M19 : Programmation Cl/Sr ADO.Net / C#

Formateur : Driouch B.
(cfmoti.driouch@gmail.com)
Etablissement : OFPPT/GC/CFMOTI
Année Scolaire : 2011/2012 (10/02/2012)
http://www.ista-ntic.net/

Plan du Cours

- Définition (Généralité, Architecture N-tier)
- Framework .Net / ADO.Net
- Connexion à une BD (Connection)
- Mode Connecté (Commad, DataReader)
- Mode Déconnecté (DataSet, DataAdapter)
- Mode Déconnecté (DataTable, DataColumns, DataRows, DataRelation, DataView)
- Contrôle utilisateur
- Utilisation Procédure Stocker et Transaction
- DataSet Typé (fichier XSD)
- Édition des états (CrystalReport)
- Déploiement

Présentation

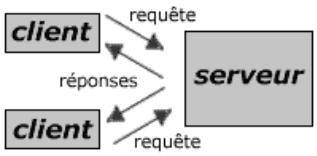
- De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des **machines clientes** contactent un **serveur**, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des **services**. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion,…
- Les services sont exploités par des programmes, appelés **programmes clients**, s'exécutant sur les machines clientes. On parle ainsi de client FTP, client de messagerie, ..., lorsque l'on désigne un programme, tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès du serveur (dans le cas du client FTP il s'agit de fichiers, tandis que pour le client messagerie il s'agit de courrier électronique).

Avantages / Inconvénients

- Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont:
  - **des ressources centralisées**
  - **une meilleure sécurité**
  - **une administration au niveau serveur**
  - **un réseau évolutif**
- L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles:
  - **un coût élevé**
  - **une maillon faible**

Fonctionnement

- Un système client/serveur fonctionne selon le schéma suivant:

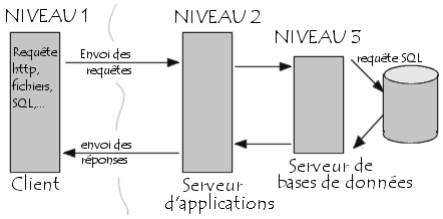


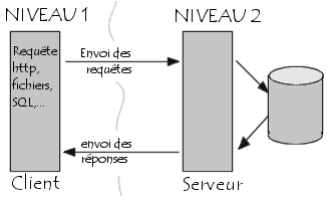
- Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port

DRIOUCH B. www.ista-ntic.net 5

Architecture à 2 et 3 niveaux

L'architecture à deux niveaux caractérise les systèmes clients/serveurs dans lesquels le client demande une ressource et le serveur la lui fournit directement.





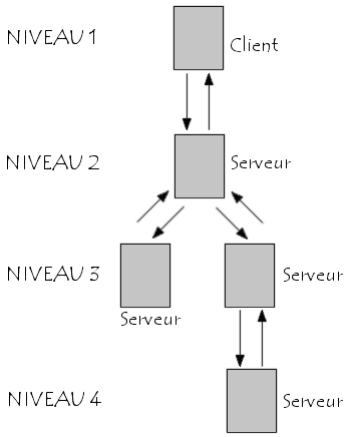
Dans l'architecture à 3 niveaux (appelées *architecture 3-tier*), il existe un niveau intermédiaire, c'est-à-dire que l'on a généralement une architecture partagée entre:

- Le client: le demandeur de ressources
- Le serveur d'application (appelé aussi **middleware**): le serveur chargé de fournir la ressource mais faisant appel à un autre serveur
- Le serveur secondaire (généralement un serveur de base de données), fournissant un service au premier serveur

DRIOUCH B. www.ista-ntic.net 6

Architecture multi-niveaux

Dans l'architecture à 3 niveaux, chaque serveur (niveaux 1 et 2) effectue une tâche (un service) spécialisée. Ainsi, un serveur peut utiliser les services d'un ou plusieurs autres serveurs afin de fournir son propre service. Par conséquence, l'architecture à trois niveaux est potentiellement une architecture à N niveaux...



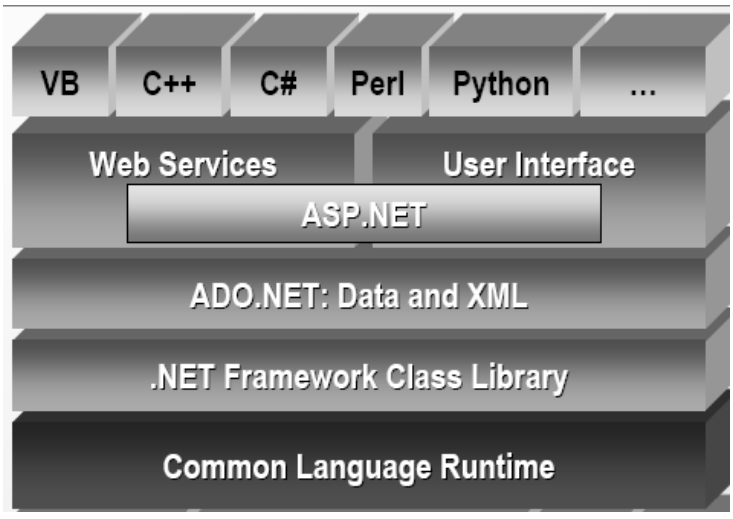
Framework Dotnet

- .NET est une plate forme complète pour développer, déployer et exécuter des applications Web, Windows, Mobiles et serveur. La plate forme repose sur la notion de framework qui est un cadre de travail offrant des outils et spécifications nécessaires pour développer et déployer des applications.
- Le Framework .NET est conçu pour remplir les objectifs suivants :
  - Fournir un environnement cohérent de programmation orientée objet que le code objet soit stocké et exécuté localement, exécuté localement mais distribué sur Internet ou exécuté à distance.
  - Fournir un environnement d'exécution de code qui minimise le déploiement de logiciels.
  - Fournir au développeur un environnement cohérent entre une grande variété de types d'applications comme les applications Windows et les applications Web.
  - Générer toutes les communications à partir des normes d'industries pour s'assurer que le code basé sur le Framework .NET peut s'intégrer à n'importe quel autre code.

Composants de .Net

- Le CLR (Common language Runtime) : il représente la machine virtuelle de la plate forme (un peu comme la JVM pour java). Il est responsable de l'exécution des applications et gère tous les aspects de sécurité : le garbage Collector, la gestion des événements etc.
- Un ensemble de librairie de classes : située au dessus de la CLR, cette rubrique offre une couche pour la gestion des données. On y retrouve les Windows Forms (WinForms), ensemble de classes permettant la conception d'IHM pour Windows (un peu comme AWT ou Swing).
- Adonet (Data and xml): une nouvelle génération de composants d'accès aux bases de données
- Nous avons ASP.NET qui fournit un ensemble de classe pour la conception de site dynamique, la création d'IHM pour le web, les WebForms et la conception de services web.
- Au sommet de la pile nous avons les langages supportés par .NET tels C#, VB.NET, C++, J# qui sont des produits de Microsoft et d'autres tels que Cobol, Delphi, etc… ; .NET offre ainsi un choix de langage contrairement à son concurrent J2EE.

Composants de .Net



ADO.Net

- **Que nous apporte ADO .NET ?**
- Une architecture plus optimisée: Avec .Net, de nouveaux fournisseurs de données voient le jour. Par exemple le .NET Framework comprend le fournisseur de données SQL Server .NET en mode natif via le namespace System.Data.SqlClient.
- Un meilleur support du mode déconnecté : Aujourd'hui Microsoft encourage le mode déconnecté et propose des classes spécialisées supportant les deux modes : connecté (via l'objet DataReader) et déconnecté (via l'objet DataSet).
- Un meilleur support de XML : Le XML est utilisé au sein du Framework .NET comme le standard de description et de persistance des données.

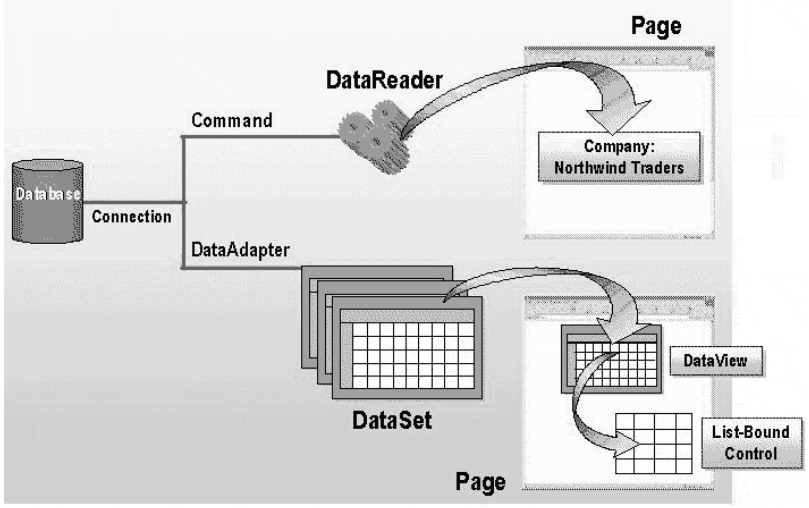
Choix d'un fournisseur de données .NET

- Pour pouvoir faire appel aux classes proposées par ADO .Net il est nécessaire d'inclure une référence à l'espace de noms correspondant. Vous pouvez soit inclure l'espace System.Data soit inclure des classes de cet espace comme System.Data.OleDb ou System.Data.SqlClient ; tout dépend de la source de données utilisée.

```
'Inclusion d'un namespace
using System.Data
using System.Data.SqlClient
```

Espace de nom	Description
System.Data.SqlClient	Fournisseur de données spécifiques pour SQLServer V7 ou supérieure
System.Data.OleDb	Propose la gestion de sources de données accédées via un driver OleDb
System.Data.Odbc	Propose la gestion de sources de données accédées via un driver Odbc
System.Data.OracleClient	Propose un accès à des sources de données Oracle (v8.1.7 ou supérieure)
System.Data.XML	Propose des classes permettant d'accéder à la fonctionnalité XML sous SQL Server 2000 ou supérieur
System.Data.SQLTypes	Propose des classes pour des types de données spécifiques à Microsoft SQL Server

Le modèle objet ADO .NET



Le modèle objet ADO .NET

- Regardons de plus près chacun de ces objets.

Objet	Description
Connection	Ouvre une connexion vers une source de données spécifique
Command	Exécute une commande sur une source de données
DataReader	Lit un flux de données à partir d'une source de données en mode **connecté**. Le mode d'accès est en lecture seule avec un curseur en avant seulement.
DataSet	Représente un ensemble de données en mode **déconnecté**. Il peut être constitué de plusieurs tables ainsi que des relations et contraintes existant entre elles.
DataAdapter	Remplit un **DataSet** et répercute les mises à jour dans la source de données.

Connexion à une base de données

- Modes de connexion

L'évolutions d'ADO.Net est plus importante du coté mode déconnecté afin de soulager d'une part le serveur BD et d'autre part d'épargner la bande passante, Le mode "déconnecté" permet au client d'effectuer des fonctions telles que le tri, le filtrage, sans avoir recours au serveur BD, grâce à une copie locale des données.

L'avantage du mode "déconnecté" sur le mode "connecté" est évident, ce dernier utilisant une connexion permanente avec la base de données.

Ce qui nous permet d'avoir deux modes de connexion (Mode connecté et Mode déconnecté)

Création de la connexion

- Pour déplacer des données entre une base de données et votre application, vous devez disposer d'une connexion préalable à la base de données. Pour créer une connexion à une base de données, vous devez identifier le nom du serveur de base de données, le nom de la base de données et les informations de connexion requises.
- En fonction du type de base de données à laquelle vous accédez, vous pouvez utiliser un objet **SqlConnection** ou **OleDbConnection**. Utilisez un objet SqlConnection pour la connexion aux bases de données SQL Server version 7.0 ou ultérieure et un objet OleDbConnection pour la connexion à toutes les autres bases de données.
- Pour créer un objet **SqlConnection,** passez une chaîne de connexion qui fournit les paramètres requis pour créer une connexion à une source de données.

```
string strConn = ''data source=localhost ;  initial catalog=GestStg ;
integrated security=true'';
Sqlconnection  conn = New SqlConnection(strConn);
conn.open();
conn.close();
```

Création de la connexion

- **Paramètres de la chaîne de connexion**

Paramètre	Description
Connexion Timeout (Délai de connexion)	Délai d'attente en secondes, d'une connexion au serveur avant de terminer la tentative de connexion et de générer une exception. Le délai par défaut est de 15 secondes
Data Source (Source de données)	Nom du serveur SQL Server à utiliser lorsqu'une connexion est ouverte ou nom de fichier à utiliser lors de la connexion à une base de données Microsoft Acces
Initial Catalog(Catalogue d'origine)	Nom de la base de données
Integrated Security(Sécurité intégrée)	Paramètre déterminant si une connexion doit être Sécurisée. Les valeurs possibles sont **True**, **False** et **SSPI**. **SSPI** correspond à **True**
Password (Mot de passe)	Mot de passe de connexion à la base de données SQL Server.
User ID (ID de l'utilisateur)	Nom du compte de connexion à SQL Server.
Provider(Fournisseur)	Propriété utilisée pour définir ou retourner le nom du fournisseur de la connexion ; ce paramètre est seulement utilisé pour les objets **OleDb Connection**
Persist Security Info (Persistance des informations de sécurité)	Lorsque ce paramètre a la valeur **False**, les informations relatives à la sécurité(le mot de passe, par exemple), ne sont pas retournées comme une partie de la connexion si la propriété a la valeur **True**, cela peut représenter un risque pour la sécurité. Le paramètre par défaut est **False**

Accès aux données en mode connecté

- L'objet Command

Une fois la connexion vers une base de données effectuée, vous pouvez exécuter une requête et récupérer son résultat en utilisant l'objet Command. La création d'un objet Command nécessite l'instanciation d'un objet **SqlCommand**. Cet objet expose différentes méthodes **Execute** à utiliser selon le résultat attendu :

- La méthode **ExecuteReader** peut être utilisée pour récupérer un jeu d'enregistrements et retourne un objet **DataReader**.
- La méthode **ExecuteScalar** récupère une valeur unitaire.
- La méthode **ExecuteNonQuery** exécute une commande ne retournant pas de lignes. Mais retourne une valeur numérique indiquant le nombre de ligne affecté.

Exemple (Command)

```
using System;
using System.Data.SqlClient ;
class Program
   {  static void Main(string[] args)
      {  string str="data source=.\\SQLExpress;initial catalog=GestStg;integrated
    security=true";
         string sql="INSERT INTO Matiere VALUES (11,'ADO.Net',3)";
         SqlConnection oCon=new SqlConnection(str);
         SqlCommand oComd=new SqlCommand(sql,oCon);
         try{
         oCon.Open();
         oComd.ExecuteNonQuery();
         oCon.Close();
         Console.WriteLine(("la commande est bien exécuté"));}
         catch(Exception e){
         Console.WriteLine(("L'erreur suivante a été rencontrée :" + e.Message));
         }
         Console.ReadKey();
      }
   }
```


Mode connecté

- l'objet DataReader

L'objet DataReader permet de récupérer d'une source de données un flux en lecture seule en avant seulement (read only, forward only). Il résulte de l'exécution de la méthode **ExecuteReader** sur un objet Command.

L'objet DataReader ne stocke en mémoire qu'une seule ligne à la fois, permettant ainsi d'augmenter les performances d'une application et d'en réduire la charge.

- Il est recommandé d'utiliser cet objet si :
  - Vous n'avez pas besoin de réaliser un cache des données
  - Vous traitez un jeu d'enregistrements trop important pour être stocké en mémoire
  - Vous souhaitez accéder à des données rapidement en lecture seule en avant seulement

Mode connecté

- Par défaut, un DataReader charge une ligne entière en mémoire à chaque appel de la méthode **Read**. Il est possible d'accéder aux valeurs de colonnes soit par leurs noms soit par leurs références ordinales. Une solution plus performante est proposée permettant d'accéder aux valeurs dans leurs types de données natifs (GetInt32, GetDouble, GetString) Par exemple si la première colonne de la ligne indicée par 0 est de type int, alors il est possible de la récupérer à l'aide de la méthode **GetInt32** de l'objet **DataReader**.
- La méthode **Close** ferme un objet DataReader. Précisons que si l'objet Command utilisé contient des paramètres en sortie ou des valeurs de retours, ils ne pourront être récupérés qu'a l'issue de la fermeture du DataReader.

Exemple (DataReader)

```
//Création de la connexion et de l'objet Command
SqlConnection conn=new SqlConnection("data source=localhost\SQLExpress ;
    integrated security=true ; initial catalog=GestStg");
SqlCommand cmdStg = new SqlCommand("select * from Stagiaire", conn);
conn.Open();
//Création d'un DataReader et affichage des données
SqlDataReader dr;
dr=cmdStg.ExecuteReader();
while(dr.Read()){
     Console.WriteLine(dr["nom"] + " - " + dr["moyenne"]);
  }
//Fermeture de DataReader et de la connexion
dr.Close();
conn.Close();
Console.ReadKey();
```

Gestion des erreurs

- Lors de l'utilisation de connexions avec l'objet DataReader, vous devriez toujours utiliser une instruction Try…catch…Finally pour garantir la fermeture des connexions en cas d'échec quelconque. Sinon, la connexion peut demeurer indéfiniment ouverte.
- Le code suivant d'un objet DataReader intercepte des erreurs et ferme la connexion

```
try{
     conn.Open();
     dr=cmdStg.ExecuteReader();
     //utilisation des données retournées dans les dataReader  }
catch (Exception e){
      //Gestion de l'erreur }
finally {
      //Fermeture de DataReader et de la connexion
      dr.Close();
      conn.Close();       }
```


Utilisation ComboBox

```
try
{
 oCon.Open();
 oComd.CommandText = "Select idstg,nom from stagiaire";
 oRdr = GLB.oComd.ExecuteReader();
 while (oRdr.Read())
  {
     comboBox1.Items.Add(oRdr["idstg"]);
  }
  oRdr.Close();
 }
catch (Exception e1)
 {
    MessageBox.Show("Erreur : " + e1.Message);
 }
finally {
 if (oCon.State == ConnectionState.Open) oCon.Close();
 }
```

Utilisation DataGridView

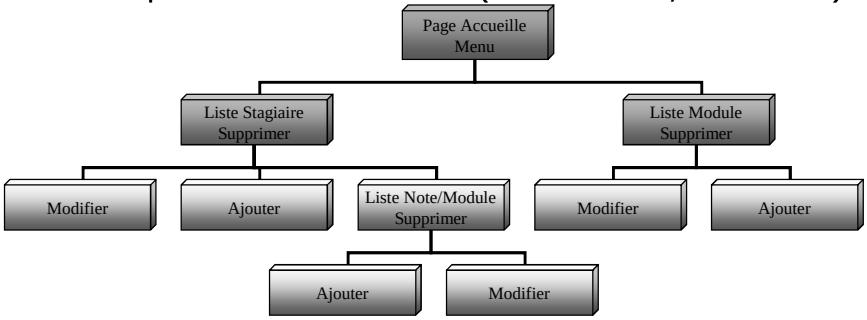
```
try {
 GLB.oCon.Open();
 GLB.oComd.CommandText = "Select m.idmat,libelle,coeff,note from note n inner join matiere m on
     n.idmat=m.idmat where idstg=" + comboBox1.SelectedItem.ToString();
 GLB.oRdr = GLB.oComd.ExecuteReader();
 dataGridView1.Columns.Clear();
 dataGridView1.Rows.Clear();
 dataGridView1.Columns.Add("id","ID");
 dataGridView1.Columns.Add("libelle","Matiere");
 dataGridView1.Columns.Add("coeff","Coefficient");
 dataGridView1.Columns.Add("note","Note");
while (GLB.oRdr.Read())
 {dataGridView1.Rows.Add(GLB.oRdr["idmat"], GLB.oRdr["libelle"], GLB.oRdr["coeff"], GLB.oRdr["note"]);
}
 GLB.oRdr.Close();
}
catch (Exception e2){
   MessageBox.Show("Erreur : " + e2.Message);
}
Finally {
   if (GLB.oCon.State == ConnectionState.Open) GLB.oCon.Close();
}
```


Exercice

Réaliser une application avec le schéma relationnel suivant

- Stagiaire (IdStg, nom, prenom, DateN, Groupe, moyenne)
- Matiere (IdMat, Libelle, Coeff)
- Note (IdStg, IdMat, Note)

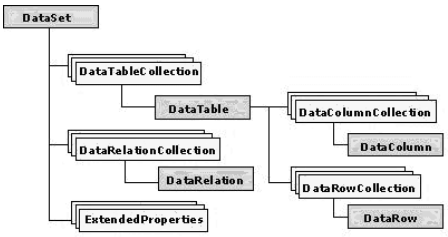
Et la conception de fenêtre suivant (DataGridview, ComboBox)



Mode Déconnecté

- DataSet:

L'objet DataSet représente une copie locale des données provenant d'au moins une source de données. L'objet DataSet peut utiliser un objet DataAdapter pour charger des données à partir d'une source de données et peut se déconnecter ensuite de cette dernière. L'utilisateur peut ensuite utiliser et manipuler les données. Lorsque les données doivent être mises à jour dans la source de données, un objet DataAdapter est utilisé pour se reconnecter à la source de données et la mettre à jour.



Modèle objet du Dataset

DataSet

- La collection "**DataTableCollection**"

Cette collection peut contenir de zéro à n objets de type DataTable. Chaque objet DataTable représente une table d'une source de données. Chaque DataTable est constituée d'une collection Columns et d'une collection Rows qui peuvent contenir respectivement de zéro à n objets DataRow et DataColumn.

- La collection "**DataRelationCollection**"

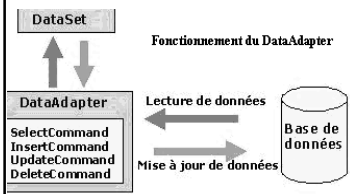
Cette collection peut contenir de zéro à n objets de type DataRelation. Un objet DataRelation définit une relation parent-enfant entre deux tables à partir des valeurs des clés étrangères.

- La collection "**ExtendedProperties**"

Cette collection correspond à un objet de type PropertyCollection qui peut contenir de zéro à n propriétés définies par un utilisateur. Cette collection peut être utilisée afin de stocker des informations personnalisées liées au DataSet utilisé (date et heure de génération des données, ordre select passé pour générer les données.).

DataAdapter

- Un objet DataSet doit être capable d'interagir avec une ou plusieurs sources de données. Pour réaliser cette action le framework Microsoft .Net fournit un objet nommé DataAdapter. L'objet DataAdapter sert de liaison entre un objet DataSet et une source de données. Un fournisseur de données .NET va se servir d'un objet DataAdapter pour remplir de données un DataSet puis répercuter les modifications réalisées sur une source de données. Il est possible de définir les actions à réaliser par le DataAdapter en utilisant l'une des quatre propriétés suivantes. Chaque propriété exécutera soit un ordre SQL soit une procédure stockée.



Propriété	Description
SelectCommand	La propriété SelectCommand permet de récupérer des données d'une source
InsertCommand	La propriété InsertCommand écrit les lignes insérées d'un DataSet vers une source de données
UpdateCommand	La propriété UpdateCommand écrit les lignes modifiées d'un DataSet vers une source de données
DeleteCommand	La propriété DeleteCommand efface les lignes supprimées d'un DataSet d'une source de données

Remplissage d'un DataSet

- La méthode Fill de l'objet DataAdapter permet d'extraire les données d'une source de données en exécutant la requête SQL spécifiée dans la propriété SelectCommand. Elle prend en paramètre le DataSet et le nom du DataTable à remplir avec les données retournées.

```
// Création d'un objet SqlDataAdapter
SqlDataAdapter oSqlDataAdapter =new SqlDataAdapter("select * from Stagiaire",
    oConnection);
DataSet oDataSet = new DataSet("GestStg");
oSqlDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
oSqlDataAdapter.Fill(oDataSet, "Stagiaire");
```

- La mise à jour de données avec un DataAdapter et un DataSet :

La méthode Update de l'objet DataAdapter permet de répercuter sur une source de données les modifications effectuées dans un DataSet. Cette méthode admet un objet DataSet qui contient les données modifiées et un objet DataTable optionnel qui contient les modifications à extraire.

```
oSqlDataAdapter.Update(oDataSet.Tables["stagiaire"]);
```

En réalité, lorsque la méthode Update est invoquée l'objet DataAdapter analyse les modifications effectuées au niveau de la collection DataRow. En fonction des modifications rencontrées, les commandes InsertCommand, UpdateCommand et DeleteCommand sont appelées par l'objet DataAdapter.

Création d'un DataSet

Pour créer une copie locale d'une base de données, créez et remplissez un objet **DataSet** au moyen d'objets **DataTable.** Pour créer un objet DataSet, vous devez d'abord déclarer le nom de l'objet DataSet. Le code suivant crée un objet DataSet nommé ds : DataSet ds = new DataSet();

- **Remplissage du DataSet**

Après avoir crée un objet DataSet, remplissez les objets DataTable en créant un objet DataAdapter. Vous pouvez appeler la méthode **Fill** de l'objet DataAdapter, puis préciser l'objet **DataTable** à remplir. Le code suivant remplit la table Authors (Auteurs) de l'objet DataSet intitulé **ds** en utilisant un DataAdapter nommé **da** : da.Fill(ds,''Stagiaire'');

- **Accès à un DataTable**

chaque objet DataSet se compose d'au moins un objet DataTable que vous pouvez référencer par nom ou par position ordinale :

ds.Tables[''Stagiaire''];

-Ou-

ds.Tables[0];

CommandBuilder

- Après remplissage d'un DataSet avec DataAdapter en utilisant l'objet SelectCommand, on peut utiliser les donnée pour les traitements de MAJ (Ajouter, Modifier et supprimer), et a la fin des différent traitements il faut faire appelle une 2eme fois le DataAdapter pour répercuter les modification du DataSet sur la base de donnée source, qui fait appel au trois autre objet (InsertCommand, UpdateCommand et DeleteCommand) selon l'etat(ligne Ajouter, Modifier ou supprimer) de chaque ligne du DataTable utilisé
- Pour effectuer cet opération, il faut construire les trois objets (InsertCommand, UpdateCommand et DeleteCommand), la difficulté de leur construction nous oblige a faire appel a un autre objet CommandBuilder qui permet leur construction a partir de SelectCommand

- **Utilisation de CommandBuilder**

```
SqlConnection StgCon = new SqlConnection("Data source=.\SQLExpress; integrated
    security=true;initial catalog=GestStg");
SqlDataAdapter StgDA = new SqlDataAdapter("", StgCon);
DataSet StgDS = new DataSet;
SqlCommandBuilder StgCB ;
StgDA.SelectCommand = new SqlCommand("Select idstg,nom,moyenne from stagiaire", StgCon);
StgDA.Fill(StgDS, "stg");
//Modification des donnée dans le DataSet
StgCB = new SqlCommandBuilder(StgDA);
StgDA.Update(StgDS.Tables("stg"));
```

DataRow et **DataColumn**

Utiliser un objet DataRow et ses propriétés et méthodes pour extraire et évaluer les valeurs d'un objet DataTable. Le DataRowCollection représente les objets DataRow réels présents dans l'objet DataColumn qui décrivent le schéma de l'objet DataTable. La propriété Rows de l'objet DataTable fournit un accès par programmation au DataRowCollection. La propriété Columns de l'objet DataTable fournit un accès par programmation au DataColumnCollection.

```
DataColumn col;
ForEach (col In ds.Tables[0].Columns){
        LstItems.Items.Add(col.ColumnName);
}
```

le nombre de lignes ou de colonnes d'un objet **DataTable**:

```
ds.Tables[''Stagiaire''].Rows.Count;
ds.Tables[''Stagiaire''].Columns.count;
```

accéder à ces champs par position ordinale (de base 0) Ou par nom :

```
DSet.Tables[0].Rows[x][1]
DSet.Tables[0].Rows[x][''nom_champ'']
```


DataTable (AutoIncrement)

```
DataTable Matable = new DataTable("stagiaire");
DataColumn MaColonne = new DataColumn("idstg", Type.GetType("System.Int32") );
    //typeof(int));     //Ajout de colonne
MaColonne.Unique = true;
MaColonne.AutoIncrement = true;
MaColonne.AutoIncrementSeed = 1;
MaColonne.AutoIncrementStep = 1;
MaColonne.AllowDBNull = false;
MaColonne.ReadOnly = true;
Matable.Columns.Add(MaColonne);
MaColonne = new DataColumn("nom", typeof(string));              //Ajout de colonne
Matable.Columns.Add(MaColonne);
DataRow MaLigne;                                    //Ajout de ligne
MaLigne = Matable.NewRow();
MaLigne["nom"] = "Ali";
Matable.Rows.Add(MaLigne);
MaLigne = Matable.NewRow();                         //Ajout de ligne
MaLigne["nom"] = "Ahmed";
Matable.Rows.Add(MaLigne);
dataGridView1.DataSource = Matable;                 //affichage dans DataGrid
```

DataRelation

```
DataSet MonData = new DataSet();
MonData.Tables.Add(Matable);              //Définition d'une clé primaire
MonData.Tables["stagiaire"].PrimaryKey = new DataColumn[]
    {MonData.Tables["stagiaire"].Columns["idstg"] };
Matable = new DataTable("note");
MaColonne = new DataColumn("idstg", typeof(int));
Matable.Columns.Add(MaColonne);
MaColonne = new DataColumn("note", typeof(float));
Matable.Columns.Add(MaColonne);
MaLigne = Matable.NewRow();               //Ajout de ligne
MaLigne["idstg"] = "1";
MaLigne["note"] = "12";
Matable.Rows.Add(MaLigne);
MaLigne = Matable.NewRow();               //Ajout de ligne
MaLigne["idstg"] = "1";
```


DataRelation

```
MaLigne["note"] = "13";
Matable.Rows.Add(MaLigne);
MonData.Tables.Add(Matable);
DataRelation Relation;                    // Définition de relation
Relation = new DataRelation("stgnote",
    MonData.Tables["stagiaire"].Columns["idstg"],
    MonData.Tables["note"].Columns["idstg"]);
MonData.Relations.Add(Relation);
DataRow Stgligne;                         //utilisation de Relation
DataRow[] Noteligne;
Relation = MonData.Tables["stagiaire"].ChildRelations["stgnote"];
Stgligne = MonData.Tables["stagiaire"].Rows[0];
Noteligne = Stgligne.GetChildRows(Relation);
dataGridView1.DataSource = MonData.Tables["stagiaire"];
dataGrid1.DataSource = MonData.Tables["stagiaire"];
```

DataView

- Représente une vue de DataTable personnalisée pouvant faire l'objet de liaisons de données pour le tri, la recherche, la modification et la navigation.
- Constructeur DataView(DataTable, String, String, DataViewRowState) : Initialise une nouvelle instance de la classe DataView avec les DataTable, RowFilter, Sort et DataViewRowState spécifiés.

```
DataView view = new DataView(dataSet.Tables["stagiaire"], "Nom Like 'A%'",
    "nom", DataViewRowState.CurrentRows);
 view.AllowEdit = true;
view.AllowNew = true;
view.AllowDelete = true;
```


DataBinding

DataSet et XML

XML est une format de source de donnée quand peut utilisé pour exporter et importer les données d'un DataSet

Si en prend l'exemple cité précédemment, DataSet avec les deux tables Stagiaire et note:

- Enregistrement dans un fichier XML avec le schéma (structure):

```
MonData.WriteXml("GestSTG.XML",XmlWriteMode.WriteSchema );
```

- La Récupération :

```
MonData = new DataSet();  //DataSet vide
MonData.ReadXml("GestSTG.XML", XmlReadMode.ReadSchema);
```


Gestion Erreur

```
try{        SqlConnection conn = new SqlConnection("...");
            SqlDataAdapter da = new SqlDataAdapter("...", conn);
            DataSet ds = new DataSet();
            da.Fill(ds);
          }
catch(SqlException  ex1){
            switch(ex1.Number){
               case 17:
                  LblErrors.Text = lblErrors.Text + "nom de serveur non valide";
               case 156, 170:  //syntaxe SQL incorrecte
                  LblErrors.Text = lblErrors.Text + "syntaxe incorrecte";
               case 207: //nom de champ incorrect dans select
                  LblErrors.Text = lblErrors.Text + "nom de colonne non valide";
               case 208: //nom de table incorrect dans select
                  LblErrors.Text = lblErrors.Text + "nom d'objet non valide";
               case 18452:
                  LblErrors.Text = lblErrors.Text + "nom de l'utilisateur non valide";
               case 18456:
                  LblErrors.Text = lblErrors.Text + "mot de passe non valide";
               case 4060:
                  LblErrors.Text = lblErrors.Text + "base de données non valide";
            } }
catch(Exception ex2){
            LblErrors.Text = lblErrors.Text + "Exception inattendue : " + ex2.Message + " , ";
          }
```

Gestion Erreur

L'exemple suivant montre comment parcourir la collection Errors pour rechercher les détails sur les erreurs SQLServer survenues :

```
SqlErrorCollection erData = ex1.Errors;
for(int i = 0;i<=erData.Count-1;i++){
        lbErrors.Text += "Erreur " + i + " : " +
            erData(i).Number + " , " +
            erData(i).Class & " , " +
            erData(i).Message;
    }
```


Exercice

- Quel est le résultat du code suivant :

```
DataRow r;
String str;
ForEach( r in ds.Tables[''Stagiaire''].Rows){
    Str +=r[1];
    Str +=r[''nom''];
}

DataTable dt =ds.Tables[''Stagiaire''];
CBmtr.ValueMember = "idstg " ;
CBmtr.DisplayMember = "nom " ;
CBmtr.DataSource = dt;
```

- Reproduire la Série M22 N1 en mode déconnecté.

Contrôle Utilisateur(Composant)

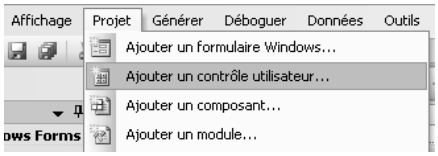
- En programmation, le terme composant est généralement utilisé pour désigner un objet qui est réutilisable et qui peut interagir avec d'autres objets.
- Un composant doit définir des propriétés à la place de champs publics, car les concepteurs visuels, tels que Visual Studio .NET, affichent des propriétés et non des champs dans la fenêtre de propriétés.
- Les propriétés sont comme des champs intelligents. Une propriété possède généralement une donnée membre privée accompagnée de fonctions d'accesseur et est accédée du point de vue syntaxique comme un champ d'une classe
- Voire Atelier UserControle (Horloge) sur la partie téléchargement du site :
- http://www.ista-ntic.net/ → Téléchargement

Atelier UserContrôle

- **1- étape lancer VS.net et créer un projet de type "application Windows "**

et l'enregistrer sous le nom MonPremierControle Le VS.net crée un formulaire qui va servir pour tester notre Horloge

- **2- ajouter un contrôle utilisateur vide**



Lui affecter le nom Horloge.cs

Question : quelle est la différence entre les deux classes Form1 et Horloge?

- **3- créer notre Horloge.**

On a besoin de deux contrôles minimum, oui c'est ça un label et un Timer. Ajoutons les alors. Après nous modifions les propriétés du Timer pour qu'il rafraîchir l'heure chaque seconde, donc il faut choisir comme intervalle 1000ms (millet seconde) et enabled à true.

est ce que c'est suffisant? Non, il faut l'activer pour que l'événement Tick se déclenche chaque intervalle (ici 1000).

Je rappelle que l'événement tick se déclenche chaque 1000 millet seconde. Donc, ajouter une méthode et l'associer à l'événement tick (vous double cliquez sur le contrôle Timer)

Ajouter le code suivant: Label1.text="il est : " + Date.now;

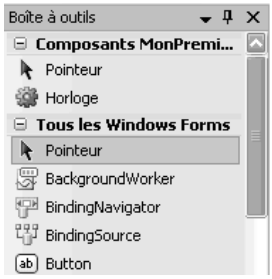
Atelier UserContrôle

- **4- tester le contrôle.**

Pour tester le contrôle, vous devez compiler le projet, et il va créer sur la boite d'outils votre contrôle Horloge que vous pouvez insérer ensuite dans le formulaire 'Form1'

Afficher le formulaire Form1 et regarder dans la palette des contrôles, est ce que vous voyez votre contrôle

Ajouter la dans votre formulaire et exécuter le (optionnel).



Intégrer les Transactions

- Les transactions sont des groupes d'opérations combinées en unités de travail logiques. Elles sont utilisées pour contrôler et conserver la cohérence et l'intégrité de chaque action d'une transaction, malgré les éventuelles erreurs rencontrées sur le système.
- Avec un SGBDR qui prend en charge la gestion des transactions vous pouvez créer des transactions explicites à l'aide d'instructions SQL BEGIN TRANSACTION, COMMIT TRANSACTION ou ROLLBACK TRANSACTION
- Pour effectuer une transaction :

1. Appelez la méthode BeginTransaction de l'objet SqlConnection pour marquer le début de la transaction. La méthode **BeginTransaction** retourne une référence à la transaction.
2. Assignez l'objet **Transaction** à la propriété Transaction du **SqlCommand** à exécuter. Si une commande est exécutée sur une connexion sur laquelle une transaction est active et si l'objet **Transaction** n'a pas été affecté à la propriété **Transaction** de l'objet **Command**, une exception est levée.
3. Exécutez les commandes requises.
4. Appelez la méthode Commit de l'objet SqlTransaction pour terminer la transaction, ou la méthode Rollback pour l'annuler. Si la connexion est fermée ou libérée avant que l'une des méthodes **Commit** ou **Rollback** ait été exécutée, la transaction est annulée.

Intégrer les Transactions (Exp)

```
SqlConnection connection= new SqlConnection(connectString);
connection.Open();
// Début d'une transaction local.
SqlTransaction sqlTran = connection.BeginTransaction();
// liaison du command avec la transaction courante.
SqlCommand command = connection.CreateCommand();
command.Transaction =sqlTran ;
try{
    command.CommandText="INSERT INTO Stagiaire(Nom) VALUES('Ali')";
    command.ExecuteNonQuery();
    command.CommandText="INSERT INTO Stagiaire(Nom) VALUES('Ahmed')";
    command.ExecuteNonQuery();
    sqlTran.Commit();
}
catch(Exception ee){
    sqlTran.Rollback();
    MessageBox.Show("Erreur :" + ee.ToString());
 }
```

NB: Pour une utilisation avancé des transaction, cherche l'utilisation des transaction distribué avec l'API dans le name space System.Transaction

Procédure Stocké (Exécution)

- Puisque l'utilisation des procédures stocké avec les SGBDR permet l'optimisation et la centralisation du code SQL au niveau du SGBD, il faut pas négligé leur utilisation dans une application client serveur (ADO.Net), les mêmes objets pour exécuter une instruction SQL sont réutilisé pour l'exécution d'une procédure stocké, en plus d'un objet Parameter pour définir explicitement les paramètres à une PS
- Exp:soit AffNote une procédure qui affiche la liste des notes d'un stagiaire donnée en paramètre

```
SqlConnection conn =new SqlConnection(@"data source=.\SQLExpress;initial catalog=GestStg; integrated
    security=true");
SqlCommand cmd = new SqlCommand("AffNote", conn);
cmd.CommandType=CommandType.StoredProcedure ;
// Création d'un paramètre en entrée
SqlParameter oParam= cmd.Parameters.Add("@idstg", SqlDbType.Int, 32);
oParam.Value="1";
conn.Open();
SqlDataReader dr = cmd.ExecuteReader();
while(dr.Read())
{   for (int i=0; i < dr.FieldCount; i++)
  {     richTextBox1.Text = richTextBox1.Text + dr[i].ToString() + "  " ;   }
  richTextBox1.Text = richTextBox1.Text + "\n« ;}
dr.Close();
conn.Close();
```

```
create proc AffNote
@idstg int
as
begin

select m.*, nt_note
from matiere m, note n
where
m.mt_id=n.mt_id and
st_id=@idstg

end
```

Procédure Stocké (Paramètre OUTPUT)

Pour une procédure avec paramètre de sortie (output) il faut spécifié la direction du paramètre avec ParametresDirection.Output, ici on a un exemple avec la procédure AffMoy (@idstg,@Moy OUTPUT) qui retourne la moyenne dans @moy pour le stagiaire avec @idstg

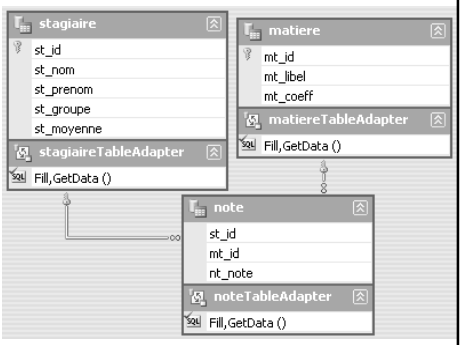
```
SqlConnection conn = new SqlConnection(@"data source=.\SQLExpress;initial catalog=GestStg0;
    integrated security=true");
SqlCommand cmd = new SqlCommand("Affmoy", conn);
cmd.CommandType = CommandType.StoredProcedure;
// Création d'un paramètre en entrée
SqlParameter oParam = cmd.Parameters.Add("@idstg", SqlDbType.Int);
oParam.Value = "1";
oParam = cmd.Parameters.Add("@Moy", SqlDbType.Float);
oParam.Direction = ParameterDirection.Output;
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();

richTextBox1.Text =
"Moyenne : " + cmd.Parameters["@Moy"].Value.ToString();
```

```
create proc affmoy
@idstg int, @moy
real output
as
begin
select @moy=(select
top 1 st_moyenne
from stagiaire where
st_id=@idstg)
end
```


les schémas XSD (DataSet typé)

Le DataSet est en fait un fichier XML qui définit les différentes tables qui composeront les sources de donnée de votre travail (ComboBox, DataGrid, CrystalReport…). Ce qui évite tout problème lié au changement de connexion a la base de donné.

- Pour ça en ajoute un nouveau élément dans le projet de type DataSet (ext xsd), après en ajoute des tables via une connexion dans l'explorateur de serveur, ou bien, en ajoute un nouveau DataAdapter avec une requête SQL spécifique et une chaîne de connexion, un DataTable s'ajoute automatiquement, avec un assistant qui permet de créer les autres objets du DataAdapter (InsertCommand, DeleteCommand, UpdateCommande)



États (CrystalReports)

Dans toute application en a besoin de rapport de sortie imprimable, soit des rapports de listing, de calcule ou de graphe.

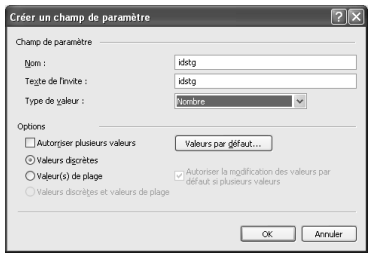
Pour ça en ajoute un nouveau élément de type CrystalReport (ext rpt), dans l'explorateur des champs en peut tous paramétré, champs de base de donnée, champs de formule, champs de paramètre, champs de nom de groupe, champs de total cumulé et autre.

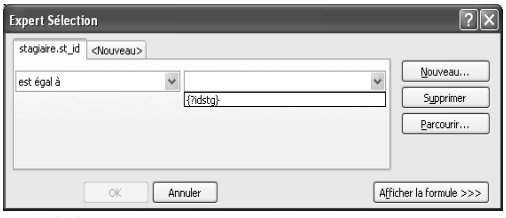
Exemple : pour avoir un état de tous les stagiaires avec leur notes et matière, en commence par paramétré le champs base de donnée avec le DataSet typé, en paramètre le champs de nom de groupe pour spécifié le regroupement par id_stg et en glisse les éléments dont en a besoin sur le CrystalReport, pour affiché ce rapport, il faut utilisé un CrystalReportViewer dans une Forms et ajouté le code suivant au démarrage de la Forms:

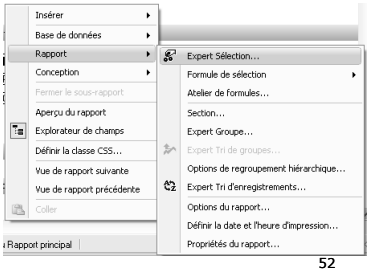
```
using GestionSTG.GestStgDSTableAdapters;

GestStgDS DS=new GestStgDS() ;  // instanciation du dataSet typé
StagiaireTableAdapter DAstg=new StagiaireTableAdapter() ;
MatiereTableAdapter DAmat=new MatiereTableAdapter() ;
NoteTableAdapter DAnote = new NoteTableAdapter() ;

STGListeNote etatn=new STGListeNote() ;  // remplisssage des Datatables dans le DataSet
DAstg.Fill(DS.Stagiaire);
DAmat.Fill(DS.Matiere);
DAnote.Fill(DS.Note);
etatn.SetDataSource(DS);  // lien entre CR et DataSet
crystalReportViewer1.ReportSource = etatn;   // lien entre CRViewer et CR
```


États (CrystalReports)

- Pour afficher un seule relevé pour un seule stagiaire, il faut travailler le champs de paramètre pour créer un paramètre. Donnez lui un nom et un type
- Click droit dans un emplacement vide dans le rapport pour affiche l'Expert Sélection, après le chois de l'élément cliquez sur OK pour précisé l'égalité entre l'élément sélectionné et le paramètre créer







États (CrystalReports)

- Et pour le programme, il faut faire les changement suivant pour prendre en considération le paramètre:

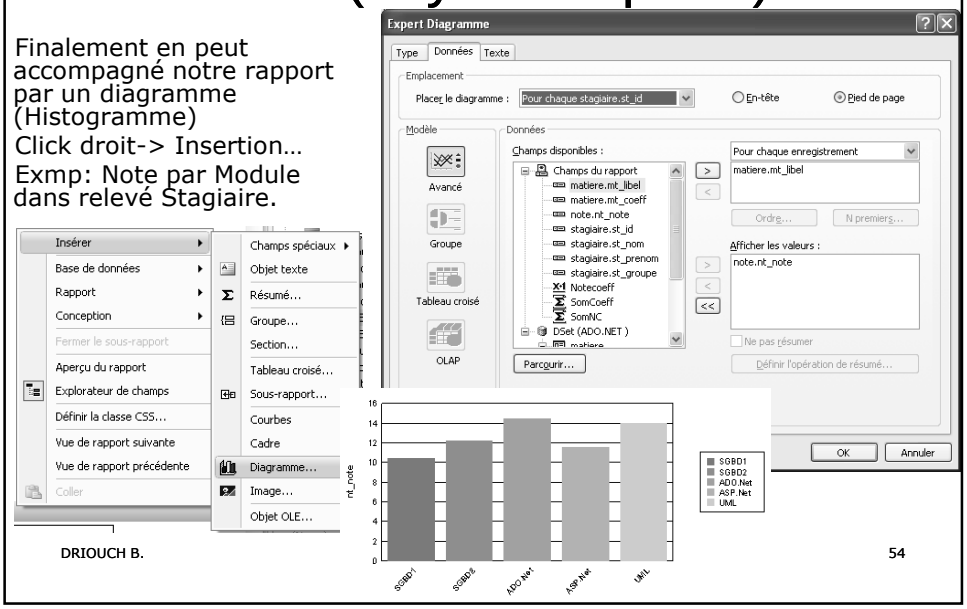
```
GestStgDS DS=new GestStgDS() ; // instanciation du dataSet typé
StagiaireTableAdapter DAstg=new StagiaireTableAdapter() ;
MatiereTableAdapter DAmat=new MatiereTableAdapter() ;
NoteTableAdapter DAnote = new NoteTableAdapter() ;

STGListeNote etatn=new STGListeNote() ;// remplisssage des Datatables dans le
    DataSet
DAstg.Fill(DS.Stagiaire);
DAmat.Fill(DS.Matiere);
DAnote.Fill(DS.Note);

etatn.SetDataSource(DS); // lien entre CR et DataSet
etatn.SetParameterValue("idstg", idstg); //idstg: valeur du paramètre

crystalReportViewer1.ReportSource = etatn;  // lien entre CRViewer et CR
```


États (CrystalReports)

Finalement en peut accompagné notre rapport par un diagramme (Histogramme)

Click droit-> Insertion…

Exmp: Note par Module dans relevé Stagiaire.



Déploiement (Install)

- Il y as plusieurs méthode de publication(publication normal, avec projet de déploiement, autres outils externes…)
- Dans la page de propriété de votre projet il y as une partie Publier qu'il faut configuré avant le lancement de votre publication
  - Emplacement du dossier de publication : dossier de sortie
  - Fichiers d'application: ajouter les fichier de ressource de votre application (base de donnée, image, Icon, ficher text, …)
  - Composants requis: composant utilisé dans votre application (rendre les outils utilisé comme CrystalReport, SQLExpress, …) installable à partir de votre application
  - Mise a Jour: l'application cherche la MAJ ou non(si oui indiqué le lien Web)
  - Options : autres information