C# - FICHE PRATIQUE N° 4 – Pour aller plus loin

Une petite gestion de personnel
création de composants, délégués, propriétés.

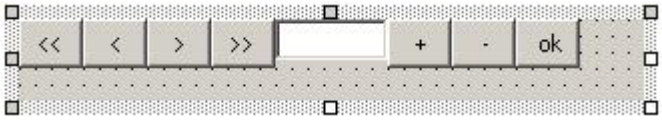
Il est possible de réaliser un composant de navigation de manière assez simple. Cela permet de voir :

Comment créer un contrôle utilisateur.
Comment créer et utiliser un « délégué ».

Remarque : la gestion des erreurs n'est pas traitée ici, consulter le code du composant DbNavigateur pour l'intégrer à cet exercice.

1. Création du composant de navigation

- Faire un clic droit sur le projet dans l'explorateur de solutions et choisir « ajouter un nouvel élément ».
- Choisir le type « Contrôle utilisateur » et le nommer « BarreNavigation ».
- Créer l'ensemble de boutons ci-dessus (ou les copier/coller depuis le projet de la fiche 3).



- Ajuster la taille du contrôle.
- Créer le membre privé permettant l'accès aux données et une méthode « Init » permettant de le renseigner :

```
private CurrencyManager dbBindContext;

public void Init(BindingManagerBase p_dbBindContext)
{
    dbBindContext=(CurrencyManager)p_dbBindContext;
    dbBindContext.PositionChanged += new EventHandler(affichePosCpt);
    dbBindContext.Position=0;
}
```

- L'avant dernière instruction est destinée à mettre en place la gestion de l'événement lié au déplacement dans le jeu d'enregistrements. Elle doit être complétée par la définition de la méthode de traitement :

```
private void affichePosCpt(object sender, EventArgs e)
{
    string message=(dbBindContext.Position + 1).ToString();
    message+="/";
    message+=(dbBindContext.Count).ToString();
    tb_posCpt.Text=message;
}
```

- Réaliser l'ensemble des méthodes liées au clic sur les boutons de navigation

```
private void bt_premier_Click(object sender, System.EventArgs e)
{
    dbBindContext.Position=0;
}
```

```
private void bt_precedent_Click(object sender, System.EventArgs e)
{
    dbBindContext.Position--;
}

private void bt_suivant_Click(object sender, System.EventArgs e)
{
    dbBindContext.Position++;
}

private void bt_dernier_Click(object sender, System.EventArgs e)
{
    dbBindContext.Position=dbBindContext.Count - 1;
}

private void bt_plus_Click(object sender, System.EventArgs e)
{
    dbBindContext.AddNew();
}

private void bt_moins_Click(object sender, System.EventArgs e)
{
    dbBindContext.RemoveAt(dbBindContext.Position);
}

private void bt_ok_Click(object sender, System.EventArgs e)
{
    dbBindContext.EndCurrentEdit();
}
```

2. Remarque concernant le bouton OK et test du composant

Si on voulait mettre à jour directement la table du SGBD à l'aide du bouton OK sans passer par le gestionnaire de l événement « RowChanged », il faudrait :

- Mettre fin à l'édition en cours en utilisant la méthode « EndCurrentEdit ».
- Récupérer la ligne en cours à l'aide de la propriété « Current » du contexte de liaison de données. Cet objet est intéressant, il permet de modifier les valeurs contenues dans le DataSet car il donne accès à chacun des attributs (ligne["designation"] = "Emboutissage" ; par exemple).
- La ligne permet de récupérer la table et le DataSet.
- La table connaît son nom…
- Ceci conduit au code ci-dessous :

```
private void bt_ok_Click(object sender, System.EventArgs e)
{
    dbBindContext.EndCurrentEdit();
    DataRow ligne = ((DataRowView)dbBindContext.Current).Row;
    DataTable maTable=ligne.Table;
    DataSet monDataSet=maTable.DataSet;
    string nomTable = maTable.TableName;
    dbAdapter.Update(monDataSet,nomTable);
}
```

Remarque : la solution de l'événement « RowChanged » est meilleure, mais ce code est intéressant…

Il reste à tester ce composant de navigation, sur le formulaire Fm_service par exemple :

- Générer le projet, le composant est automatiquement ajouté à la palette d'outils.

- Ajouter un objet « BarreNavigation » sur le formulaire Fm_service (dbBn_service).
- Ajouter le code ci-dessous sur l'événement Load du formulaire :

```
private void Fm_service_Load(object sender, System.EventArgs e)
{
    dbBn_service.Init(BindingContext[dbDs_service1,"tp1_service"]);
}
```

3. Notion de délégué

Un délégué peut être considéré comme une variable de type « liste de fonctions ». Un délégué peut donc « contenir » : null, une fonction, ou plusieurs fonctions (ou encore plusieurs fois la même fonction).

Pour créer un (ou plusieurs) délégué(s), il faut tout d'abord déclarer le type correspondant :

```
public delegate void MonTypeDelegue(int val);
```

Ceci est fait dans un nameSpace, à l'extérieur de toute classe. Dans l'exemple, « MonTypeDelegue » servira à créer des délégués pouvant contenir des fonctions dont le prototype est : void maFonction(int val).

On peut ensuite déclarer un délégué dans n'importe quelle classe du nameSpace :

```
private MonTypeDelegue delegue;
```

Considérons maintenant les deux fonctions suivantes dont le prototype correspond au type de délégué (le nom du paramètre est sans importance) :

```
private void methode1(int valeur)
{
   MessageBox.Show("Methode 1 : " + valeur);
}
private void methode2(int val)
{
   MessageBox.Show("Methode 2 : " + val*2);
}
```

On peut maintenant faire en sorte que le délégué « contienne » la première méthode en écrivant :

```
delegue = new MonTypeDelegue(methode1);
```

Dès lors, on peut lancer l'exécution de « methode1 » par :

```
delegue(5);
```

Il est possible d'ajouter la seconde méthode au délégué :

```
delegue += new MonTypeDelegue(methode2);
```

L'instruction ci-dessous provoquera successivement l'exécution des deux méthodes :

```
delegue(10);
```

On peut également retirer une fonction du délégué :

```
delegue -= new MonTypeDelegue(methode1);
```

Ou encore en retirer toutes les fonctions :

```
delegue=null;
```

L'appel ci-dessous provoquera alors une erreur d'exécution et l'arrêt du programme :

```
delegue(10);
```

Il est donc nécessaire de gérer cette éventuelle erreur lors de l'utilisation d'un délégué :

```
try
{
    delegue(5);
}
catch (Exception erreur)
{
    MessageBox.Show(erreur.ToString());
}
```

4. Ajout d'un délégué au composant de navigation

Il s'agit d'ajouter la possibilité d'exécuter une fonction particulière lors de l'ajout d'un enregistrement par l'intermédiaire du composant. Ceci permettra par exemple de proposer automatiquement un code lors de l'ajout d'un service.

Dans le code du composant « barreNavigation » :

- Déclarer le type de délégué (à l'extérieur de la classe barreNavigation) :

```
public delegate void FonctionSurAjout();
```

- Ecrire une seconde méthode Init :

```
public void Init(   BindingManagerBase p_dbBindContext, FonctionSurAjout p_surAjout)
{
    Init(p_dbBindContext);
    surAjout=p_surAjout;
}
```

- Modifier la méthode associée au bouton bt_plus :

```
private void bt_plus_Click(object sender, System.EventArgs e)
{
    dbBindContext.AddNew();
    if (surAjout!=null)
    {
        surAjout();
    }
}
```

Dans le code du formulaire Fm_service :

- Ajouter la méthode permettant de générer une valeur pour le code (cette méthode de génération du code est nettement insuffisante !) :

```
private void onAjout()
{
    string code ;
    code="s" + (BindingContext[dbDs_service1,"tp1_service"].Count + 1);
    tb_code.Text=code;
}
```

- Modifier l'appel à Init au chargement du formulaire :

```
private void Fm_service_Load(object sender, System.EventArgs e)
{
    dbBn_service.Init(BindingContext[dbDs_service1,"tp1_service"], new FonctionSurAjout(onAjout));
}
```

Remarque : cette technique d'initialisation d'un attribut est différente de celle utilisée pour le numéro d'employé, mais elles sont compatibles. Notamment, celle-ci ne s'applique qu'aux enregistrements ajoutés à l'aide du composant.

5. Notion de propriété

Une propriété est un mécanisme demandant au compilateur de générer automatiquement des fonctions d'accès en lecture et/ou en écriture du type « GetPropriété » et « SetPropriété ».

Prenons l'exemple d'une classe Personne, possédant les membres données suivants :

```
private string nom;
private string prenom;
private int anneeNaissance;
```

On peut définir une propriété permettant d'accéder au nom :

```
public string Nom
{
    get
    {
        return nom;
    }
    set
    {
        nom=value;  // expression utilisée pour l'affectation
    }
}
```

Ceci permettra d'écrire d'une manière plus commode que p.SetNom(…) ou p.GetNom() :

```
private Personne p=new Personne();
p.Nom = "Durand";
MessageBox.Show(p.Nom);
```

Remarque : l'usage est de donner le nom du membre accédé à la majuscule près. Une erreur fréquente en découle :

```
get
{
    return Nom;  // N majuscule
}
```

Cette écriture conduit à une récursivité infinie…

Il est possible de ne fournir qu'un accès en écriture ou en lecture (ici en écriture) :

```
public string Prenom
{
    set
    {
        prenom=value;
    }
}
```

Le mécanisme peut être plus complexe :

```
public string Patronyme
{
    get
    {
        return prenom+" "+nom;
    }
}
```

Ceci permettra d'écrire :

```
MessageBox.Show(p.Patronyme);
```

De même, les accès en écriture peuvent être plus élaborés :

```
public int Age
{
    get
    {
        int anneeCourante;
        anneeCourante=DateTime.Today.Year;
        return anneeCourante - anneeNaissance;
    }
    set
    {
        int anneeCourante=DateTime.Today.Year;
        anneeNaissance = anneeCourante - value;
    }
}
```

Ceci permettra d'écrire :

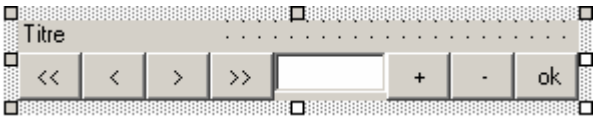
```
p.Age=25;    // cette affectation renseigne anneeNaissance

MessageBox.Show(p.Age.ToString());    // calcul basé sur anneeNaissance
```

6. Ajout d'un libellé au composant

Il s'agit maintenant d'ajouter un libellé qui sera affiché sur le composant de navigation (le nom de la table par exemple).

Ajouter un label sur le composant (lbl_titre) :



Fermer le formulaire Fm_service et régénérer l'application pour mettre à jour la barre de navigation du formulaire (elle doit être redimensionnée).

On veut pouvoir paramétrer le titre affiché par le composant. Une solution simple serait de prévoir un paramètre de plus dans la méthode Init :

```
public void Init(   BindingManagerBase p_dbBindContext, FonctionSurAjout p_surAjout, string p_titre)
{
    Init(p_dbBindContext);
    surAjout=p_surAjout;
    lbl_titre.Text=p_titre;
}
```

Il suffit de passer le titre voulu à l'appel de la fonction.

Cette solution fonctionne correctement, mais il y en a une autre qui rend la chose plus aisée : l'utilisation d'une propriété.

- Ajouter une propriété « Titre » au composant barreNavigation :

```
public string Titre
{
    get
    {
        return lbl_titre.Text;
    }
    set
    {
        lbl_titre.Text=value;
    }
}
```

- Régénérer la solution et consulter les propriété du contrôle dbBn_service dans le formulaire Fm_service. La propriété Titre est présente, il suffit d'en fixer la valeur au moment de la conception (« Services » par exemple). Bien entendu, cette valeur pourrait être modifiée à l'exécution.

7. Remarques

- Il est facile de regrouper un ensemble de composants au sein d'un projet de type « Bibliothèque de contrôles Windows ». On obtient alors un fichier « .dll » qui permet de réutiliser les outils réalisés dans une autre application en les ajoutant à la « boîte à outils ».
- On peut également utiliser l'héritage entre les composants en choisissant un élément de type « Contrôle utilisateur hérité ».
- L'application fournie implémente le composant « BarreNavigation ».