# C# - FICHE PRATIQUE N° 7 – Solutions et compléments

Une petite gestion de personnel
Grille de données, chargement partiel d'un dataset.

## A/ Grille de données

- Il est intéressant de créer un autre formulaire comportant une grille dont la propriété « DataSource » est fixée à un objet « DataViewManager » construit à partir du DataSet passé en paramètre. Ceci permet de montrer les possibilités offertes par les DataGrid en matière de navigation dans les tables, de tri des colonnes (clic sur le titre), …
- Les possibilités de paramétrage du style des grilles en mode conception sont très nombreuses. Le mieux est de faire une rapide démonstration…
- Il faut bien comprendre qu'une grille peut posséder plusieurs styles, chaque style étant associé à une table (propriété « MappingName »), et donc utilisé lorsque la grille affiche les données de cette table.
- La technique de recherche d'un enregistrement utilisée dans le formulaire « employé » peut être également appliquée à une grille.
- L'application fournie comporte un formulaire « grille2 » qui illustre l'ensemble de ces possibilités.

### Ajout d'une colonne liste déroulante

Il est possible de définir par programme d'autres types de style pour les colonnes d'un DataGrid. On peut tout faire ou presque…

La bibliothèque « Navigation » fournit une classe « DataGridComboBoxColumn » destinée à présenter une liste déroulante dans un DataGrid.

Pour l'utiliser, il suffit de :

- Définir un style pour une grille en mode conception.
- Déclarer un objet « DataGridComboBoxColumn » dans le formulaire.
- Instancier cet objet et renseigner ses propriétés dans le constructeur du formulaire.
- Intégrer l'objet créé au style défini pour la grille.

Les principales propriétés sont :

- MappingName, même rôle que pour les autres styles (colonne liée).
- Combo, ComboBox liée au contrôle.
- DataSource, DataView ou DataTable source de la liste.
- ValueMember, champ fournissant la valeur, obligatoirement une clé primaire.
- DisplayMember, champ à afficher.
- HeaderText, titre de la colonne.
- DessinerListe, indique s'il faut ou non dessiner la liste pour les lignes autres que la ligne en cours d'édition (false par défaut).
- DeroulerListe, indique s'il fait dérouler automatiquement la liste.

Pour un bon résultat visuel, il faut fixer la valeur de la propriété « PreferredRowHeight » au minimum à 21 dans le style lié à la grille. Lors de l'édition à l'aide de la liste déroulante, il est possible d'entrer la valeur nulle à l'aide de la combinaison de touches CTRL-0 (contrôle-zéro). Cette classe prend en charge les valeurs par défaut fixées dans le dataset. L'application fournie comporte un exemple d'utilisation dans le formulaire « grille3 ».

**Problème détecté avec les DataGrids**

Un bug un peu troublant qu'il vaut mieux connaître…

Dans la situation suivante (simplement déduite de l'exemple rencontré…) :

- Une DataGrid est basée sur un DataView.
- La propriété « ApplyDefaultSort » du DataView n'est pas à true (ou l'utilisateur a trié en cliquant sur l'entête d'une colonne).
- Le DataTable comporte une clé étrangère.
- La table est vide.

Avec certaines valeurs pour la clé étrangère, la ligne saisie se recopie dans la seconde ligne du DataGrid qui devient alors liée à un enregistrement « fantôme »…

Le problème apparaît également quand on exploite une relation dans un DataGrid et que le jeu d'enregistrement enfant est vide à l'ouverture.

Le mieux est d'essayer pour comprendre :

- Créer une grille pour éditer la table « employé ».
- Baser cette grille sur un DataView trié en ordre croissant des noms.
- Vider les enregistrements de la table.
- Ajouter un employé à l'aide de la grille (essayer tous les codes service en se remettant à chaque fois dans la situation de départ).

Solution : l'utilisateur résout le problème en cliquant sur un entête de colonne, ce qui provoque un tri et fait disparaître la ligne « fantôme ». Il est possible de contourner ce problème de la manière suivante :

- Ecrire dans le constructeur du formulaire :

```
dbDv_employe.ApplyDefaultSort=true;
if (dbDv_employe.Count==0)  // Eviter le bug...
{
    dgTs_employe.AllowSorting=false;
    dbDv_employe.Table.RowChanged+=new DataRowChangeEventHandler(Table_RowChanged_contournerBug);
}
```

- Ajouter un gestionnaire d'événement :

```
private void Table_RowChanged_contournerBug(object sender,DataRowChangeEventArgs e)
{
    if (e.Action==DataRowAction.Add)
    {
        dgTs_employe.AllowSorting=true;
        dbDv_employe.Table.RowChanged -=
                                    new DataRowChangeEventHandler(Table_RowChanged_contournerBug);
    }
}
```

Ce n'est pas très élégant, mais en attendant le service pack…

## B/ Chargement partiel d'un dataSet

Le mode déconnecté d'ADO.net est déroutant au premier abord, mais il est intéressant :

- Il décharge considérablement le SGBD qui ne doit pas maintenir systématiquement des connexions ouvertes à chaque parcours d'un jeu d'enregistrements.
- Il minimise le trafic réseau puisqu'une fois chargées, les données sont consultées localement. Un exemple simple : si le formulaire « employé » était réalisé dans un contexte « classique » d'accès aux données, l'ensemble des n-uplets de la table « service » seraient rechargés à chaque ouverture (et donc circuleraient sur le réseau) pour alimenter la liste déroulante.