

Secteur Tertiaire Informatique

Filière étude - développement

Développer des composants d'interface

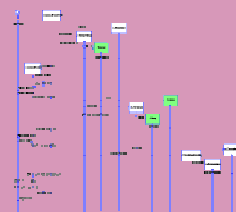
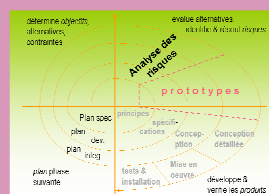
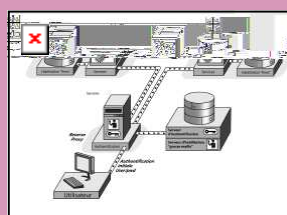
Accès aux données : Le mode déconnecté

Accueil

Apprentissage

Période en entreprise

Evaluation



Code barre

SOMMAIRE

I	LE MODE DÉCONNECTÉ : PRINCIPES	4
I.1	La classe DATASET	4
I.2	La classe DATAADAPTER	5
I.3	Création d'un DATASET	6
I.3.1	Par programmation	6
I.3.2	A l'aide de l'objet DATAADAPTER	7
I.3.3	Avec XML	8
I.4	L'objet DATAVIEW	9
I.5	Accès aux données	10
I.5.1	Lecture des données	10
I.5.2	Modification des données dans le DATASET	11
I.6	Traiter les données modifiées	12
I.6.1	Report des modifications dans la base	12
I.6.2	La méthode UPDATE	13
I.6.3	Gestion des accès concurrents	14
II	LES DATASET TYPES	15
II.1	Création d'un DATASET typé	15
II.2	La classe TABLEADAPTER	17
II.3	Accès aux données	20
III	UTILISER LES LIAISONS DE DONNEES	22
III.1	Liaison Simple	22
III.1.1	La liaison avec une boîte de liste	22
III.1.2	La liaison avec une zone d'édition	22
III.2	Graphiquement	23
III.3	Le composant BindingSource	28
IV	ANNEXES	29
IV.1	Les membres de la classe DATASET	29
IV.2	Les membres de la classe DATATABLE	30
IV.3	Les membres de la classe DATAADAPTER	32
IV.4	Les membres de la classe TABLEADAPTER	34

I LE MODE DECONNECTE : PRINCIPES

Au lieu de travailler avec un curseur de bases de données ouvert pendant tout le temps de l'application, l'idée est de pouvoir travailler en mémoire, en se connectant le minimum de temps au serveur.

Intérêt : Diminution du nombre de licences et démultiplication des capacités de montée en charge.

La structure de données principale, très proche d'une structure de base de données relationnelles, mais en mémoire, est représentée par la classe **DataSet**.

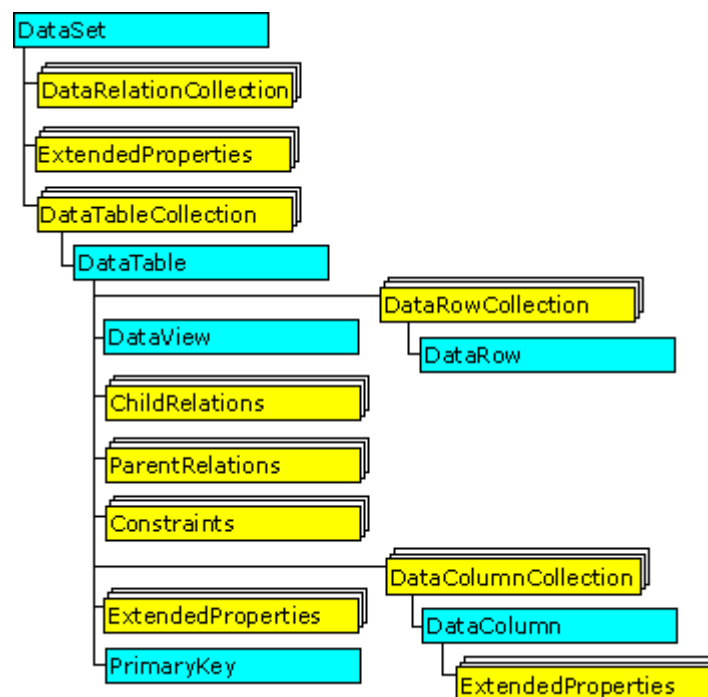
Le travail avec des objets déconnectés présente plusieurs avantages notamment dans des architectures multi-niveaux.

I.1 LA CLASSE DATASET

Le **DataSet** peut contenir plusieurs tables (classe **DataTable**), contenues dans une *DataTableCollection*, chaque table étant accessible par la propriété *Tables*, ainsi que les relations entre ces tables (classe **DataRelation**).

Chaque **DataTable** comporte une collection de colonnes (classe **DataColumn**) et une collection de lignes (classe **DataRow**).

Le modèle objet du DataSet

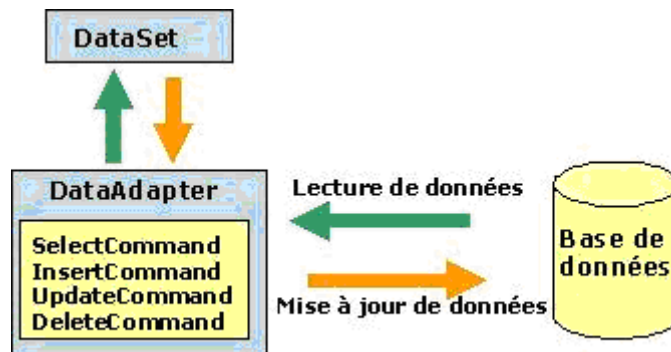


Les modifications apportées aux données du **DataSet** sont stockées en cache et reportées au moment de la connexion sur la base de données. On peut accéder aux lignes modifiées du **DataSet** pour déterminer le type de mise à jour à effectuer ainsi que comparer la version originale et la version actuelle de chaque ligne de données en cache.

I.2 LA CLASSE DATAADAPTER

Le chargement d'une **DataTable** du **DataSet** à partir de la base, ainsi que le report des modifications stockées sur la base, sont effectuées grâce à l'objet **DataAdapter**.

Schéma de fonctionnement de l'objet DataAdapter



- Le **DataAdapter** comporte une collection **TableMappings** :

Par défaut, le **DataAdapter** considère que les noms de colonnes de la table correspondent aux noms de colonnes du **DataSet**. Le cas échéant, si l'on souhaite que le nom des colonnes du **DataSet** diffère du nom des colonnes de la table, on emploiera un mécanisme d'alias au niveau de la requête.

Ainsi, la colonne *Codart* pourrait s'appeler *CodeProduit* dans le **DataSet** par le biais de l'exécution de l'instruction suivante en SQL :

```
"SELECT Codart AS CodeProduit, Numfou FROM Produit";
```

Le **DataAdapter** offre un mécanisme de mise en correspondance des noms de colonnes du **DataSet** et de la table via la collection **TableMappings**.

- Le **DataAdapter** comporte quatre objets **Command**, **SelectCommand**, **InsertCommand**, **UpdateCommand**, **DeleteCommand** dédiés au chargement et à la mise à jour de la table dans la base de données.

Le **DataAdapter** remplit la **Datatable** du **DataSet** au moyen de sa méthode **Fill** qui récupère des lignes de la source de données à l'aide de l'instruction **SELECT** spécifiée par la propriété **SelectCommand** associée.

La base de données sera mise à jour au moyen de la méthode **Update** du **DataSet** qui invoquera les instructions **Insert**, **Update** et **Delete** spécifiées par les propriétés **InsertCommand**, **UpdateCommand**, **DeleteCommand**.

I.3 CREATION D'UN DATASET

La création d'un objet **DataSet** peut se faire au travers de différentes méthodes qui peuvent être appliquées indépendamment les unes des autres ou combinées. Plusieurs possibilités existent :

I.3.1 Par programmation

Après avoir initialisé une nouvelle instance de **DataSet**, chaque objet **DataTable**, **DataRelation** et **Constraint** peut être créé par programmation et ajouté à l'objet **DataSet** ; Les tables de données sont ensuite peuplées, en ajoutant des lignes de données.

EXEMPLE : La base de données relationnelle CLUB est constituée des relations suivantes :

PAYS (PaysID, LibelleP)

REGION (RegionID, LibelleR, PaysId)

Déclaration

```
DataSet dsPaysRegion = new DataSet("PaysRegion");
```

La propriété **DataSetName** du **DataSet** a été positionnée.

Ajout de la DataTable PAYS, et création des colonnes

```
// Créer une nouvelle DataTable.
DataTable wTable = new DataTable("Pays");

// Créer la 1ere colonne ❶
DataColumn column = new DataColumn();
column.DataType = System.Type.GetType("System.String");
column.ColumnName = "PaysId";
column.AutoIncrement = true;
column.Caption = "PaysId";
column.ReadOnly = true;
column.Unique = true;
// Ajout de la colonne à la table
wTable.Columns.Add(column);

// Créer la 2eme colonne ❷
wTable.Columns.Add("LibelleP", typeof(String));

// Déclarer la clé primaire. ❸
DataColumn[] PKColumns = new DataColumn[1];
PKColumns[0] = wTable.Columns["PaysID"];
wTable.PrimaryKey = PKColumns;

// Ajout de la table au dataset
dsPaysRegion.Tables.Add(wTable); ❹
```

❶ Pour créer cette **DataColumn**, on utilise le constructeur

Chaque **DataColumn** possède une propriété **DataType** qui détermine le type de données ; Les propriétés, telles que **AllowDBNull**, **Unique** et **ReadOnly** imposent des restrictions à l'entrée et à la mise à jour des données, afin de préserver leur intégrité.

❷ Pour créer cette colonne, on appelle la méthode **Add** de la propriété **Columns** de la table.

❸ La déclaration de clé primaire se fait en déclarant un tableau de un élément, car la table ne possède qu'une colonne définissant la clé primaire ; on aurait pu coder :

```
wTable.PrimaryKey = new DataColumn[] {wTable.Columns["PaysId"]};
```

❹ Ajout de la table créée au DataSet

Pour ajouter des lignes à **DataTable**, on utilisera la méthode **NewRow** pour retourner un nouvel objet **DataRow**. La méthode **NewRow** retourne une ligne avec le schéma de **DataTable**, tel qu'il est défini par le **DataColumnCollection** de la table.

```
DataRow row;
row = wtable.NewRow();
row["PaysId"] = "FR";
row["LibelleP"] = "France";
wtable.Rows.Add(row);
```

De la même manière, on pourrait créer d'autres **DataTable**, que l'on ajouterait au **DataSet**.

Création de la clé étrangère

```
// La DataRelation nécessite deux DataColumn, une colonne parent,
// une colonne enfant et un nom
DataColumn parentColumn =
dsPaysRegion.Tables["Pays"].Columns["PaysId"];
DataColumn childColumn =
dsPaysRegion.Tables["Regions"].Columns["PaysId"];
// Création de la datarelation et ajout au dataSet
DataRelation relation = new DataRelation("FK_Regions_Pays",
parentColumn, childColumn);
dsPaysRegion.Tables["Regions"].ParentRelations.Add(relation);
```

DataTable contient aussi une collection d'objets **Constraint** qui peuvent être utilisés pour garantir l'intégrité des données.

I.3.2 A l'aide de l'objet DATAADAPTER

L'objet **DataAdapter** va permettre de remplir l'objet **DataSet** de tables de données provenant d'une source de données relationnelles existante.

```
// accès à la base
string connectionString = "Data Source=SRVSQL;Initial
Catalog=BDTest;Integrated Security=True";
sqlConnect = new SqlConnection(connectionString);

// Création du dataSet.
DataSet dsPaysRegion = new DataSet("paysRegions");

string strSql = "SELECT PaysID, LibelleP FROM dbo.Pays";
SqlDataAdapter paysAdapter = new SqlDataAdapter(strSql,
sqlConnect);
```

Reste à remplir la **DataTable** Pays du **DataSet** dsPaysRegions par la méthode **Fill** du **DataAdapter**.

```
paysAdapter.Fill(dsPaysRegions, "Pays");
```

Lorsque le **DataAdapter** remplit le **DataSet**, il crée les tables et les colonnes nécessaires pour les données retournées. Cependant, les informations de clés primaires ne sont pas incluses dans le schéma créé implicitement, à moins que la propriété **MissingSchemaAction** ait la valeur **MissingSchemaAction.AddWithKey**.

Ce code n'ouvre pas et ne ferme pas la **Connection** de manière explicite. Si la connexion est fermée avant l'appel à **Fill**, elle est ouverte pour extraire les données, puis automatiquement fermée. Si la connexion est ouverte avant l'appel à **Fill**, elle reste ouverte.

L'objet **DataTable** représente une table de données en mémoire. Utilisé comme conteneur de données dans un **DataSet**, il est également valide comme objet autonome et peut se remplir de la même manière.

```
DataTable dt = new DataTable("Pays");  
string strSql = "SELECT PaysID, LibelleP FROM dbo.Pays";  
SqlDataAdapter paysAdapter = new SqlDataAdapter(strSql, sqlConnect);  
paysAdapter.Fill(dt);
```

I.3.3 Avec XML

On peut charger et rendre persistant le contenu d'un objet **DataSet** à l'aide de **XML**.

La méthode **ReadXml** fournit un moyen de lire soit les données uniquement, soit les données et le schéma, dans un **DataSet** à partir d'un document XML, alors que la méthode **ReadXmlSchema** lit uniquement le schéma. Pour lire à la fois les données et le schéma, utilisez l'une des surcharges **ReadXML** qui incluent le paramètre **mode** et affectez-lui la valeur **ReadSchema**.

```
// Création du dataSet.  
DataSet dsPaysRegions= new DataSet("paysRegions");  
// Remplir le dataSet  
string filePath = "C:\\Pays.XML"; // Chemin complet  
paysRegions.ReadXml(filePath);
```

Cette remarque s'applique également aux méthodes **WriteXml** et **WriteXmlSchema**, respectivement. Pour écrire les données XML, ou à la fois le schéma et les données à partir du **DataSet**, utilisez la méthode **WriteXml**. Pour écrire uniquement le schéma, utilisez la méthode **WriteXmlSchema**.

I.4 L'OBJET DATAVIEW

La classe **DataRowView** représente une vue personnalisée d'un **DataTable**. A tout moment, plusieurs vues différentes des données sont disponibles.

Par défaut, la vue de la table n'est pas filtrée et contient toutes les lignes de la table. Les propriétés **RowFilter** et **RowStateFilter** ou **Sort**, permettent de restreindre l'ensemble des lignes dans une vue particulière, et/ou de les trier.

Il existe deux façons de créer un objet **DataRowView**.

- en utilisant le constructeur de **DataRowView**
Le constructeur de **DataRowView** peut être vide, mais il peut également accepter soit un **DataTable** comme argument unique, soit un **DataTable** avec des critères de filtre, de tri et un filtre d'état de ligne.

Exemple 1 :

```
// Création du dataSet
DataSet dsPaysRegion = new DataSet("paysRegions");

// Création de la vue.
DataRowView viewR = new DataRowView();
viewR.Table = dsPaysRegion.Tables["Region"];
viewR.AllowDelete = true;
viewR.AllowEdit = true;
viewR.AllowNew = true;
viewR.RowFilter = "PaysId = 'FR'";
viewR.RowStateFilter = DataRowViewRowState.ModifiedCurrent;
view.Sort = "LibelleR";
```

La vue est créée sur la table Région du **DataSet** dsPaysRegion : elle permettra les suppressions, les modifications, les insertions ; seules les lignes en version actuelle, contenant un code Pays égal à « FR » seront sélectionnées et apparaîtront triées par libellé région.

Ce qui équivaut à :

```
DataRowView viewR = new DataRowView(dsPaysRegion.Tables["Region"],
"PaysId = 'FR'", "LibelleR", DataRowViewRowState.ModifiedCurrent);
viewR.AllowDelete = true;
viewR.AllowEdit = true;
viewR.AllowNew = true;
```

- en créant une référence à la propriété **DefaultView** de l'objet **DataTable**

```
DataRowView viewR = dsPaysRegion.Tables["Region"].DefaultView;
```

A la différence du système de vues fourni par les différents SGBD, la classe **DataRowView** ne permet pas de créer des vues sur une jointure de tables. Elle ne permet pas non plus d'exclure des colonnes.

I.5 ACCES AUX DONNEES

I.5.1 Lecture des données

Le **DataAdapter** représente un ensemble de commandes couplées avec une connexion utilisée pour remplir le **DataSet** et reporter les modifications dans la base

Les données sont traitées alors que l'application n'est plus connectée à la base : Le **DataSet** fait office de cache déconnecté de données.

La propriété **Tables** du **DataSet** donne accès à chacune des tables du dataset représenté par un objet **DataTable** ; les propriétés remarquables de l'objet **DataTable** sont :

- La propriété **Columns** qui donne accès aux différentes colonnes de la table à travers l'objet **DataColumn**.
- La propriété **Rows** qui donne accès aux différentes lignes de données à travers l'objet **DataRow**.

Exemples:

```
// accès à la 1ere table de la collection
dsPaysRegion.Tables[0];
dsPaysRegion.Tables["Pays"]

// accès à la 2eme colonne de la table Pays
dsPaysRegion.Tables[0].Columns[1]
dsPaysRegion.Tables["Pays"].Columns[1]
dsPaysRegion.Tables["Pays"].Columns["LibelleP"]

// accès au nombre de colonnes
int nc = dsPaysRegion.Tables[0].Columns.Count;

// accès au nombre de lignes
int nl = dsPaysRegion.Tables["Pays"].Rows.Count;

// accès à la 3eme ligne de la table
dsPaysRegion.Tables["Pays"].Rows[2];

// accès à la 2eme colonne de la 3eme ligne de la table
dsPaysRegion.Tables["Pays"].Rows[2][1]; // est de type object
string s = dsPaysRegion.Tables["Pays"].Rows[2][1].ToString();
```

Exemple : Après avoir créé le **DataSet** avec l'objet **DataAdapter**, on souhaite charger en liste le contenu de la colonne **LibelleP** de la **DataTable Pays**.

```
// remplissage de la liste
DataTable dt = dsPaysRegion.Tables[0];
foreach (DataRow dr in dt.Rows)
{
    lstPays.Items.Add(dr["LibelleP"]);
}
```

I.5.2 Modification des données dans le DATASET

Les objets **DataRow** et **DataColumn** sont les principaux composants de **DataTable**. On pourra utiliser l'objet **DataRow** ainsi que ses propriétés et méthodes pour récupérer, évaluer, insérer, supprimer et mettre à jour les valeurs de **DataTable**.

- Pour créer un nouveau **DataRow**, on utilisera la méthode **NewRow** de l'objet **DataTable** et on ajoutera ce nouveau **DataRow** à **DataRowCollection** en utilisant la méthode **Add**.
- On supprimera un **DataRow** en appelant la méthode **Remove** de **DataRowCollection** ou la méthode **Delete** de l'objet **DataRow**. La méthode **Remove** supprime la ligne de la collection. Par contre, **Delete** marque **DataRow** en vue de sa suppression. La suppression se produit effectivement lorsque vous appelez la méthode **AcceptChanges**.

Pour modifier le nom du Pays dans la troisième ligne, on écrira :

```
dsPaysRegion.Tables["Pays"].Rows[2][1] = "Libellé modifié";
```

Pour ajouter un nouveau Pays dans le DataSet, on écrira :

```
DataRow dr = dsPaysRegion.Tables[0].NewRow();  
dr["PaysId"] = "SU";  
dr["LibelleP"] = "Suede";  
dsPaysRegion.Tables[0].Rows.Add(dr);
```

Pour supprimer la troisième ligne de la table Pays dans le DataSet, on écrira :

```
dsPaysRegion.Tables[0].Rows[2].Delete();
```

En interne, une **DataTable** maintient les données courantes et les données originales.

La valeur courante est la valeur accessible en lecture, la valeur originale est la dernière valeur validée de la cellule : Lorsqu'une ligne a été créée, la valeur affectée est la valeur courante, la valeur originale est null.

Le développeur n'a accès qu'aux données courantes.

Chaque ligne d'une **DataTable** présente la propriété **Rowstate** qui indique si ces données ont subi des changements.

L'état actuel de la ligne est obtenu par l'énumération **DataRowstate** :

Nom de membre	Description
Added	La ligne a été ajoutée à DataRowCollection et AcceptChanges n'a pas été appelé.
Deleted	La ligne a été supprimée à l'aide de la méthode Delete de DataRow .
Detached	La ligne a été créée, mais n'appartient à aucun DataRowCollection . DataRow est dans cet état immédiatement après sa création et avant son ajout à une collection, ou s'il a été supprimé d'une collection.
Modified	La ligne a été modifiée et AcceptChanges n'a pas été appelé.

Unchanged

La ligne n'a pas été modifiée depuis le dernier appel à **AcceptChanges**.

La méthode **BeginEdit** positionne un **DataRow** en mode édition. Dans ce mode, les événements sont temporairement suspendus, ce qui permet à l'utilisateur d'apporter différentes modifications à plusieurs lignes sans déclencher de règles de validation. Par exemple, pour garantir que la valeur de la colonne de montant total est égale aux valeurs des colonnes de débit et de crédit sur une ligne, vous pouvez mettre chaque ligne en mode édition pour suspendre la validation des valeurs des lignes jusqu'à ce que l'utilisateur essaye de les valider.

En mode édition, **DataRow** stocke des représentations de la valeur d'origine et de la nouvelle valeur proposée.

Par conséquent, tant que vous n'appellez pas la méthode **EndEdit**, vous pouvez récupérer la version d'origine ou la version proposée en passant **DataRowVersion.Original** ou **DataRowVersion.Proposed** comme paramètre version de la propriété **Item**.

Vous pouvez également annuler des modifications à ce stade en appelant la méthode **CancelEdit**.

I.6 TRAITER LES DONNEES MODIFIEES

I.6.1 Report des modifications dans la base

Pour mettre à jour la base grâce à la méthode **Update** du **DataAdapter**, il faut au préalable configurer ses propriétés

- DeleteCommand (suppression dans la base)
- InsertCommand (insertion dans la base)
- UpdateCommand (modification dans la base)

Chaque commande contient des paramètres qui spécifient la ligne en cours de modification.

```
// configuration du data adapter
// DeleteCommand
strSql = "DELETE FROM Pays WHERE PaysId = @pCode";
paysAdapter.DeleteCommand = new SqlCommand(strSql, sqlConnect);
paysAdapter.DeleteCommand.Parameters.Add( "@pCode", SqlDbType.Char );

// UpdateCommand
strSql = "Update Pays set LibelleP = @pNom WHERE PaysId = @pcode";
paysAdapter.UpdateCommand = new SqlCommand(strSql, sqlConnect);
paysAdapter.UpdateCommand.Parameters.Add( "@pCode", SqlDbType.Char );
paysAdapter.UpdateCommand.Parameters.Add( "@pNom", SqlDbType.Char );
```

avant l'appel de la méthode **Update**, il faut alors donner la valeur des paramètres :

```
paysAdapter.UpdateCommand.Parameters[ "@pCode" ].Value = txtCodeP.Text;
paysAdapter.UpdateCommand.Parameters[ "@pNom" ].Value = txtLibP.Text;

// Update
paysAdapter.Update(dsPaysRegion, "Pays" );
```

I.6.2 La méthode UPDATE

Pour envoyer les données modifiées à une base de données, on appellera la méthode `Update` d'un **TableAdapter** (ou **DataAdapter**), qui met à jour la table de données et exécute la commande correcte (INSERT, UPDATE ou DELETE) en fonction du **RowState** de chaque ligne de données contenue dans la table.

La méthode **HasChanges** d'un groupe de données retourne la valeur **true** si des modifications ont été apportées au groupe de données.

Après avoir déterminé qu'il existe des lignes modifiées, on peut appeler la méthode **GetChanges** d'un **DataSet** ou d'un **DataTable** pour retourner un jeu de lignes modifiées. Par la méthode **GetChanges**, on obtient une copie du **DataSet** contenant l'ensemble des modifications qui lui ont été apportées depuis son dernier chargement ou depuis l'appel à **AcceptChanges**, pouvant être filtrée par **DataRowState**.

Il peut arriver qu'un enregistrement contienne une erreur : La propriété **HasErrors** du **DataSet**, de la **DataTable**, d'une **DataRow** peut être appelée pour déterminer si la ligne est en erreur. La méthode **GetErrors** permet de récupérer un tableau des **DataRow** en erreur.

Au cours de la validation d'une ligne par la méthode **AcceptChanges()**, la valeur courante est dupliquée comme valeur originale, et l'état général de la ligne devient *unchanged*.

Si l'on appelle **AcceptChanges()**, les lignes qui étaient dans l'état *Deleted* sont effectivement détruites.

Si l'on appelle **RejectChanges()**, les lignes qui étaient dans l'état *Added* sont détruites, et les données modifiées sont remises à leur état initial.

Exemple : Traitement des lignes modifiées

```
private void TraiterModif()
{
    if (dataSet1.HasChanges(DataRowState.Modified))
    {
        // Création d'un DataSet temporaire.
        DataSet xDataSet;
        // Appel de GetChanges uniquement pour les lignes
        // modifiées.
        xDataSet = dataSet1.GetChanges(DataRowState.Modified);
        // Vérification si erreurs dans le DataSet.
        if (xDataSet.HasErrors)
        {
            // Traitement en cas d'erreur
            MessageBox.Show("Erreur de mise à jour");
        }
        // Après traitement des erreurs, mise à jour de la source
        // de données
        sqlDataAdapter1.Update(xDataSet, "toto");
        // Prise en compte des changements dans le DataSet
        dataSet1.AcceptChanges();
    }
}
```

I.6.3 Gestion des accès concurrents

Lorsque plusieurs utilisateurs travaillent simultanément en mode déconnecté avec les mêmes bases de données, il y a possibilité de conflit lors de la modification de données.

L'utilisateur A lit un adhérent à 12h ; Supposons que l'utilisateur B modifie ce même adhérent à 12h05 ; Lorsque l'utilisateur A envoie la modification de ce même adhérent à 12h10, il se produira un conflit de mise à jour.

ADO.Net gère ces conflits de manière **optimiste** : rien n'est mis en œuvre pour les prévenir, mais si il survient, il est détecté, et la mise à jour n'est pas effectuée.

Dans notre exemple, l'adhérent conserve les valeurs des colonnes mises à jour à 12h05, et l'utilisateur A est averti du conflit.

Lorsque le conflit est détecté, une erreur de type **System.Data.DbConcurrencyException** est levée lors de l'appel de la méthode Update sur l'adapteur.

Théoriquement, il est possible de prévenir le conflit (gestion **pessimiste**) en utilisant des systèmes de verrous sur les lignes afin de synchroniser les accès. Ce système de verrous implique des temps d'attente , et donc une baisse significative des performances.

Il est conseillé aux développeurs ADO.NET de gérer les données de manière optimiste.

II LES DATASET TYPES

En réalité, la classe **DataSet** est rarement utilisée : On préfère utiliser des **DataSet** typés.

Un **DataSet** typé dérive de la classe **DataSet** mais implémente des propriétés et des méthodes spécifiques à la structure des données qu'il contient. Il contient notamment le type des champs de ces données. Cela permet de contrôler plus efficacement les données et d'éviter des erreurs lors de la mise à jour de la source de données. Le schéma des données est contenu dans un fichier .xsd lié au **DataSet**.

Dans le cas d'un **DataSet** typé, le programmeur profite de l'**Intellisense**, et l'écriture du code est simplifiée. Ainsi, le code suivant s'appliquant à un **DataSet** classique.

```
DataRow dr = ds.Tables["Table1"].Rows[0];  
string s = (string)dr["Champ1"].ToString();
```

devient, si ds est un DataSet typé

```
DS.TABLE1Row dr = ds.TABLE1.FindByyyy(xxx)  
string s = dr.Champ1 ;
```

ou TABLE1Row est une DataRow de TABLE1, et ou on adresse directement TABLE1 sans passer par la collection Tables.

II.1 CREATION D'UN DATASET TYPE

Pour créer un DataSet typé, il est possible :

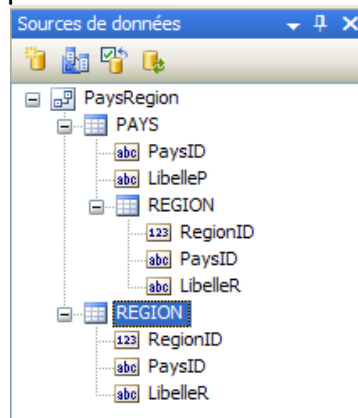
- soit d'utiliser l'Assistant **Configuration de source de données** fourni dans Visual Studio :

A partir du menu **Données**, écrans après écrans, après avoir sélectionné, **Ajouter une nouvelle source de données**, puis en choisissant **Base de données, Nouvelle connexion**, la sélection des objets de bases de données constituant le **DataSet** sera possible :

L'assistant, après avoir demandé si les commandes sont supportées par des requêtes ou des procédures stockées, proposera une chaîne de requête correspondant à la requête Select de chaque table et pourra créer les 2 méthodes du TableAdapter qui exécuteront cette chaîne de requête (Fill et GetData).

Il permettra également de créer les chaînes de requêtes correspondant aux modifications des données dans la base (Insert, Update, Delete).

Le **DataSet** ainsi constitué apparaîtra alors dans la fenêtre Sources de données.



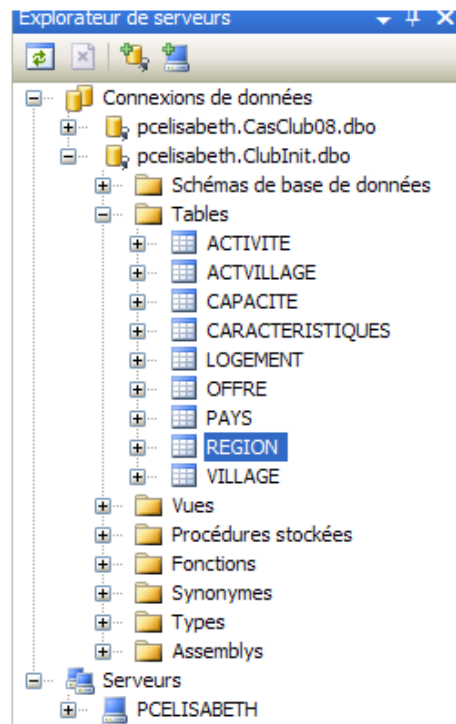
Accès aux données

- soit dans l'explorateur de solutions, **Click droit sur le projet, Ajouter / Nouvel élément / DataSet**

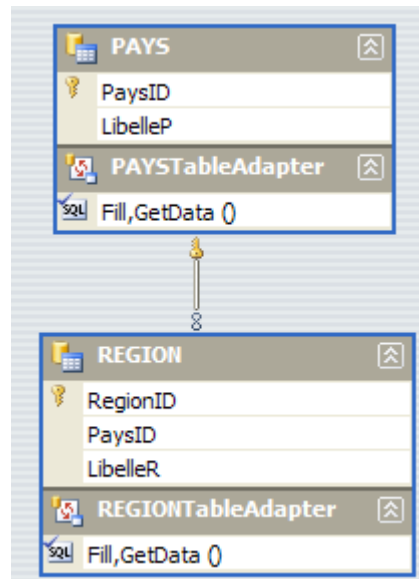
Il suffit alors de sélectionner une base de données et les tables à prendre en compte dans le **DataSet** typé au moyen de la fenêtre **Explorateur de serveurs**.

L'**explorateur de serveurs** permet d'accéder et de manipuler des ressources issues de tout serveur pour lequel l'accès est autorisé. Il permet, entre autres, de se connecter à des serveurs de bases de données via ODBC, telles que SQL Server ou Oracle, de visualiser et manipuler les objets des bases définies sur ce serveur.

Depuis ce composant, il est possible de créer des bases de données, des tables, des requêtes, des procédures stockées ou des déclencheurs, et de glisser des objets provenant de ces bases sur des formulaires.



Dans les deux cas, le projet contient un nouveau fichier d'extension **.xsd**, et un fichier de code source associé qui contient le code généré de la classe **DataSet** et de ses classes encapsulées.



Le code généré par le **Concepteur de DataSet** est disponible dans le fichier *PaysRegion.Designer.cs*, et peut être régénéré

Pour empêcher la suppression de votre code pendant la régénération, ajoutez du code au fichier de classe partielle (click droit sur *PaysRegion.xsd* / Afficher le code).

```
namespace Deconnecte {

    partial class PaysRegion
    {
        partial class REGIONDataTable
        {
        }
    }
}
```

La technique des classes partielles permet de séparer le code généré et le code développé dans deux fichiers distincts.

II.2 LA CLASSE TABLEADAPTER

Comme le **DataAdapter**, le **TableAdapter** comporte quatre objets **Command**, **SelectCommand**, **InsertCommand**, **UpdateCommand**, **DeleteCommand** dédiés au chargement et à la mise à jour de la table dans la base de données, pouvant être modifiées dans la fenêtre des propriétés.

La classe **TableAdapter** ne fait pas partie du .NET Framework, et ne se trouve pas dans la documentation ou dans l'**Explorateur d'objets**. Les **TableAdapters** sont **uniquement créés** à l'aide du **Concepteur de DataSet** au sein de groupes de données fortement typés, et remplacent les **DataAdapters** de la version antérieure.

On peut créer un tableAdapter dans un formulaire :

Après avoir créé un **DataSet** typé, un **TableAdapter** pourra être créé :

- En faisant glisser les nœuds (tables) de la fenêtre **Sources de données** vers un formulaire dans votre application Windows pour créer automatiquement une instance d'un **TableAdapter** sur le formulaire

- ou -

- Après avoir créé un **TableAdapter**, générez le projet. Le **TableAdapter** apparaît ensuite dans la **Boîte à outils**. Faites glisser le **TableAdapter** de la **Boîte à outils** vers un formulaire pour créer une instance.

- ou -

- par programme :

Les **TableAdapters** se trouvent à l'intérieur d'un espace de noms, au sein du projet, qui est identifié en fonction du nom du groupe de données associé au **TableAdapter**.

La convention d'affectation de noms est : *DataSetName* + "TableAdapters" : il faut donc ajouter

```
using Deconnecte.PaysRegionTableAdapters;  
// Deconnecte est le nom du projet,  
// PaysRegion est le nom de la classe DataSet
```

Pour utiliser le DataSet :

```
// Déclaration du DataSet  
PaysRegion dsPaysRegion = new PaysRegion ();  
// Déclaration des TableAdapter  
PAYSTableAdapter paysAdapt = new PAYSTableAdapter();  
REGIONTableAdapter regionAdapt = new REGIONTableAdapter();
```

La méthode **Fill** du **TableAdapter** récupère les lignes de la source de données à l'aide de l'instruction SELECT spécifiée par la propriété **SelectCommand** associée

```
// Chargement de la DataTable Region  
regionAdapt.Fill(dsPaysR.REGION);
```

On notera ici l'emploi de la propriété **REGION** permettant d'accéder directement à la DataTable.

La méthode **GetData** du **TableAdapter** retourne dans l'application une table de données typées remplie des résultats de la requête.

```
dsPaysRegion.PAYS uneTablePays;  
uneTablePays = paysAdapt.GetData();
```

Outre les fonctionnalités standard d'un **DataAdapter**, les **TableAdapters** peuvent contenir **autant de requêtes que l'exige l'application pour remplir les tables de données** qui leur sont associées pour permettre de charger des données qui répondent à différents critères.

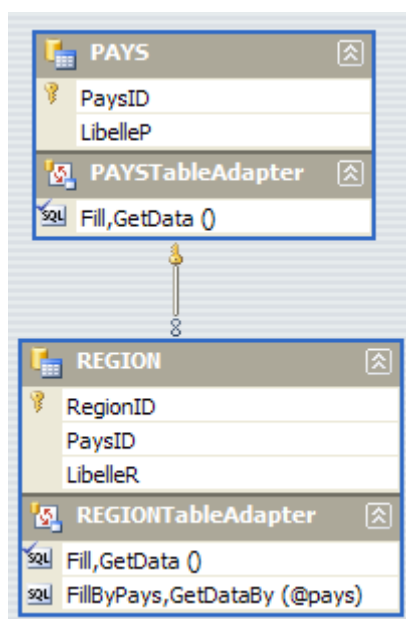
Avec un **TableAdapter**, on va pouvoir créer autant de commandes **FillBy...** que l'application le demande en créant de nouvelles requêtes.

Exemple : Pour remplir la table Régions avec les pays Français, la requête **FillByPays** prend en paramètre le code pays

```
SELECT * FROM regions WHERE PaysId = @Pays.
```

On créera une nouvelle méthode **FillBy...** à l'aide de l'**Assistant Configuration de TableAdapter** disponible dans le **Concepteur de DataSet**.

(click droit sur le nom du **TableAdapter** dans le Concepteur de DataSet).



Chargement de la **DataTable** Region du **DataSet** dsPaysRegion par la méthode **FillBypays** du **TableAdapter**.

```
regionAdapt.FillByPays(dsPaysRegion.REGION, "FR".);
```

La requête **GetDataBy** retournera de la même manière, une table de données typées remplie des résultats de la requête.

Il est également possible d'ajouter des requêtes qui retournent des valeurs scalaires (uniques), par exemple, la création d'une requête qui retourne un nombre de régions

```
SELECT Count(*) From Region
```

est valide pour un **REGIONTableAdapter**, même si les données retournées ne respectent pas le schéma de la table.

Le **TableAdapter** ajoute une propriété non disponible à la classe **DataAdapter** de base. Par défaut, chaque fois que vous exécutez une requête pour remplir la table de données d'un **TableAdapter**, les données sont effacées et seuls les résultats de la requête sont chargés dans la table. En affectant à la propriété **ClearBeforeFill** du **TableAdapter** la valeur **false**, vous ajouterez ou fusionnerez les données retournées par une requête aux données déjà existantes.

Le choix d'utiliser des **TableAdapters** ou des **DataAdapters** dépend de la manière dont a été créé le groupe de données.

Si l'**Assistant Configuration de source de données** a été utilisé, le groupe de données contient des **TableAdapters**.

On utilisera toujours les propriétés et méthodes typées du DataSet.

II.3 ACCES AUX DONNEES

La classe DataSet typé PaysRegion contient :

- Des classes encapsulées :
 - les classes typées DataRow, **PaysRow** et **RegionRow** qui possèdent une propriété correctement typée pour chaque colonne de la table, et des méthodes permettant de naviguer dans la relation liant les 2 tables.
 - Les classes typées DataTable, **PAYSDataTable** et **REGIONDataTable** qui permettent à leurs instances de faire des ajouts typés (AddPAYSRows) ou des recherches typées (FindByPaysId).
- Des propriétés : les propriétés **PAYS** et **REGION** permettent d'accéder directement aux tables et permettant de faire des ajouts typés (AddPAYSRows) ou des recherches typées (FindBy....).

```
// accès à la 1ere table de la collection
dsPaysRegion.PAYS

// accès à la 2eme colonne de la table Pays
dsPaysRegion.LIBELLEPColumn

// accès à la 3eme ligne de la table
dsPaysRegion.PAYS.Rows[2];

// accès à la 2eme colonne de ligne repéré par le code AU
// est de type PAYSRow
PayRegion.PAYSRow pr = dsPapyrus.PAYS.FindByPAYSID("AU")
string s = pr.LIBELLEP.ToString();
```

Exemple : Après avoir créé le DataSet avec l'objet TableAdapter, on souhaite charger en liste le contenu de la colonne LibelleP de la DataTable Pays.

```
// remplissage de la liste
PaysRegion.PAYSDataTable pdt = dsPaysRegion.PAYS;
foreach (PaysRegion.PAYSRow pr in pdt.Rows)
{
    lstPays.Items.Add(pr.LIBELLEP);
}
```

Pour ajouter une ligne, on utilisera la méthode générée :

```
pdt.AddPAYSRow(pr)
```

Pour modifier les données d'une ligne, on accédera directement aux propriétés représentant les colonnes de la table.

```
pr.LIBELLEP = txtLib.Text;
```

Les contrôles sur la ligne modifiée ou ajoutée peuvent être gérés avant l'affectation à la colonne concernée, et également au moment de l'affectation par le biais de certains événements de la DataTable.

Événement	Description
ColumnChanged	Se produit quand une valeur a été insérée dans une colonne avec succès.
ColumnChanging	Se produit quand une valeur a été proposée pour une colonne.
RowChanged	Se produit après qu'une ligne de la table a été modifiée avec succès.
RowChanging	Se produit quand une ligne de la table est en cours de modification.
RowDeleted	Se produit après qu'une ligne de la table a été marquée comme Deleted .
RowDeleting	Se produit avant qu'une ligne de la table soit marquée comme Deleted .

```
pdt.PAYSRowChanging // Niveau ligne
pdt.PAYSRowDeleting // Niveau ligne
pdt.ColumnChanging // Niveau colonne
```

associés par un délégué à une méthode dont la signature est générée automatiquement par l'IDE Visual Studio.

```
pdt.PAYSRowChanging +=new
PaysRegion.PAYSRowChangeEventHandler(pdt_PAYSRowChanging);
pdt.ColumnChanging +=new DataColumnChangeEventHandler(pdt_ColumnChanging);
void pdt_PAYSRowChanging(object sender, PaysRegion.PAYSRowChangeEvent e)
{
    // en cas de modification ou ajout
    if (e.Action == DataRowAction.Change | e.Action == DataRowAction.Add)
    {
        // le code doit être saisi
        e.Row.RowError = "Le libelle doit être saisi";
    }
}
```

Le programme peut alors marquer la ligne en erreur grâce à la propriété **RowError**.

Ces erreurs seront détectées par la propriété **HasError** du **DataSet**, de la **DataTable**, d'une **DataRow**, et pourront être récupérées dans un tableau par la méthode **GetErrors**

Se reporter au paragraphe I 6 2 pour l'utilisation de la méthode UPDATE.

III UTILISER LES LIAISONS DE DONNEES

Dans Windows Forms, il est possible de lier des contrôles non seulement à des sources de données traditionnelles mais également à toute structure qui contient des données. Vous pouvez créer une liaison avec un tableau des valeurs calculées au moment de l'exécution, lues dans un fichier ou dérivées des valeurs d'autres contrôles.

III.1 LIAISON SIMPLE

III.1.1 La liaison avec une boîte de liste

Plusieurs composants visuels présentent une propriété **DataSource**, permettant d'associer directement une source de données au composant (ListBox par exemple). Pour afficher dans une boîte de liste le champ LibelleP de la table Pays chargée dans le DataSet dsPaysRegion, et après avoir peuplé le DataSet, on codera :

```
lstPays.DataSource = dsPaysRegion.PAYS;  
lstPays.DisplayMember = "LIBELLEP";
```

III.1.2 La liaison avec une zone d'édition

Pour effectuer une liaison entre les champs du DataSet et les composants visuels, on utilisera la propriété DataBindings du composant visuel en ajoutant les liaisons à effectuer ;

```
txtCode.DataBindings.Add( "Text", dsPaysRegion.PAYS, "PaysId" );  
txtCode.DataBindings.Add( "Text", dsPaysRegion.PAYS, "LibelleP" );
```

La méthode Add appliquée à la propriété DataBindings du composant visuel prend trois arguments :

- La propriété du composant visuel à initialiser avec le champ de la donnée (propriété Text)
- La table du DataSet
- Le nom du champ dans cette table.

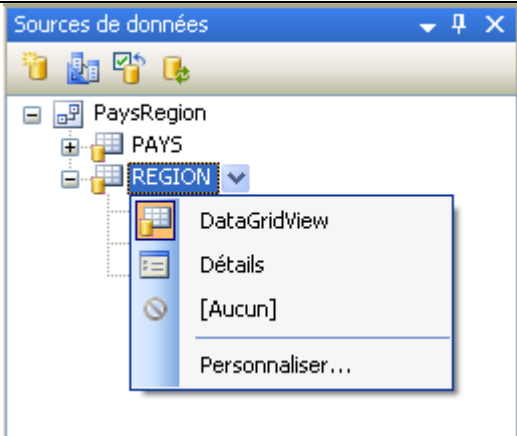
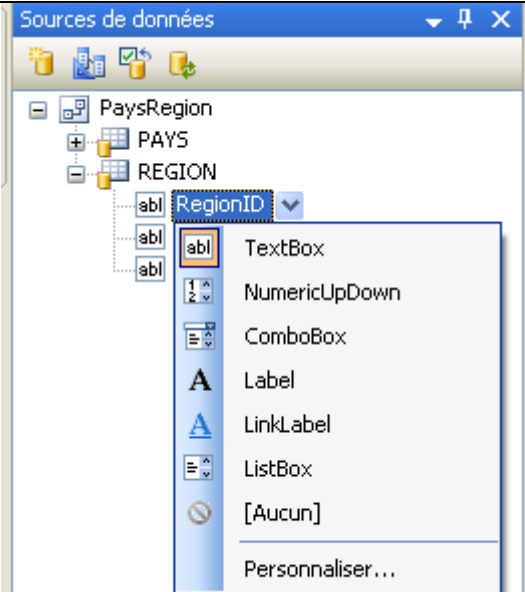
Grâce à certains outils de Visual Studio, il est possible de développer une fenêtre de présentation de données en 2 minutes, montre en main. C'est ce qu'on appelle le RAD (Rapid Application Development).

Les contrôles liés aux données seront créés en faisant glisser des éléments depuis la fenêtre **Sources de données** jusqu'au formulaire de l'application Windows.

III.2 GRAPHIQUEMENT

Avant d'exécuter l'opération de glissement, il faut définir les contrôles à créer sur le formulaire en sélectionnant un contrôle dans la liste des contrôles de chaque élément.

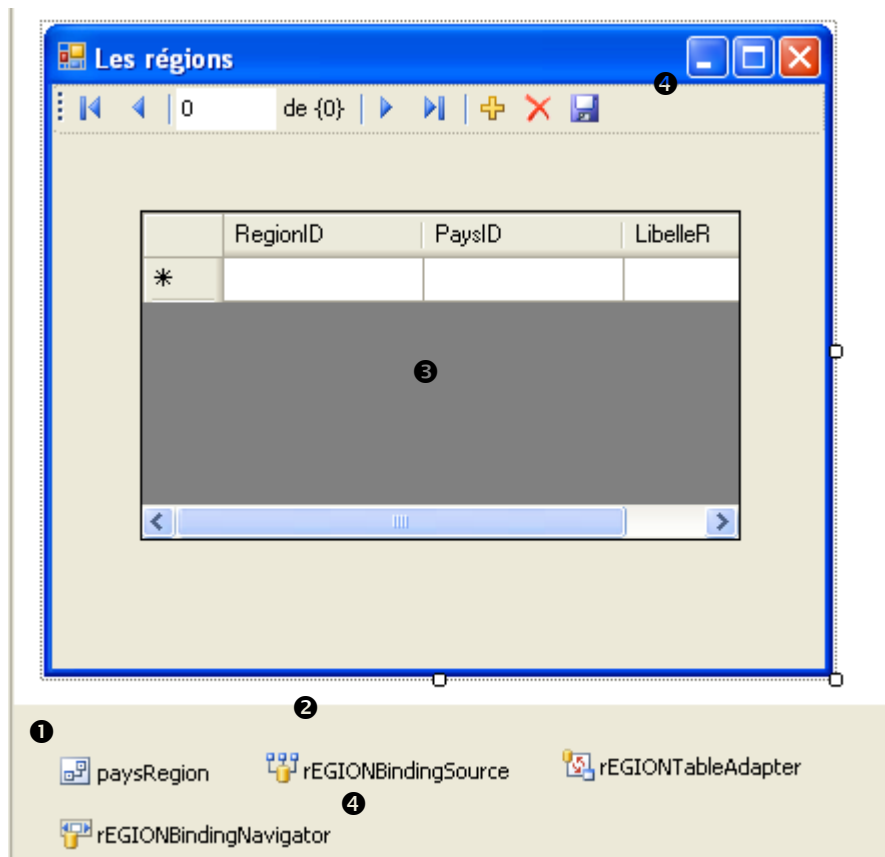
1. Choisir entre l'affichage de l'intégralité de la table dans un seul contrôle (**DataGridView**) ou l'afficher de chaque colonne dans un contrôle distinct (**Détails**).
2. Ensuite, si vous souhaitez afficher les éléments dans des contrôles distincts, sélectionner le contrôle individuel à créer pour chaque élément. (Chaque élément possède un contrôle par défaut utilisé si aucun contrôle spécifique n'est sélectionné).

Affichage ou non de l'intégralité de la table	Contrôles distincts
	

En sélectionnant Personnaliser, il est possible d'ajouter à la fenêtre **Sources de données** des contrôles situés dans la **Boîte à outils**.

Exemple 1 : Gestion des régions

Après avoir sélectionné l'option **DataGridView**, faire glisser le nœud REGION sur le formulaire.



Visual Studio 2005 a ajouté plusieurs entités à la fenêtre :

- ❶ Un **DataSet** typé de type **paysRegion**, qui représente la source de données, ainsi qu'un **TableAdapter** de type **RegionTableAdapter**.
- ❷ Un contrôle non visuel de type **BindingSource** qui représente la liaison entre la source de données (le DataSet), et les contrôles visuels de présentation et d'édition des données.
- ❸ Un contrôle visuel de type **DataGridView** qui présente la liste des régions, et qui permet d'éditer chaque cellule.
- ❹ Un contrôle visuel de type **BindingNavigator** qui contient des boutons pour naviguer dans la liste des régions

Visual Studio a également généré deux méthodes :

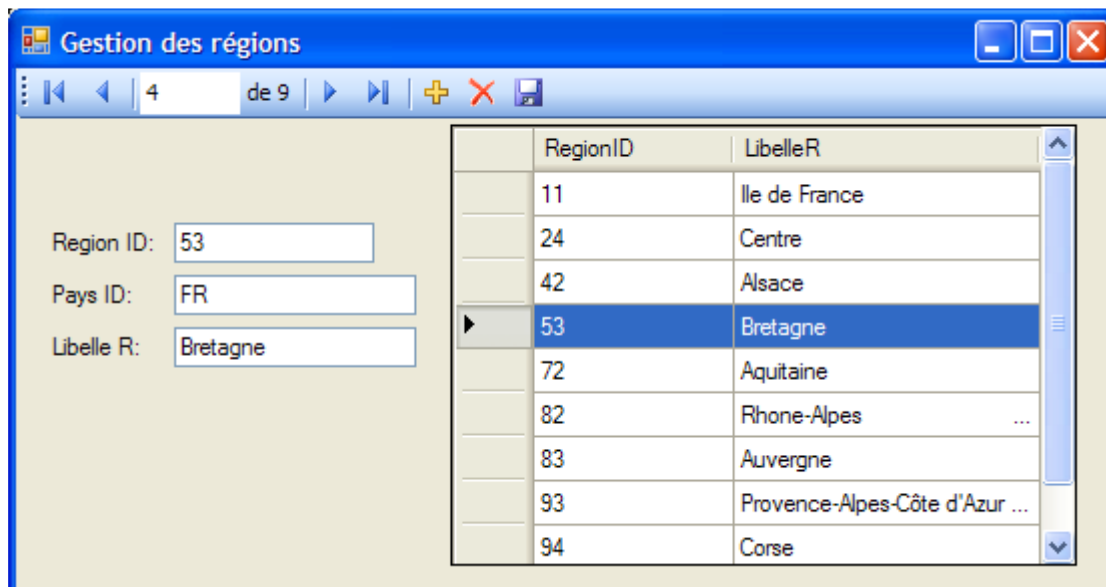
```
private void Form_Load(object sender, EventArgs e)
{
    // TODO : cette ligne de code charge les données dans la table
    // 'dsPaysRegion.REGION'. Vous pouvez la déplacer ou la
    // supprimer selon vos besoins.
    this.rREGIONTableAdapter.Fill(this.dsPaysRegion.REGION);
}
```

Cette méthode contient le code, qui permet de remplir la table Regions, à partir de la base de données, à l'aide du **TableAdapter** associé à la table Regions.

```
private void rREGIONBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
{
    this.Validate();
    this.rREGIONBindingSource.EndEdit();
    this.rREGIONTableAdapter.Update(this.dsPaysRegion.REGION);
}
```

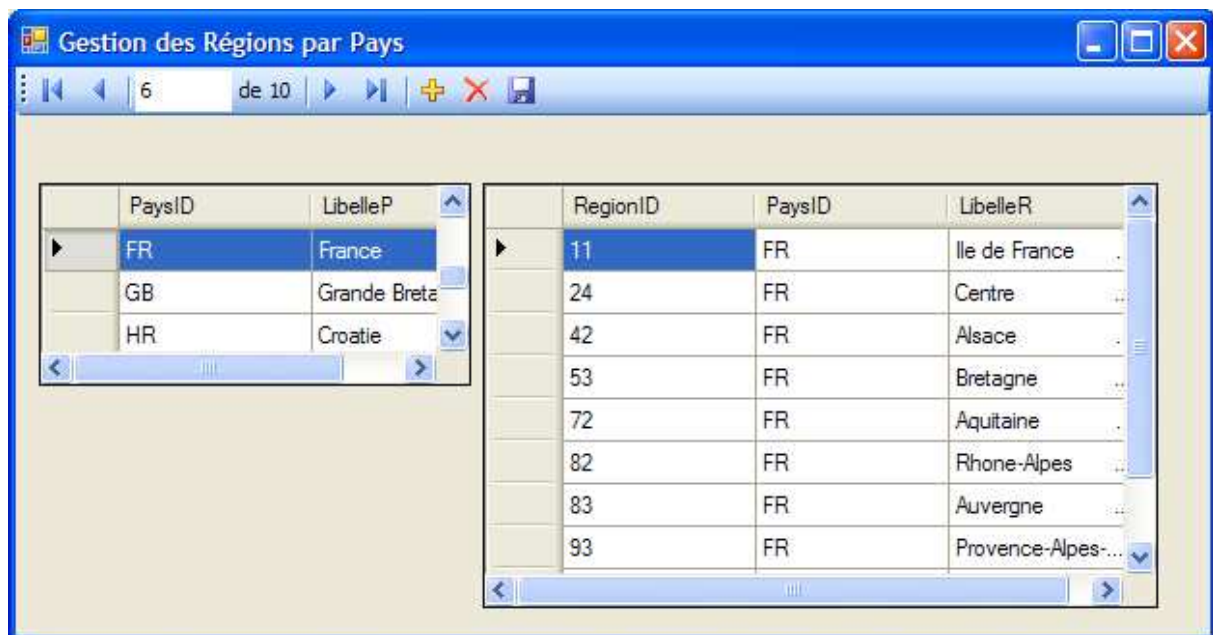
Cette méthode contient le code, qui permet de sauver dans la base les modifications effectuées, à l'aide du **TableAdapter** associé à la table Regions, lors du click sur le bouton Sauvegarde du contrôle BindingNavigator.

En cachant la colonne PaysId du **DataGridView**, et après avoir choisi l'option Détails sur le nœud Régions , en le faisant glisser sur la feuille, on obtient :



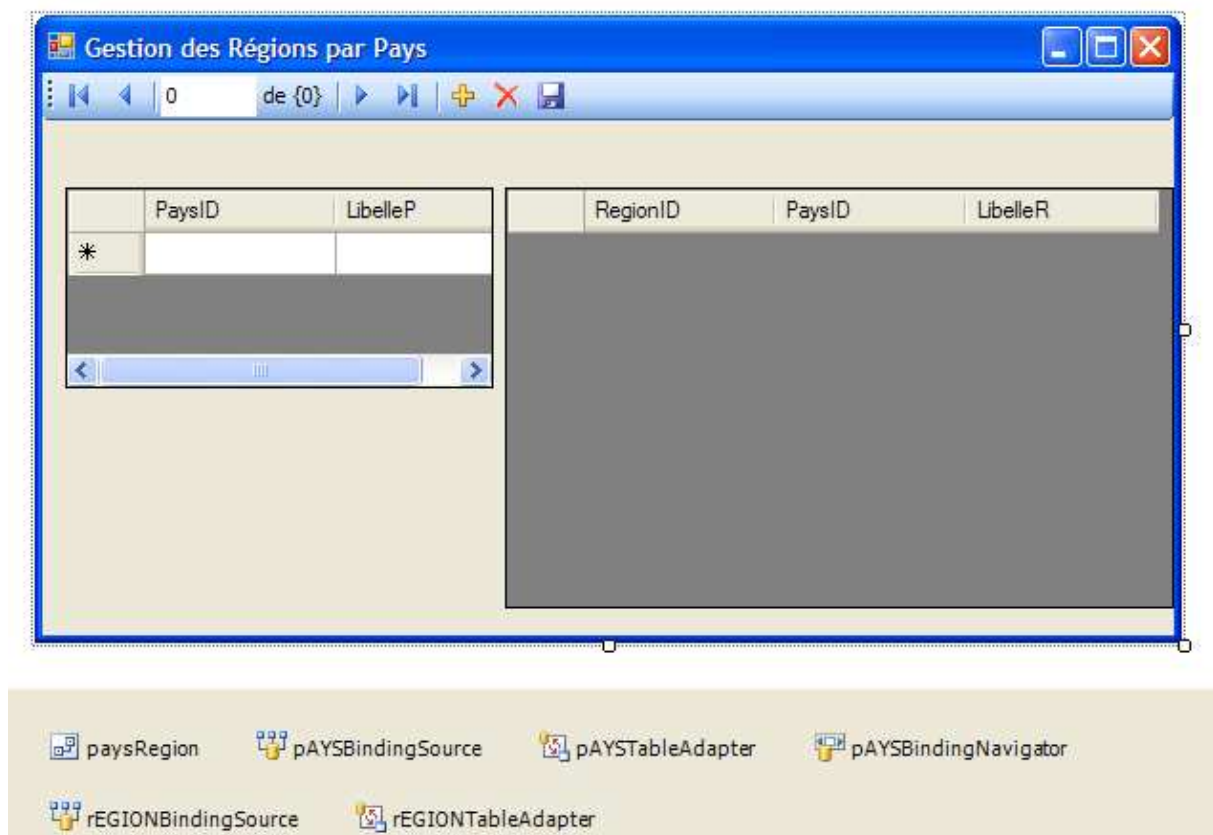
Cette grille est très pratique pour l'édition et l'insertion des lignes d'une table.

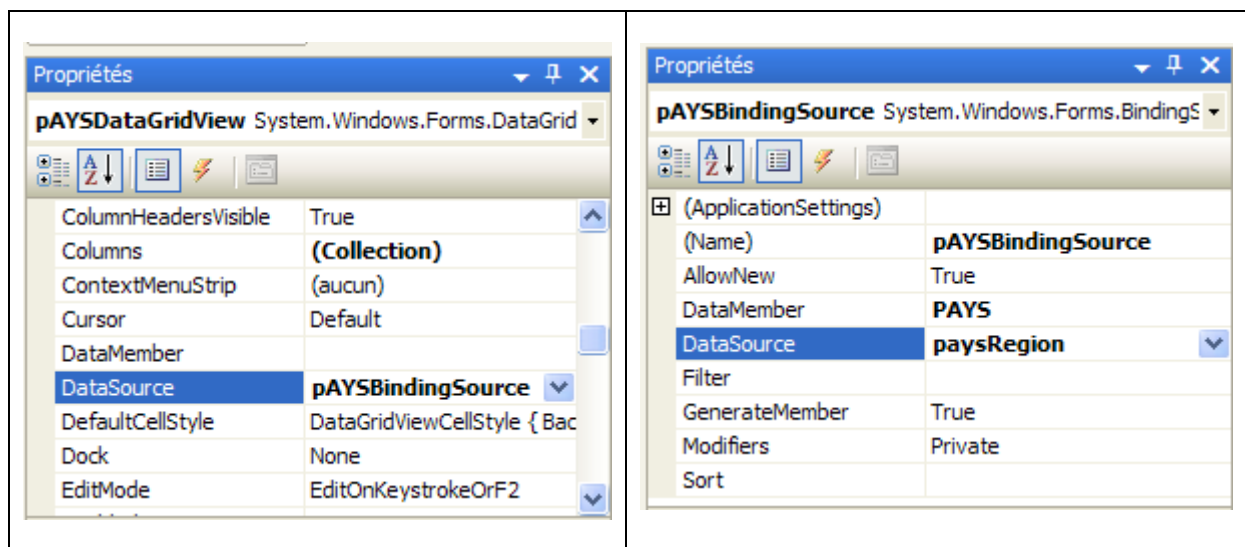
Exemple 2 : Gestion des régions par pays



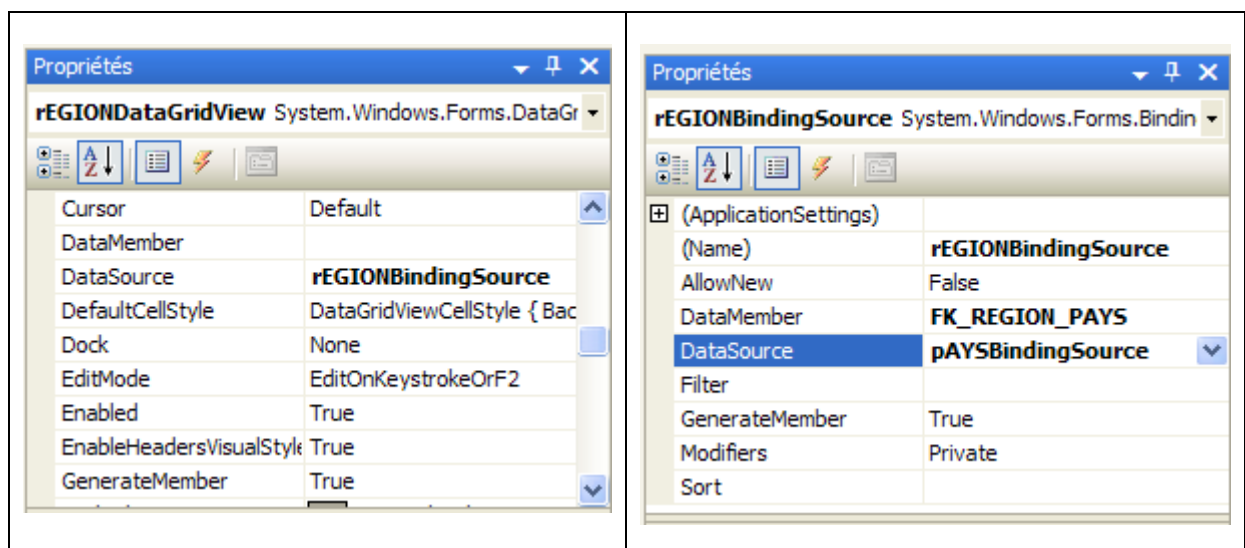
Les contrôles de type BindingSource permettent d'exploiter simplement une relation « 1 à plusieurs ».

En conception :





pAYSDataGridView est lié au contrôle Bindingsource **pAYSBindingSource**, qui est lié à la table **Pays** du DataSet **paysRegion**.



rREGIONDataGridView est lié au contrôle Bindingsource **rREGIONBindingSource**, qui est lié au premier contrôle Bindingsource **pAYSBindingSource** avec pour valeur de la propriété **DataMember** le nom de la relation.

rREGIONBindingSource est alors capable de détecter les changements de sélection de pays sur **pAYSBindingSource**, et ne retourner que les régions du pays sélectionné.

Il est possible de contrôler la saisie dans le groupe de données en interceptant certains événements déclenchés lors de la modification de valeurs dans des lignes de données. Il faut alors créer des gestionnaires d'événements (dans des classes partielles) pour valider les modifications (événements *ColumnChanging*, *RowChanging*).

III.3 LE COMPOSANT BINDINGSOURCE

Le composant **BindingSource** présente les intérêts suivants.

- En premier lieu, il simplifie les contrôles de liaison d'un formulaire aux données en **fournissant une couche d'indirection** entre les contrôles de présentation et d'édition de données, et les sources de données (possibilité de changer de source de données sans toucher au code)
- En second lieu, il peut agir comme une source de données fortement typée
- Grâce à ses propriétés *Filter* et *Sort*, les données peuvent être filtrées ou triées avant la présentation.

Comme vu précédemment, on peut affecter à un composant **BindingSource** une source de données de type `DataSet` à la propriété ***DataSource*** à condition de préciser le nom de la table source (propriété ***DataMember***).

On peut aussi affecter à cette propriété

- N'importe quel objet qui implémente l'interface `IEnumerable`
- N'importe quel objet qui représente un tableau (dont la classe dérive de **`System.Array`**)

IV ANNEXES

IV.1 LES MEMBRES DE LA CLASSE DATASET

Les principales propriétés du DataSet

Nom	Description
DataSetName	Obtient ou définit le nom du DataSet en cours.
DefaultViewManager	Obtient une vue personnalisée des données contenues dans le DataSet , permettant de filtrer, rechercher et naviguer à l'aide d'un DataViewManager personnalisé.
HasErrors	Obtient une valeur indiquant s'il existe des erreurs dans les objets DataTable de ce DataSet .
IsInitialized	Obtient une valeur qui indique si DataSet est initialisé.
Relations	Obtient la collection des relations qui relient des tables et permettent de naviguer des tables parentes aux tables enfants.
Tables	Obtient la collection des tables contenues dans le DataSet .

Les principales méthodes du DataSet

Nom	Description
AcceptChanges	Valide toutes les modifications apportées à ce DataSet depuis son chargement ou depuis le dernier appel à AcceptChanges.
Clear	Efface toutes les données de DataSet en supprimant toutes les lignes de l'ensemble des tables.
CreateDataReader	Surchargé. Retourne un DataTableReader avec un jeu de résultats par DataTable , dans la même séquence que les tables dans la collection Tables.
Equals	Surchargé. Détermine si deux instances de Object sont égales. (hérité de Object.)
GetChanges	Obtient une copie du DataSet contenant l'ensemble des modifications qui lui ont été apportées depuis son dernier chargement ou depuis l'appel à AcceptChanges .
GetXml	Retourne la représentation XML des données stockées dans le DataSet .
GetXmlSchema	Retourne le schéma XML de la représentation XML des données stockées dans le DataSet .
HasChanges	Obtient une valeur indiquant si DataSet contient des modifications, notamment des lignes nouvelles, supprimées ou modifiées.

InferXmlSchema	Applique le schéma XML du DataSet .
Merge	Surchargé. Fusionne un DataSet , un DataTable ou un tableau d'objets DataRow dans le DataSet ou le DataTable en cours.
ReadXml	Lit le schéma et les données XML dans le DataSet .
ReadXmlSchema	Lit un schéma XML dans le DataSet .
RejectChanges	Annule toutes les modifications apportées à DataSet depuis sa création ou le dernier appel à DataSet.AcceptChanges .
Reset	Rétablit l'état d'origine de DataSet . Les sous-classes doivent substituer Reset pour rétablir l'état d'origine de DataSet .
WriteXml	Écrit des données XML, et éventuellement le schéma, à partir du DataSet .
WriteXmlSchema	Écrit la structure DataSet sous la forme d'un schéma XML.

Les principaux évènements du DataSet

Nom	Description
Initialized	Se produit une fois DataSet initialisé.
MergeFailed	Se produit lorsque des DataRow cible et source possèdent la même valeur de clé primaire et que EnforceConstraints a la valeur true.

IV.2 LES MEMBRES DE LA CLASSE DATATABLE

Les principales propriétés du DataTable

Nom	Description
ChildRelations	Obtient la collection des relations enfants de ce DataTable.
Columns	Obtient la collection des colonnes qui appartiennent à cette table.
Constraints	Obtient la collection de contraintes gérée par cette table.
DataSet	Obtient le DataSet auquel cette table appartient.
DefaultView	Obtient une vue personnalisée de la table qui peut comprendre une vue filtrée ou une position de curseur.
ExtendedProperties	Obtient la collection d'informations utilisateur personnalisées.
HasErrors	Obtient une valeur indiquant s'il existe des erreurs dans

	une des lignes d'une table du DataSet auquel appartient la table.
IsInitialized	Obtient une valeur qui indique si DataTable est initialisé.
MinimumCapacity	Obtient ou définit la taille de départ initiale pour cette table.
Namespace	Obtient ou définit l'espace de noms de la représentation XML des données stockées dans le DataTable .
ParentRelations	Obtient la collection des relations parentes de ce DataTable .
PrimaryKey	Obtient ou définit un tableau de colonnes qui fonctionnent comme des clés primaires pour la table de données.
Rows	Obtient la collection des lignes qui appartiennent à cette table.
TableName	Obtient ou définit le nom de DataTable .

Les principales méthodes du DataTable

Nom	Description
AcceptChanges	Valide toutes les modifications apportées à cette table depuis le dernier appel à AcceptChanges.
BeginInit	Commence l'initialisation d'un DataTable qui est utilisé dans un formulaire ou par un autre composant. L'initialisation se produit au moment de l'exécution.
BeginLoadData	Désactive les notifications, la gestion d'index et les contraintes lors du chargement de données.
Clear	Efface toutes les données de DataTable .
Clone	Clone la structure de DataTable , y compris tous les schémas et contraintes DataTable .
CreateDataReader	Retourne un DataTableReader correspondant aux données dans ce DataTable .
EndInit	Met fin à l'initialisation d'un DataTable qui est utilisé dans un formulaire ou par un autre composant. L'initialisation se produit au moment de l'exécution.
EndLoadData	Active les notifications, la gestion d'index et les contraintes après le chargement de données.
GetChanges	Obtient une copie du DataTable contenant l'ensemble des modifications qui lui ont été apportées depuis son dernier chargement ou depuis l'appel à AcceptChanges .

GetDataTableSchema	Cette méthode retourne une instance de XmlSchemaSet contenant le code WSDL qui décrit le DataTable pour les services Web.
GetErrors	Obtient un tableau d'objets DataRow qui contiennent des erreurs.
ImportRow	Copie DataRow dans un DataTable en préservant tous les paramètres de propriété, ainsi que les valeurs d'origine et actuelles.
Load	Remplit un DataTable avec des valeurs issues d'une source de données à l'aide du IDataReader fourni. Si DataTable contient déjà des lignes, les données entrantes à partir de la source de données sont fusionnées avec les lignes existantes.
LoadDataRow	Recherche et met à jour une ligne spécifique. Si aucune ligne correspondante n'est détectée, une nouvelle ligne est créée à l'aide des valeurs données.
Merge	Fusionnez le DataTable spécifié avec le DataTable actuel.
NewRow	Crée un nouveau DataRow possédant le même schéma que la table.
ReadXml	Lit le schéma et les données XML dans le DataTable .
ReadXmlSchema	Lit un schéma XML dans le DataTable .
RejectChanges	Restaure toutes les modifications apportées à la table depuis son chargement ou le dernier appel à AcceptChanges .
Reset	Rétablit l'état d'origine de DataTable .
Select	Obtient un tableau d'objets DataRow .
WriteXml	Écrit le contenu actuel du DataTable au format XML.
WriteXmlSchema	Écrit la structure de données actuelle du DataTable sous la forme d'un schéma XML.

IV.3 LES MEMBRES DE LA CLASSE DATAADAPTER

Les principales propriétés du DataAdapter

Nom	Description
AcceptChangesDuringFill	Obtient ou définit une valeur indiquant si AcceptChanges est appelé sur DataRow après son ajout à DataTable durant les opérations Fill .
AcceptChangesDuringUpdate	Obtient ou définit si AcceptChanges est appelé pendant un Update.
ContinueUpdateOnError	Obtient ou définit une valeur qui spécifie si une exception doit être générée en cas d'erreur

	pendant la mise à jour d'une ligne.
FillLoadOption	Obtient ou définit LoadOption qui détermine comment l'adaptateur remplit DataTable du DbDataReader.
MissingMappingAction	Détermine l'action à effectuer si les données entrantes ne possèdent pas de table ou de colonne correspondante.
MissingSchemaAction	Détermine l'action à effectuer si le schéma DataSet existant ne correspond pas aux données entrantes.
ReturnProviderSpecificTypes	Obtient ou définit si la méthode Fill doit retourner des valeurs spécifiques au fournisseur ou des valeurs communes conformes CLS.
TableMappings	Obtient une collection qui fournit le mappage principal entre une table source et DataTable .

Les principales méthodes du DataAdapter

Nom	Description
Fill	Surchargé. Ajoute ou actualise les lignes de DataSet pour correspondre à celles de la source de données à l'aide du nom DataSet , et crée un DataTable.
FillSchema	Ajoute DataTable au DataSet spécifié.
GetFillParameters	Obtient les paramètres définis par l'utilisateur lors de l'exécution d'une instruction SQL SELECT.
ResetFillLoadOption	Réinitialise FillLoadOption à son état par défaut et entraîne une réponse de Fill à AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill	Détermine si la propriété AcceptChangesDuringFill doit être persistante.
ShouldSerializeFillLoadOption	Détermine si la propriété FillLoadOption doit être persistante.
Update	Appelle les instructions INSERT, UPDATE ou DELETE respectives pour chaque ligne insérée, mise à jour ou supprimée dans le DataSet spécifié à partir d'un DataTable appelé "Table".

Les principaux évènements du DataAdapter

Nom	Description
FillError	Retourné lorsqu'une erreur se produit pendant une opération de remplissage.

IV.4 LES MEMBRES DE LA CLASSE TABLEADAPTER

Méthodes et propriétés fréquemment utilisées des TableAdapters

Nom	Description
TableAdapter.Fill	Remplit la table de données associée au TableAdapter avec les résultats de la commande SELECT du TableAdapter ..
TableAdapter.Update	Renvoie les modifications à la base de données.
TableAdapter.GetData	Retourne un nouveau DataTable rempli avec des données.
TableAdapter.Insert	Crée une nouvelle ligne dans la table de données.
TableAdapter.ClearBeforeFill	Détermine si une table de données doit être vidée avant d'appeler l'une des méthodes <i>Fill</i> .

Etablissement référent

Marseille Saint Jérôme

Equipe de conception

Elisabeth Cattaneo

Remerciements :

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.
« toute représentation ou reproduction intégrale ou partielle faite sans le
consentement de l'auteur ou de ses ayants droits ou ayants cause est
illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction
par un art ou un procédé quelconques. »

Date de mise à jour 05/05/2008
afpa © Date de dépôt légal mai 08



**afpa / Direction de l'Ingénierie 13 place du Générale de Gaulle / 93108 Montreuil
Cedex
association nationale pour la formation professionnelle des
adultes
Ministère des Affaires sociales du Travail et de la
Solidarité**