOLEDB;server=APISERVEURBLEU;

Initial Catalog=PUBLI1;User Id=stage;Password=stage;");

try {connexDB.Open();} catch(Exception ex) {throw(ex);}

// exécute la création de la ligne

OleDbCommand commDB;

commDB = new OleDbCommand();

commDB.Connection = connexDB;

commDB.CommandType = CommandType.Text;

commDB.CommandText = sql;

try {**commDB.ExecuteNonQuery();**}

catch(Exception ex)

{

Label2.Text = "IMPOSSIBLE DE CREER L'AUTEUR " +TextBox1.Text +" "+TextBox2.Text;

Label2.Text += "\n" + ex.Message;

return;

}

// rajoute la ligne dans la grille en la réactualisant

commDB.CommandType = CommandType.Text;

commDB.CommandText = "select \* from auteurs order by nom\_auteur, pn\_auteur";

OleDbDataAdapter adaptDB;

DataSet leds;

adaptDB = new OleDbDataAdapter(commDB);

leds = new DataSet();

try {adaptDB.Fill(leds, "les auteurs");} catch(Exception ex) {throw(ex);}

DataGrid1.DataSource = leds;

DataGrid1.DataBind();

connexDB.Close();

}

#### LA GESTION DES ETATS

* **Première problématique:**

Lors de la conception de l'accès aux données de votre application Web, vous devez faire différents choix concernant la façon de communiquer avec une source de données, la décision de stocker les données entre les allers-retours sur la page et, dans ce cas, l'emplacement du stockage.

Les questions suivantes:

* Groupe de données (DataSet), ou accès direct et lecteur de données (DataReader) ?
* Enregistrer le groupe de données ou le recréer à chaque fois ?
* Mettre en cache sur le serveur ou sur le client ?

ont des réponses dans le chapitre suivant:

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/vbconDecidingOnDataAccessStrategy.htm

Un moyen classique de stocker et de restaurer un groupe de données est d'utiliser l**'état de session.**

* **Deuxième problématique:**

Puisque les pages Web sont recréées chaque fois que la page est publiée sur le serveur, toutes les informations associées à la page et à ses contrôles devraient être perdues à chaque aller-retour.

Pour y remédier, ASP.NET comprend une option appelée **état d'affichage**qui conserve automatiquement les valeurs des propriétés de la page et tous ses contrôles entre les allers-retours.

D'autres options de **gestion d'état** permettent également de sauvegarder à le demande des variables de la page entre ses différents aller-retours, soit en conservant les informations sur le client, soit en les stockant sur le serveur.

* **Gestion des états côté client:**

1. L'état d'affichage: c'est la méthode utilisée par défaut par ASP. Ainsi la propriété **.ViewState** de la classe **Control** obtient un dictionnaire d'informations d'état qui permet d'enregistrer et de restaurer l'état d'une information – entre autre la valeur affichée par un contrôle - entre plusieurs demandes de la même page.

On se rappelle le premier formulaire créé qui a généré ce code html sur le navigateur du client:

<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">

<input type="hidden" **name="\_\_VIEWSTATE"**

value="dDwtMTc0MDc5ODg1Mzs7PjqkTgPWFH5Uc4LyOSLLUACAvwRU" />  **…..**

1. Le champs de formulaire masqué: son contenu est alors renvoyé dans la collection Form de la requête http. Il faut alors ajouter dans le formulaire un contrôle **HtmlInputHidden** qui pourra être ou non traité aussi du côté serveur.

<INPUT type="hidden" id="Hidden1" name="Hidden1" runat="server">

1. Le cookie: il permet de stocker une petite information dans la mémoire du navigateur (cookie temporaire) ou sur le disque du client (cookie permanent). Le navigateur renvoie systématiquement les informations au seul serveur qui est à l'origine du cookie; le serveur peut alors extraire les informations de la collection Cookies de la requête http.
2. La chaîne de requête: il s'agit d'une information rajoutée à la suite de l'URL de la page (longueur souvent limitée à 255 caractères), soit par publication en méthode GET soit par réécriture de l'attribut "action" du formulaire. Le serveur récupère l'information dans la collection QueryString de la requête http.

* **Gestion des états côté serveur:**

1. L'état de l'application: c'est un espace de stockage d'informations accessible à toutes les pages de l'application Web (c'est à dire les pages d' un même répertoire virtuel), nommé "Application". On dispose ainsi d'informations globales à l'application.

Des procédures décrites dans le fichier **global.asax** peuvent être déclenchées au démarrage et à l'arrêt de l'application Web:

<script language="C#" runat="server">

public void **Application\_OnStart**() {

// Application start-up code goes here.

}

public void **Application\_BeginRequest**() {

// Application code for each request could go here.

}

public void **Application\_EndRequest**() {

//

}

public void **Application\_OnEnd**() {

// Application clean-up code goes here.

}

</script>

Application\_BeginRequest() est déclenché lors d'une demande de page Web Forms ou de service Web XML de l'application. Cet événement permet d'initialiser les ressources qui seront utilisées pour chaque demande adressée à l'application.

Les informations étant globales à toutes les instances de pages de l'application, il peut y avoir des soucis de synchronisation pour y accéder. Dans ce cas des méthiodes Lock() et UnLock() protègent les accès aux données de l'application:

Application.Lock();

Application["var1"] = …..

Application["var2"] = C…..

Application.UnLock();

[JScript]

Application.Lock()

Application["MyCode"] = 21

Application["MyCount"] = Convert.ToInt32(Application["MyCount"]) + 1

Application.UnLock()

[C++] Aucun exemple n'est disponible pour C++. Pour afficher un exemple Visual Basic, C# ou JScript, cliquez sur le bouton Filtre de langage dans le coin supérieur gauche de la page.

1. L'état de session: c'est un espace de stockage d'informations accessible à toutes les pages de l'application Web mais qui sont publiées dans la session en cours d'un navigateur client.

Chaque session ASP.NET active est identifiée et suivie à l'aide d'une chaîne SessionID communiquée sur les demandes client-serveur au moyen d'un cookie HTTP ou d'une URL modifiée.

Les données à conserver peuvent être stockées dans la collection **Session** propre à chaque client afin que ce dernier les récupère lorsqu'il revisite re-visite le serveur.

Des procédures décrites dans le fichier **global.asax** peuvent être déclenchées au démarrage et à l'arrêt de la session utilisateur:

<script language="C#" runat="server">

public void **Session\_OnStart**(){

}

public void **Session\_OnEnd**() {

}

</script>

Une session commence quand un utilisateur demande une page de l'application pour la première fois et se termine quand l'application ferme explicitement la session ou quand celle-ci dépasse le délai imparti.

Les informations de session peuvent être stockées sur d'autres machines, voire dans une base SQL Server (voir le fichier de configuration web.config).

Conserver l'état à l'aide d'une base de données constitue une solution courante lorsqu'il s'agit de stocker des informations propres à l'utilisateur et que le stockage des informations représente un volume élevé. Le stockage de base de données est particulièrement utile pour gérer un état à long terme ou un état qui doit être conservé même si le serveur doit être redémarré.

Documentation du Framework .NET:

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/

vbconIntroductionToWebFormsStateManagement.htm

ms-help://MS.VSCC/MS.MSDNVS.1036/Vbcon/html/vbconchoosingserverstateoption htm

ms-help://MS.VSCC/MS.MSDNVS.1036/cpguide/html/cpconsessionstate.htm

* **Retour sur le fichier Global.asax:**
* Il est optionnel – il n'y aura alors pas de gestionnaires d'événements de session et d'application
* Il est rangé dans la racine de l'application
* Il peut cohabiter avec un fichier global.asa pour l'asp classique
* Sa mise à jour provoque un événement Application\_OnEnd et un redémarrage de l'application

Il contient par défaut une seule directive d'application:

**<%@ Application Codebehind="Global.asax.cs" Inherits="WebApplication1.Global" %>**

En effet la classe générée pour la page .aspx hérite de la classe **WebApplication1.Global** définie dans code-behind et qui elle même hérite de la classe **System.Web.HttpApplication**

Une application ASP.NET est la somme de tous les fichiers, pages, gestionnaires, modules et code situés dans la portée d'un répertoire virtuel et de ses sous-répertoires sur un même serveur Web.

Une instance unique d'une classe **HttpApplicationState** est créée la première fois qu'un client demande une ressource URL à partir du répertoire virtuel d'une application ASP.NET particulière. Une référence à chaque instance est alors exposée via l'objet intrinsèque **Application**.

La classe **System.Web.HttpApplication** définit les méthodes, propriétés et événements communs à toutes les pages au sein d'une application ASP.NET.

Quelques propriétés de la classeSystem.Web.HttpApplication:

|  |  |
| --- | --- |
| Application | Obtient l'état (collection de tous les fichiers, pages, gestionnaires, modules et code) et permet le partage d'informations globales. Méthodes Lock(), UnLock(), Add(), Remove(), Clear(), propriétés Count, Keys ….. |
| Context | Obtient des informations spécifiques au protocole HTTP concernant l'instance en cours de l'application. |
| Request | Obtient l'objet de demande intrinsèque de la demande en cours avec ses collections Cookies, Files, Form, QueryString, Headers, Params , ses propriétés ContentLength, ContentType, HttpMethod, UserHostAddress, UserHostName, Path, Url, RawUrlet ses méthodes MapPath() ….. |
| Response | Obtient l'objet de réponse intrinsèque de la demande en cours avec sa collection Cookies, ses propriétés Output, Status, ContentType, Expires, et ses méthodes Clear(), Redirect(), Write(), BinaryWrite(), WriteFile(), End(), ……. |
| Server | Obtient l'objet serveur intrinsèque de la demande en cours avec ses propriétés MachineName, ScriptTimeOut, et ses méthodes CreateObject() –pour les objets COM-, HtmlDecode(), HtmlEncode(), UrlDecode(), UrlEncode(), UrlPathEncode(),MapPath(), Transfert(), |
| Session | Obtient l'objet session intrinsèque qui fournit un accès aux données de session avec ses propriétés SessionId, TimeOut, Count, Keys et ses méthodes Abandon(), Add(), Remove(), Clear() … |

Exemple Index.asp:

Dans le fichier **Global.asax** de l’application WebApplication1, on trouve la définition de 2 variables d’application au démarrage :

*protected void* ***Application\_Start****(Object sender, EventArgs e)*

*{*

*Application["auteur"]="Barbara Le Jan";*

*Application["dateCreation"]="Janvier 2003";*

*}*

Ces variables peuvent s’utiliser de la sorte :

*<H2>Les exemples ASP.NET de* ***<%=Application["auteur"]%>*** *- <%=Application["dateCreation"]%>*

*</H2><br><hr>*

*Ce serveur s'appelle* ***<%= Server.MachineName%>****et vous êtes connecté depuis l'adresse* ***<%=Request.UserHostAddress%>*** *<br>*

De même il est possible de faire un menu traité dans le fichier de code–behind par des redirections :

*private void RadioButtonList1\_SelectedIndexChanged(object sender, System.EventArgs e)*

*{*

***Server.Transfer****("./"+RadioButtonList1.SelectedItem.Value+".aspx");*

*}*

*private void LinkButton1\_Click(object sender, System.EventArgs e)*

*{*

***Response.Redirect****("/appliASP-1/premier.aspx");*

*}*

REMARQUE : Les redirections de pages aspx ne font par le mécanisme des exceptions de type **System.Threading.ThreadAbortException** ; donc il faut veiller à bien les dissociers des autres exceptions qui seraient gérées dans des blocs try … catch. (voir l’exemple WebForm22)

#### La télétransmission des fichiers clients : l' UPLOAD

Cette technique n'était pas proposée en standard sous ASP.

ASP.NET fournit une classe **HttpFileCollection** qui permetent d'accéder aux fichiers transférés à partir du client sous la forme d'une collection de fichiers individuels de la classe **HttpPostedFile** que l'on peut lire (propriété *InputStream*) ou enregistrer (méthode *SaveAs*).

Les fichiers sont téléchargés au format MIME multipart/form-data, d'où l'attribut *encType="multipart/form-data"* de la balise form de la page du client. Les fichiers transportés dans la tramme http sont récupérés côté serveur dans **la collection Files de l'objet Request** :

*HttpFileCollection collFic = Request.Files;*

*if (collFic[0].FileName != "")*

*string stFic ="c:\\temp\\"+ Path.GetFileName(collFic[0].FileName)*

*collFic[0].SaveAs(stFic);*

Le formulaire doit comporter un ou plusieurs éléments HTML **<input type= file>** qui permettront au client de rechercher sur son disque le(s) fichier(s) à transmettre au serveur.

Côté serveur, la classe **HtmlInputFile** permet un accès par programme à l'élément HTML <input type= file>. Voici donc un autre moyen de récupérer le fichier par la propriété PostedFile:

*if (File1.PostedFile. FileName != "")*

*// File1 est du type HtmlInputFile, et sa propriété rend un objet de HttpPostedFile*

*{*

*string stFic ="c:\\temp\\"+ Path.GetFileName(File1.PostedFile.FileName)*

*File1.PostedFile.SaveAs (stFic);*

*}*

**Attention à vérifier les permissions NTFS sur le répertoire du serveur: rajouter l'utilisateur ASPNET (compte aspnet\_wp) de la machine serveur en accès "modifier" pour pouvoir créer les fichiers.**

Exemple WebForm11

<form id="Form1" encType="multipart/form-data" method="post" runat="server">

<INPUT type="file" id="File1" name="File1" runat="server">

<INPUT type="file" id="File2" name="File2" runat="server">

<INPUT type="submit"value="GO" id="Submit1" name="Submit1" runat="server">

</form>

et le traitement suivant:

private void Submit1\_ServerClick(object sender, System.EventArgs e)

{

HttpFileCollection collFic = Request.Files;

**…..**

if (File1.PostedFile.FileName != "")

**……**

if (File2.PostedFile.FileName != "")

**……**

}

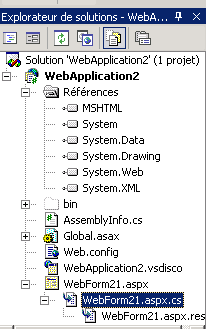
#### Ajouter des références DE COMPOSANTS au projet

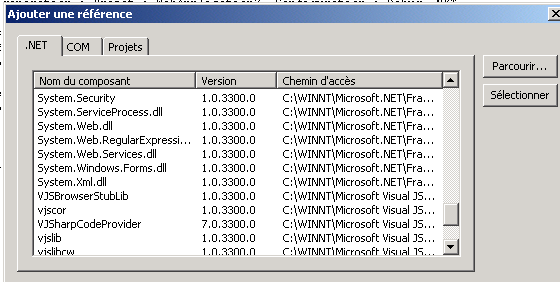
Certains composants peuvent être utilisés sous condition que leur référence soit explicitement ajoutée au projet; 3 options:

* .NET - Répertorie tous les composants .NET Framework pouvant être référencés.
* COM - Répertorie tous les composants COM pouvant être référencés.
* Projets - Répertorie tous les composants réutilisables créés dans des projets locaux.

Dans la fenêtre explorateur de projet, faire un ajout sur le noeud des références (bouton droit) et choisir "**System.Xml.dll**" dans l'onglet "**NET**"

On s'aperçoit que la bibliothèque (ou espace de noms) "System.Xml" est ajoutée.





On remarquera que si l'on avait rajouté une référence COM, la dll aurait alors été est copiée dans les répertoires suivants:

C:\Documents and Settings\blj\VSWebCache\APINET\Demarrage-ASP-NET\WebApplication2\obj (emplacement du cache) et C:\Inetpub\wwwroot\Demarrage-ASP-NET\WebApplication2\bin

La référence, avec ses classes et leurs membres, est alors utilisable dans le projet, et l'éditeur présente les membres par l'IntelliSense.

Exemple d'utilisation pour lire et afficher le contenu d'un fichier XML (exemple pris dans la documentation Microsoft):

Recherche du chemin physique

Exemple WebForm12 – projet WebApplication1:

string nomFichier **= Server.MapPath (".")+ "/doc/Fleurs.xml"**;

**System.Xml.XmlTextReader lect = new System.Xml.XmlTextReader(nomFichier);**

// ne pas retourner l'espace blanc

lect.WhitespaceHandling = System.Xml.WhitespaceHandling.None;

//lit le noeud suivant tant qu'il en reste

**while (lect.Read())**

{

switch (**lect.NodeType**) // quel type de noeud est-ce?

{

case System.Xml.XmlNodeType.Element:

txtInner.Value += "<" + lect.Name + ">";

break;

case System.Xml.XmlNodeType.Text:

txtInner.Value += lect.Value;

break;

case System.Xml.XmlNodeType.CDATA:

txtInner.Value += "<![CDATA[" + lect.Value + "]]>";

break;

case System.Xml.XmlNodeType.ProcessingInstruction:

txtInner.Value += "<?"+ lect.Name + lect.Value +"?>";

break;

case System.Xml.XmlNodeType.Comment:

txtInner.Value += "<!--" + lect.Value + "-->\n" ;

break;

case System.Xml.XmlNodeType.XmlDeclaration:

txtInner.Value += "<?xml version='1.0'?\n>";

break;

case System.Xml.XmlNodeType.Document:

break;

case System.Xml.XmlNodeType.DocumentType:

txtInner.Value += "<!DOCTYPE " +lect.Name +"[" + lect.Value + "]" ;

break;

case System.Xml.XmlNodeType.EntityReference:

txtInner.Value += lect.Name;

break;

case System.Xml.XmlNodeType.EndElement:

txtInner.Value += "</" + lect.Name + ">\n";

break;

}

}