



Concepteur Développeur en Informatique

Développer des composants d'interface

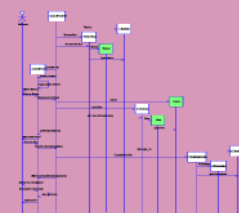
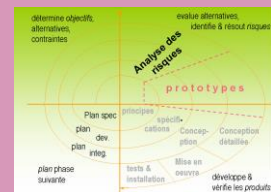
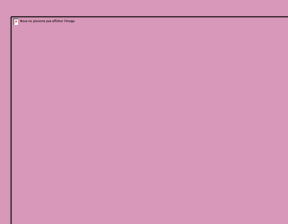
Windows Partie 6 Liaison de données : DataBinding

Accueil

Apprentissage

PAE

Evaluation



Localisation : U03-E03-S02

SOMMAIRE

1.	Introduction.....	3
2.	1er exemple : liaison des propriétés de deux contrôles.....	4
3.	Liaison des propriétés d'un objet métier à un contrôle	4
3.1.	Mise en place de la liaison de données.....	4
4.	Liaison de données au travers d'un objet BindingSource	6
4.1.	Création de la source de données	6
4.2.	Génération des composants d'interface liés	8
4.3.	Alimentation des contrôles	9
4.4.	Propriétés des contrôles liés	10
4.5.	Récupérer la référence de l'objet lié	11
4.6.	Gestion des erreurs.....	13
4.7.	Le cas de l'héritage et des spécialisations	14
5.	Une liaison Bi-Directionnelle	17
5.1.	Implémentation de la notification de modification de propriétés.....	18
6.	Fonctionnalités avancées	20
6.1.	Gestion des liaisons de second niveau.....	20
6.2.	Valider ou invalider les modifications	23
6.3.	Partager une même source de liaison entre 2 fenêtres	24

1. Introduction

Ce support est une illustration de techniques de liaison de données proposées dans nombre d'architectures logicielles.

Le DataBinding ou liaison de données est un mécanisme d'une grande richesse fonctionnelle que nous allons retrouver dans divers contextes où nous devons mettre en œuvre des opérations d'affichage et de maintenance de données, les données pouvant provenir de différentes sources comme des objets métiers en mémoire, des DataTables, ... et être présentées au sein d'interfaces aussi diversifiées que des Winforms, des WebForms, des vues Razor en MVC, des vues avec Angular, des applications Smartphones ou Tablettes, des interfaces WPF.

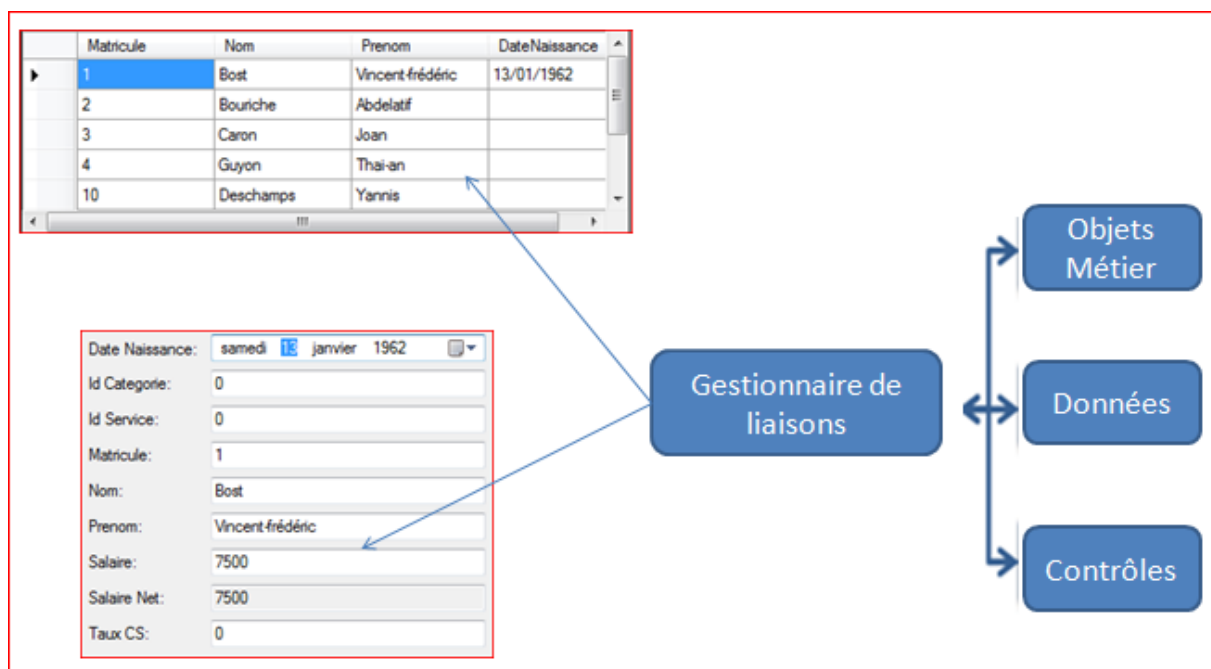
Le DataBinding permet de maintenir la valeur (donnée – data) d'une propriété d'un contrôle à jour (liée – binding) à la valeur de la propriété d'un objet fournissant des données (DataSource).

N'importe quel objet peut fournir une ou des données.

Il est donc possible de lier automatiquement la propriété Text d'un bouton avec la propriété Text d'une boîte de Texte.

Dans un contexte plus courant nous allons lier les propriétés de nos objets métiers aux propriétés de contrôles qui présentent une ou plusieurs propriétés.

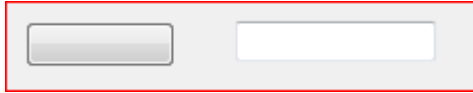
Le schéma ci-dessous illustre ce mécanisme. Nous avons ici une DataGrid qui est liée à une collection d'objets en mémoire et un formulaire affichant le détail des propriétés de l'élément courant. Nous reviendrons sur ce principe.



2. 1er exemple : liaison des propriétés de deux contrôles.

Nous allons réaliser ici la liaison par programme plutôt que par l'intermédiaire du Designer.

Nous avons créé une interface minimaliste qui comporte un bouton désigné btnCible et une boîte de texte désignée txtSource. Ces termes n'ont pas été choisis au hasard : nous avons bien une notion de source de données et de cible qui sera mise à jour en conséquence.



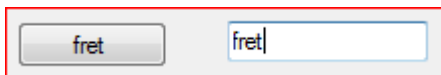
Nous allons donc associer à une propriété de notre bouton une source de données précisée selon deux termes : l'objet Source DataSource et le membre de celui-ci DataMember.

- La propriété affectée du bouton est la propriété Text
- L'objet source de données est la boîte de texte txtSource
- La propriété source de l'objet txtSource est Text

Voici donc le code qui va permettre cette liaison :

```
btnCible.DataBindings.Add("Text", txtSource, "Text");
```

Le test de notre code permet de vérifier qu'il fonctionne correctement :



3. Liaison des propriétés d'un objet métier à un contrôle

3.1. Mise en place de la liaison de données

Dans l'exemple suivant nous allons lier un objet métier de type Salarie à un ensemble de contrôles. Ce code sera exécuté sur un bouton Liaison. Dans un premier temps, nous utilisons un objet créé ex-nihilo.

Nous allons lier les propriétés de notre objet à des propriétés de contrôle par le biais des mécanismes de DataBinding réalisé dans un premier temps lors de l'exécution du programme.

Déclaration d'une variable globale au formulaire de type Salarie.

```
Salarie salarie;
```

Instanciation d'un objet de type Salarie et liaison de ses propriétés dans le gestionnaire d'événement associé au clic du bouton Création Salarie.

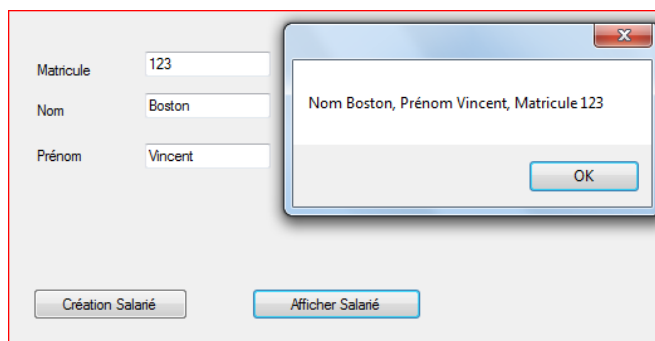
```
salarie = new Salarie("Bost", "Vincent", 123);  
txtMatricule.DataBindings.Add("Text", salarie, "Matricule");  
txtNom.DataBindings.Add("Text", salarie, "Nom");  
txtPrenom.DataBindings.Add("Text", salarie, "Prenom");
```

Nous obtenons le résultat suivant :



A screenshot of a Windows application window with a light gray background. It contains three text input fields stacked vertically. The first field is labeled 'Matricule' and contains the text '123'. The second field is labeled 'Nom' and contains the text 'Bost'. The third field is labeled 'Prénom' and contains the text 'Vincent'. Below these fields are two buttons: 'Création Salarié' on the left and 'Afficher Salarié' on the right. The entire form is enclosed in a red rectangular border.

Les propriétés de nos contrôles reflètent bien les propriétés de l'instance de salarié. Modifions la propriété Texte du contrôle txtNom et vérifions que cette modification est bien été enregistrée au niveau de l'objet source de données.



A screenshot of the same application window as before, but with a message box overlaid on top. The message box has a title bar with a close button (X) and contains the text 'Nom Boston, Prénom Vincent, Matricule 123'. Below the text is an 'OK' button. The background form shows the 'Nom' field now containing 'Boston' instead of 'Bost'. The 'Matricule' field still contains '123' and the 'Prénom' field still contains 'Vincent'. The buttons 'Création Salarié' and 'Afficher Salarié' are still visible at the bottom. The entire scene is enclosed in a red rectangular border.

Le programme semble fonctionner correctement, la propriété Nom affichée via une boîte de message a été correctement mise à jour.

4. Liaison de données au travers d'un objet BindingSource

Nous allons créer nos liaisons de données en passant par un composant tiers, le composant BindingSource. Ce dernier présente les avantages suivants :

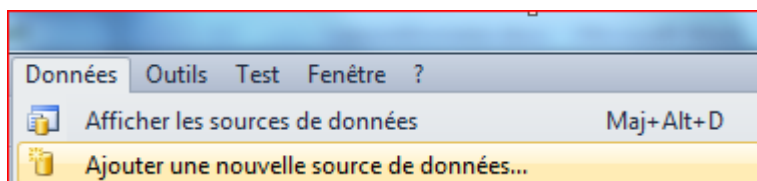
- En premier lieu, il simplifie les contrôles de liaison d'un formulaire aux données en fournissant une couche d'indirection entre les contrôles de présentation et d'édition de données, et les sources de données (possibilité de changer de source de données sans toucher au code).
- En second lieu, il 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
- Enfin, il permet de fournir une référence vers l'élément courant lié.

Nous devons définir, comme précédemment, la source de données et les membres associés.

4.1. Création de la source de données

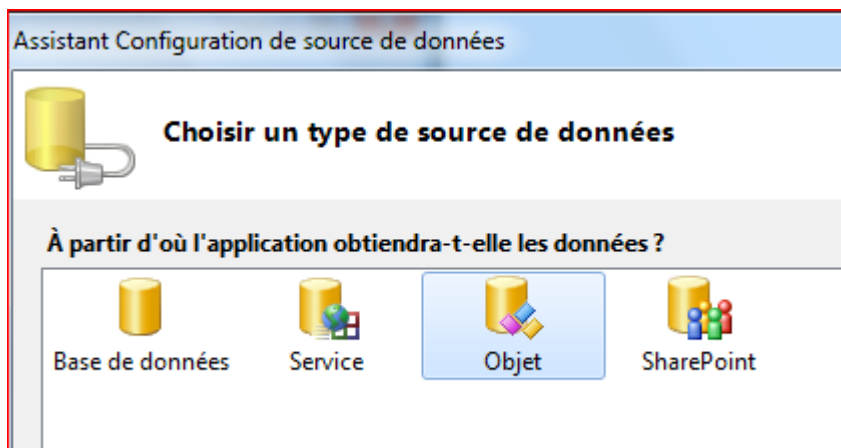
Plutôt que de définir les sources de données lors de la conception de chaque formulaire, nous allons définir celles-ci par le biais du concepteur visuel au niveau du projet d'application Windows.

Etape 1



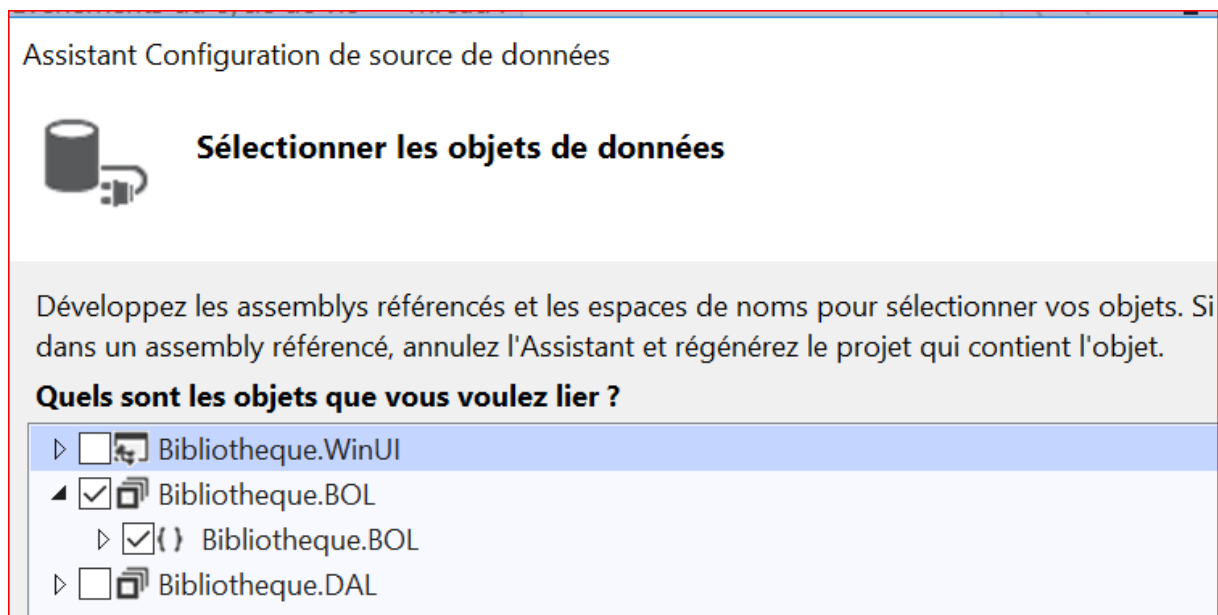
Choisissons le menu pour ajouter une source de données.

Etape 2

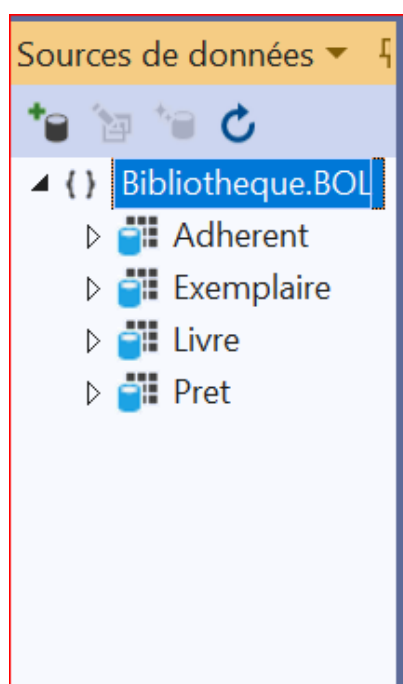


Choisissons le type d'objet de sources de données, ici **Objet**. Nous verrons les autres lors des étapes suivantes de la formation.

Etape 3



Retenons les types que nous sommes susceptibles d'utiliser comme sources de données dans le cadre de notre application.



Nous disposons maintenant d'un ensemble de sources de données de type Objets Métier que nous pouvons afficher dans la boîte à outils Sources de données.

Lors de la conception visuel d'un formulaire, nous constatons que différents symboles apparaissent à côté de nos types sources de données.

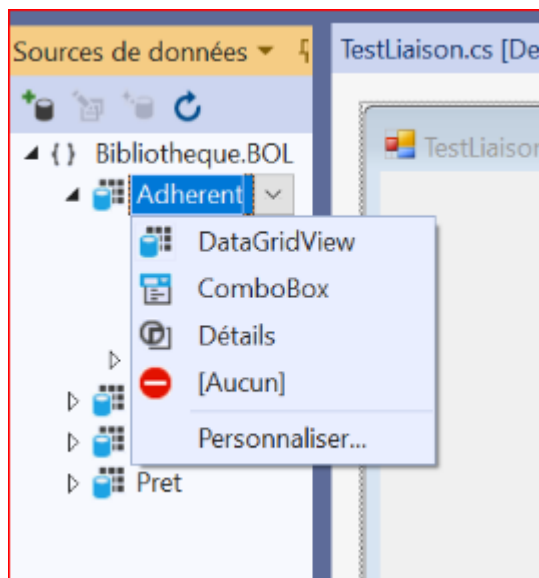
Il s'agit des type de contrôles qui seront générés lors du glisser déposer de la source de données sélectionnée sur le formulaire.

Il est possible d'obtenir un menu contextuel pour modifier le type de contrôle ou préciser les options de la liaison de donnée entre propriété de l'objet et propriété du contrôle.

4.2. Génération des composants d'interface liés

Dans ce premier exemple nous allons créer une liste déroulante qui permet de choisir un élément et d'en afficher le détail.

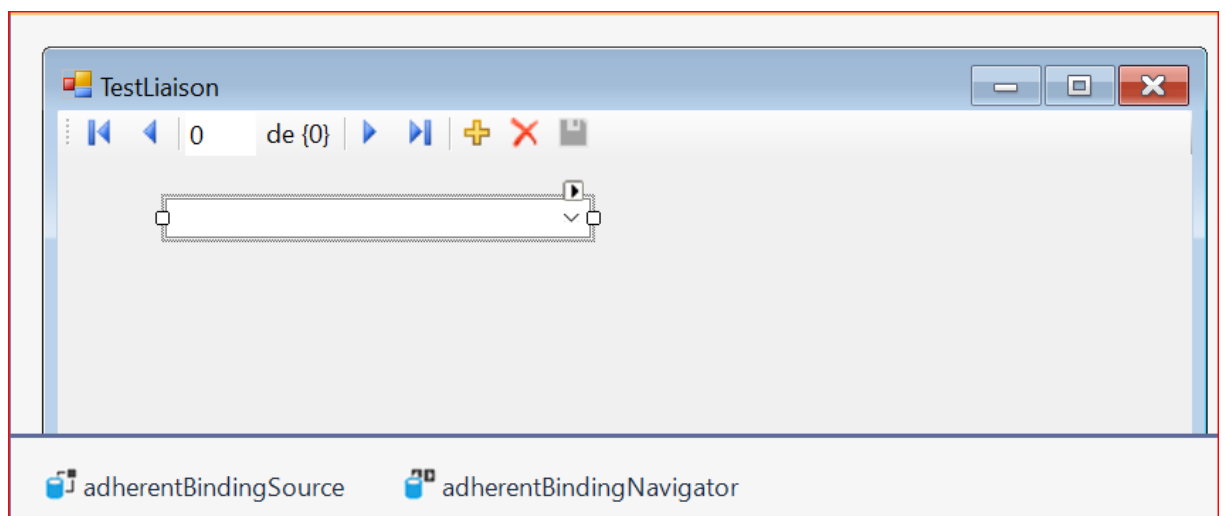
Sélection du type Adherent et choix, dans la liste déroulante, du contrôle lié généré.



Glissons déposons l'élément sélectionné sur le formulaire.

Nous avons maintenant 3 composants sur le formulaire :

- Une liste déroulante
- Une barre de navigation associé à un composant de navigation
- Un composant source de données `adherentBindingSource`



4.3. Alimentation des contrôles

Les contrôles seront alimentés via la source de données (DataSource), propriété du BindingSource. Observons les propriétés du composant BindingSource :

(Name)	adherentBindingSource
GenerateMember	True
Modifiers	Private
<input type="checkbox"/> Données	
<input type="checkbox"/> (ApplicationSettings)	
DataMember	
DataSource	Bibliotheque.BOL.Adherent

Les sources de données peuvent être de différentes nature.

Ici, la source de données est de type Adherent.

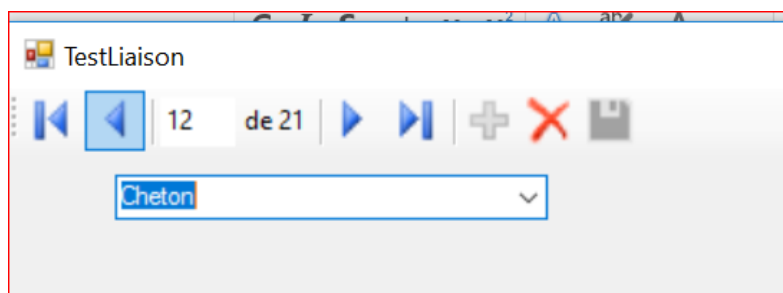
C'est à cette propriété DataSource que nous devons assigner l'objet réel qui contient les données lors de l'exécution. Ici, il s'agira d'un objet de type HashSet (mais ce pourrait être tout type Enumerable tel qu'une collection, une liste,)

```

public TestLiaison()
{
    InitializeComponent();
    adherentBindingSource.DataSource =
        AdherentDAO.Instance.GetAll();
}

```

Observons le résultat :



L'élément courant sélectionné dans la combo est synchro avec la navigation.

Supprimons l'objet Navigation et ajoutons les contrôles de formulaire permettant de maintenir les données de l'adhérent.

Choix de l'option **Détails** dans la liste déroulante et dépôt sur le formulaire

Les contrôles de détail et la liste déroulante sont alimentés par la même source de données définis au niveau du composant BindingSource.

Nouveau test concluant. Les données sont automatiquement synchronisées.

4.4. Propriétés des contrôles liés

Au niveau de la liste déroulante, catégorie **DataBindings**

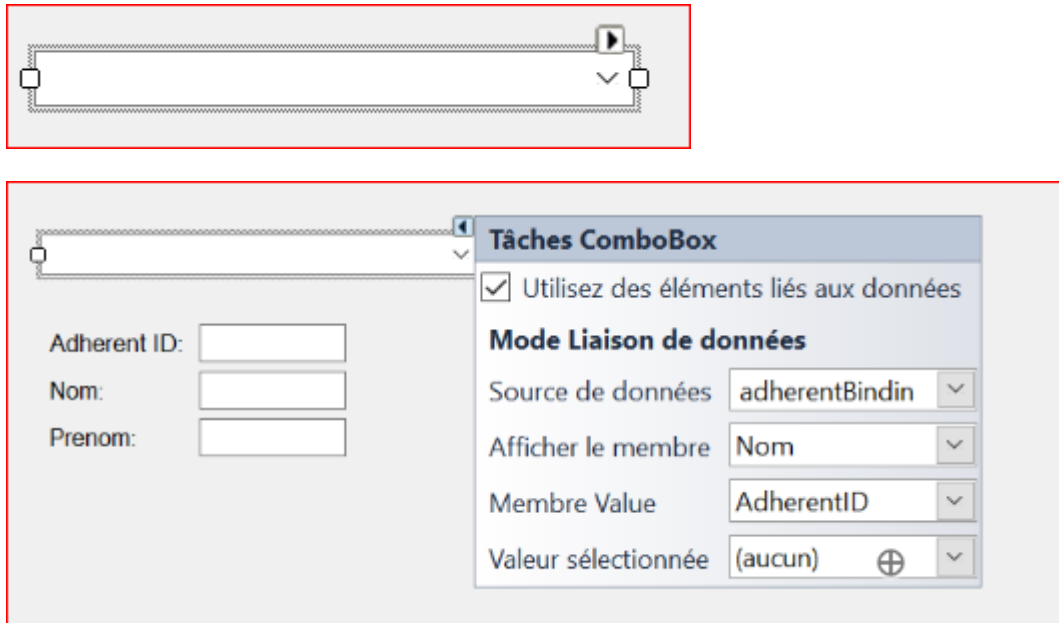
adherentComboBox System.Windows.Forms.ComboBox	
SelectedValue	(aucun)
Tag	(aucun)
Text	(aucun)
DataSource	adherentBindingSource
DisplayMember	Nom
Items	(Collection)
Tag	
ValueMember	AdherentID

Deux éléments clés :

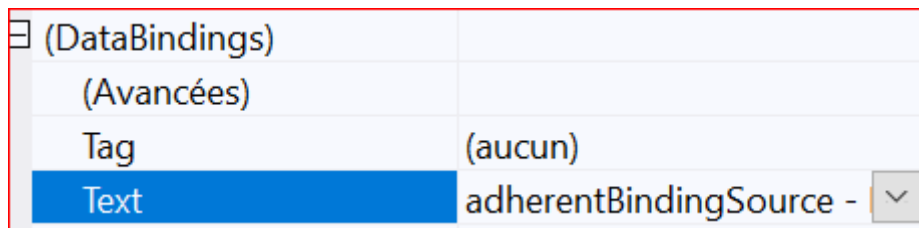
DisplayMember qui désigne l'attribut affiché

ValueMember qui désigne l'attribut sélectionné
SelectedValue

Ces attributs sont modifiables en cliquant sur la flèche qui apparaît lors de la sélection du contrôle en Design Time :



Au niveau des liaisons simples (Une propriété du contrôle liée à un attribut de l'objet métier) choix à partir de la liste accessible depuis la propriété du contrôle lié :



4.5. Récupérer la référence de l'objet lié

Une fois les mécanismes de liaison de données définis, observons comment manipuler les données au travers de nos gestionnaires de liaison.

Nous récupérons la référence à un objet lié par le biais de la propriété **Current** de l'objet **BindingSource**

Cette propriété est toujours de type Object et doit être convertie dans le type cible ad hoc. Dans notre exemple, l'objet référencé par **Current** est de type **Adherent**. Ainsi, pour afficher le nom de l'adhérent :

```
private void button1_Click(object sender, EventArgs e)
{
    Adherent adherent =
        adherentBindingSource.Current as Adherent;
    MessageBox.Show(adherent.Nom);
}
```

Une fois la référence à l'entité métier obtenue, je pourrai alors alimenter une opération de persistance :

```
AdherentDAO.Instance.Update(adherent);
```

Pour éviter de nombreuses difficultés de mise au point et conserver au maximum les bénéfices de l'indirection, il est nécessaire de sélectionner, créer, modifier ou supprimer les données en se référant à l'objet BindingSource.

Le composant bindingSource offre l'avantage de disposer d'un mécanisme de navigation et d'identification de l'élément courant.

Il assure une synchronisation correcte des propriétés des contrôles.

Les autres manœuvres sont susceptibles de poser de nombreux problèmes de synchronisation des contrôles.

Important : Ainsi, pour ajouter une nouvelle entité, nous passerons toujours par le bindingsource et sa méthode Addnew.

Autre exemple en ajout de données :

Il faut être assuré que la source de liaison accepte l'ajout de données.

```
private void btnNouveau_Click(object sender, EventArgs e)  
{  
    beneficiaireBindingSource.AllowNew = true;  
    beneficiaireBindingSource.AddNew();  
}
```

Matricule	TitreCivile	Prenom	Nom	DateNaissance	Age
18071006	Monsieur	Alan	Audivert	13/01/1962	56
18073270	Monsieur	Théo	Merino	13/01/1962	56
18077808	Monsieur	Florian	Carrey	13/01/1962	56
96265878	Madame	Alba			0

Nouveau

Matricule: 96265878

Titre Civile: Madame

Prenom: Alba

Nom: Trouvere

Date Naissance: mercredi 18 mai 1988

Age: 0

Vous pouvez alors observer la synchronisation des données.

Les données de la grille sont synchronisées lors de la validation de la propriété.

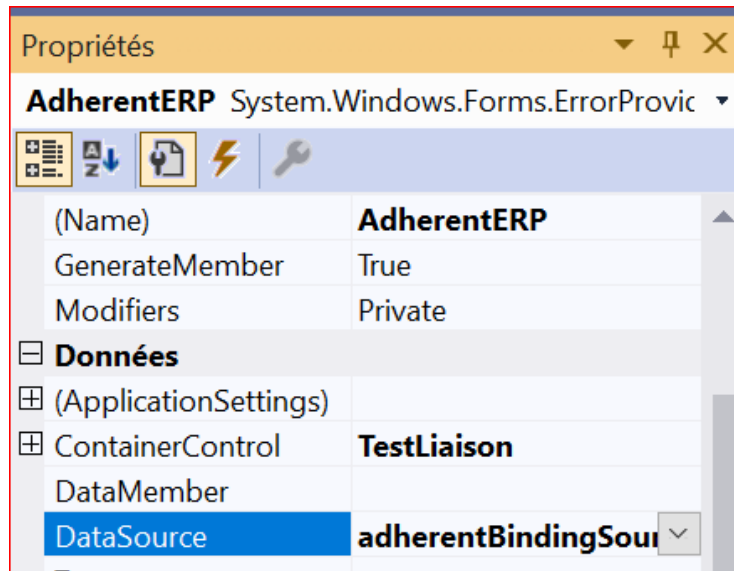
```
Beneficiaire ben = beneficiaireBindingSource.Current as Beneficiaire;
```

Vous pouvez aussi utiliser la méthode **Add** de l'objet **BindingSource** pour ajouter un objet existant.

4.6. Gestion des erreurs

Il est aisé de récupérer les erreurs via le fournisseur ErrorProvider.

Pour ce faire, il convient de lier une instance de ce composant à la source de données.



Vous aurez donc un composant de type ErrorProvider par composant de type BindingSource.

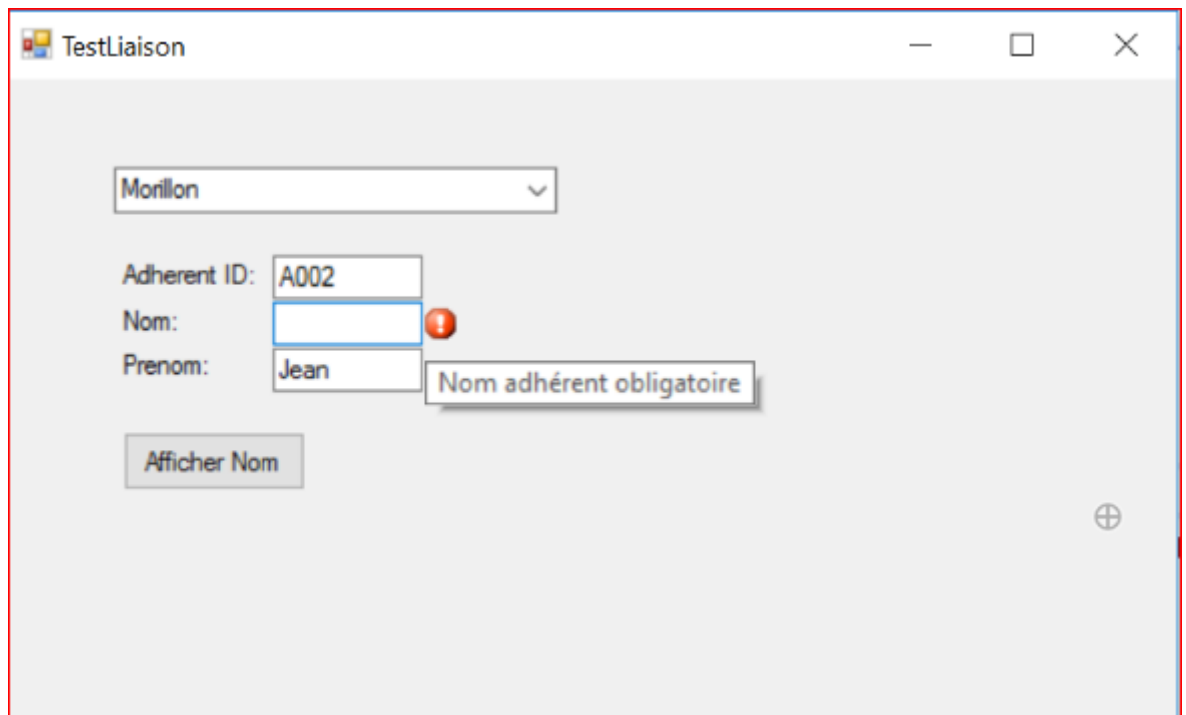
Les erreurs sont par défaut prises en compte lors de la validation du contrôle auquel est lié la propriété.

Les erreurs sont par défaut prises en compte lors de la validation du contrôle auquel est lié la propriété, qui déclenche la mise à jour de l'attribut de l'objet.

Exemple : vérification que le nom de l'adhérent soit renseigné. Dans la classe Adherent de la couche BOL, une exception est levée.

```
public string Nom
{
    get => _nom;
    set
    {
        if (string.IsNullOrEmpty(value))
        {
            throw new
                ApplicationException("Nom adhérent obligatoire");
        }
        _nom = value;
    }
}
```

Au niveau de l'interface, l'exception est récupérée et son message affiché par le composant ErrorProvider.



Fabuleux non !

4.7. Le cas de l'héritage et des spécialisations

Lorsque nous affectons au composant BindingSource une collection ou liste générique en source de données, cette source de données sera du type du premier élément ajouté au Bindingsource.

La plupart du temps, cela ne pose aucun problème.

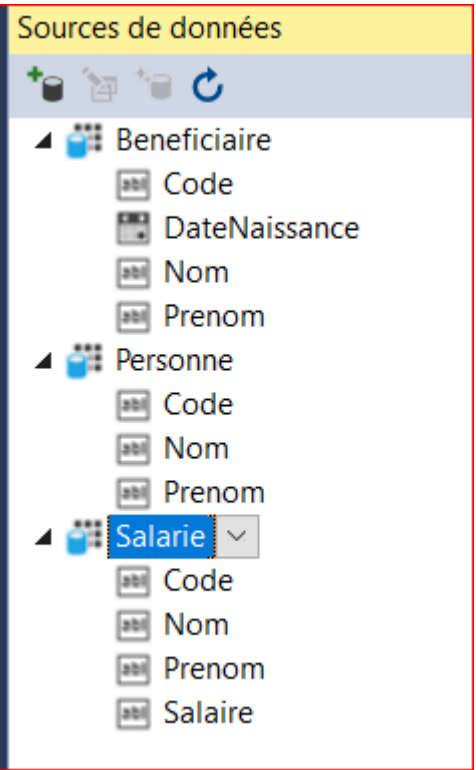
Mais, parfois, notre source de données doit comporter des objets de différents types héritant d'un type commun.

C'est le cas dans l'exemple qui suit où nous avons une liste déclarée du type générique Personne (abstrait).

Elle comportera des objets qui héritent du type Personne spécialisés en Bénéficiaire ou Salarié.

Cet exemple illustre la création d'une liste de personnes constituée de l'union d'un ensemble de bénéficiaires et de salariés.

Définition de la liaison de données



La source de données désignera donc des objets du type parent `Personne`.

Design	
(Name)	personneBindingSource
GenerateMember	True
Modifiers	Private
Données	
(ApplicationSettings)	
DataMember	
DataSource	TestHashSetHeritage.Personne

Génération de la liste par la fusion de 2 HashSet.

```
1 référence
private HashSet<Personne> chargerPersonnes()
{
    // Création d'une liste de personnes --> salariés
    HashSet<Salarie> salaries = new HashSet<Salarie>
    {
        new Salarie() { Code = 123, Nom = "Bost", Prenom = "Vincent", Salaire = 4300 },
        new Salarie() { Code = 234, Nom = "Morillon", Prenom = "Jean", Salaire = 4300 },
        new Salarie() { Code = 345, Nom = "Bueno", Prenom = "Ange", Salaire = 4300 }
    };
    // Création d'une liste de personnes --> bénéficiaires
    HashSet<Beneficiaire> beneficiaires = new HashSet<Beneficiaire>
    {
        new Beneficiaire() { Code = 456, Nom = "Audivert", Prenom = "Alan", DateNaissance = new DateTime(2000, 10, 05) },
        new Beneficiaire() { Code = 567, Nom = "Carrey", Prenom = "Florian", DateNaissance = new DateTime(1999, 10, 05) },
        new Beneficiaire() { Code = 789, Nom = "Frugier", Prenom = "Nicolas", DateNaissance = new DateTime(1997, 10, 05) }
    };

    // Création d'une liste de personnes Union des deux ensembles

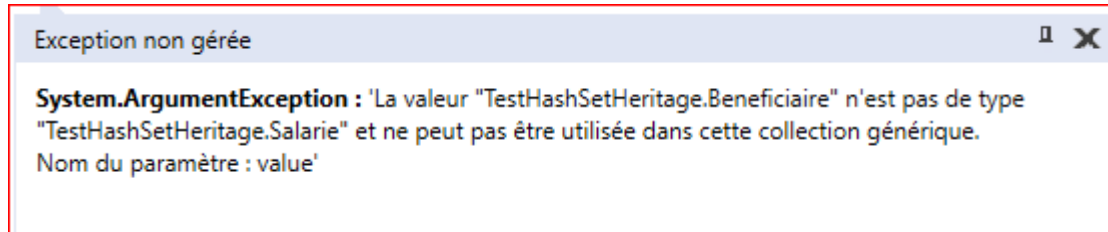
    HashSet<Personne> personnes = new HashSet<Personne>();
    personnes.UnionWith(salaries);
    personnes.UnionWith(beneficiaires);

    return personnes;
}
```

Si nous exécutons ensuite ce code :

```
personneBindingSource.DataSource = chargerPersonnes();
```

Nous obtenons une erreur d'exécution :

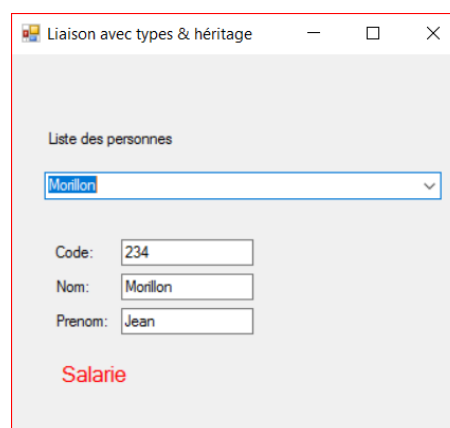
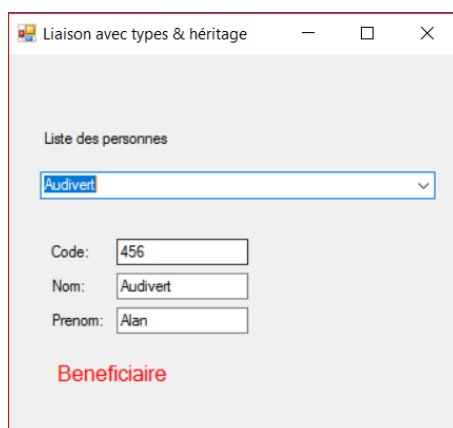


L'erreur indique que la source de données est de type Salarie. Cela correspond au premier objet de la liste. Nous pouvons résoudre ce problème en réalisant l'alimentation de la source de données par programme.

Ainsi, nous ajoutons chaque élément de la liste par le biais de la méthode Add du BindingSource.

```
foreach (var item in chargerPersonnes())  
{  
    personneBindingSource.Add(item);  
}
```

Nous avons alors au sein de la liste des salariés et des bénéficiaires.



Prenez connaissance des deux exemples fournis pour bien appréhender les mécanismes de liaison avec spécialisation et manipulation de HashSets.

5. Une liaison Bi-Directionnelle

La liaison de données est bi-directionnelle.

Nous avons pu vérifier que la modification de la propriété d'un contrôle lié mettait à jour la propriété de l'objet.

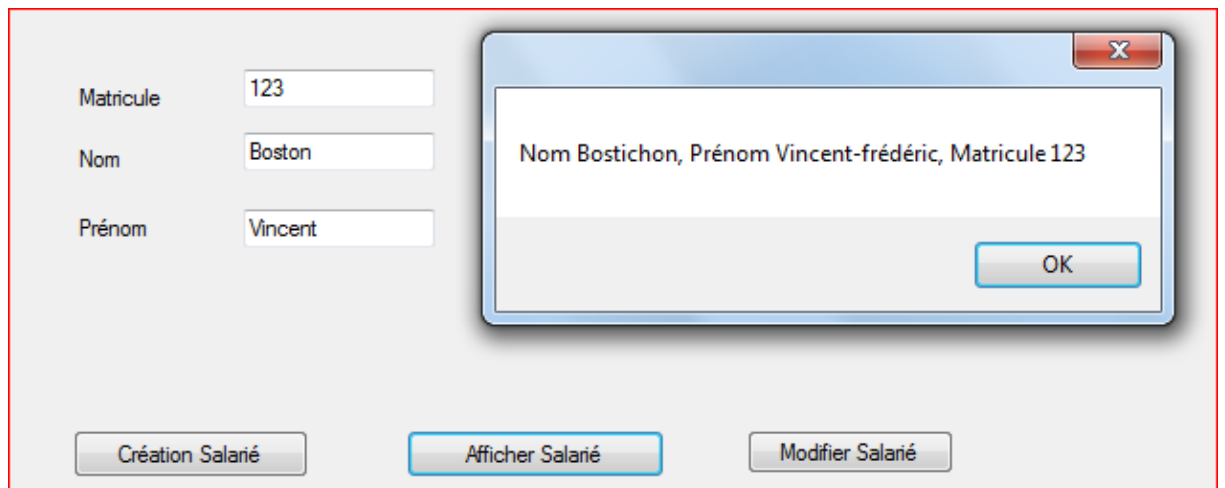
La propriété du contrôle sera-t-elle mise à jour si la propriété de l'objet est modifiée par programme.

Essayons donc maintenant de modifier cette même propriété via le programme pour voir si l'interface prend en compte cette mise à jour.

Nous ajoutons un bouton btnModifierSalarie dont le gestionnaire associé au clic est le suivant :

```
private void btnModifierSalarie_Click(object sender, EventArgs e)
{
    salarie.Nom = "Bostichon";
    salarie.Prenom = "Vincent-Frédéric";
}
```

Exécutons cette procédure et vérifions son effet au niveau de l'interface :



L'interface n'a pas été rafraîchie bien que, comme l'atteste l'affichage des propriétés le nom et le prénom aient bien été modifiés.

Pour que le contrôle soit informé de la modification nous allons devoir modifier notre type Salarie et préciser qu'en cas de modification d'une propriété, le contrôle lié doit en être informé. Il s'agit d'une autre illustration de l'utilisation des événements. Plusieurs approches sont possibles, je vous donne celle la plus élégante et la moins lourde à mettre en œuvre.

Nous allons implémenter une interface spécialisée **INotifyPropertyChanged**. Comme son nom l'indique, elle permet, via un événement, de notifier le changement d'une propriété.

Nous allons pour l'exemple l'implémenter au niveau de la modification de la propriété Nom du salarié.

5.1. Implémentation de la notification de modification de propriétés

Reprenons le projet « ComposantSalaries » réalisé lors de l'étape consacrée à l'initiation à la programmation objet.

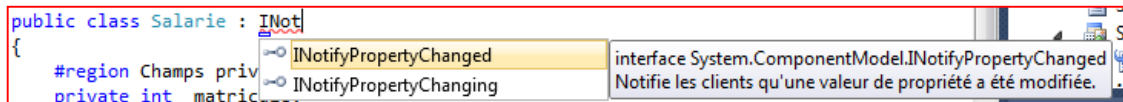
L'interface `INotifyPropertyChanged` se trouve dans l'espace de noms `System.ComponentModel`.

Première chose à réaliser donc, la mise en place d'une directive `using`.

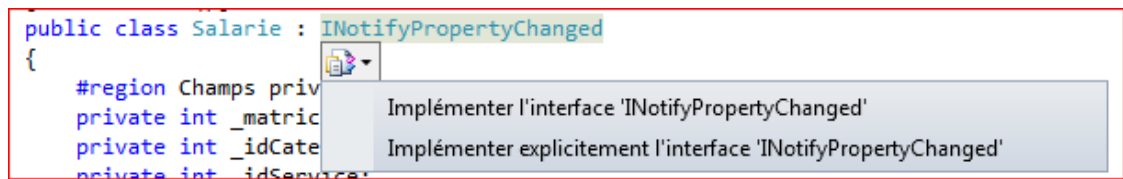
```
using System.ComponentModel;
```

Ensuite, le mécanisme de l'héritage de l'interface `INotifyPropertyChanged`

Vous remarquerez que vous avez deux interfaces qui ne sont pas sans vous rappeler des éléments déjà vus :



Son implémentation par sélection de la barre de soulignement bleu.



Regardons le code généré :

```
#region INotifyPropertyChanged Membres

public event PropertyChangedEventHandler PropertyChanged;

#endregion
```

Cette interface n'est pas autre chose qu'un délégué spécialisé dans la notification par événement des changements de valeur d'une propriété.

Mettons maintenant en place la notification par événement proprement dite.

Vous constaterez que les données d'événement sont de type **PropertyChangedEventArgs**

Il s'agit en fait d'une chaîne qui représente le nom de la propriété. Nous transmettons donc en arguments deux données, l'émetteur à l'origine de l'événement **this** correspondant à `sender` et **PropertyChangedEventArgs** le nom de la propriété modifiée.

```
private void NotifyPropertyChanged(String nomPropriete)
{
    // Si un abonné
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(nomPropriete));
    }
}
```

Il nous reste à mettre en place le mécanisme de notification lorsqu'une modification survient.

```
public string Nom
{
    get { return (this._nom); }
    set
    {
        controleSaisieNom(value);
        if (value != this._nom)
        {
            NotifyPropertyChanged("Nom");
        }
        this._nom = string.Format(CultureInfo.CurrentCulture, "{0}{1}", value.Trim().Substring(0, 1).ToUpper(CultureInfo.CurrentCulture),
    }
}
```

Nous contrôlons que la valeur ait bien été réellement modifiée pour ne pas émettre des événements inutiles.

Vous trouverez, dans certains exemples, la possibilité d'ignorer le nom de la propriété en recourant à un attribut, `CallerMemberName`, qui permet de récupérer le nom du membre de la méthode appelante. Dans ce contexte, cette méthode est toujours l'accessoire Set de la propriété considérée.

Vous pouvez alors écrire la méthode ainsi :

```
private void NotifyPropertyChanged([CallerMemberName] String nomPropriete = "")
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(nomPropriete));
    }
}
```

Et l'invoquer ainsi, l'argument `nomPropriete` étant optionnel :

Nous recompilons notre dll et observons le comportement de notre interface Windows suite à cette modification. Nous constatons avec joie que les valeurs de nos propriétés sont correctement actualisées au niveau des contrôles.

6. Fonctionnalités avancées

6.1. Gestion des liaisons de second niveau

Si la liaison de données fonctionne avec des objets simples, il est nécessaire de procéder à des amendements par code pour des problématiques plus complexes.

Il s'agit notamment de la liaison avec des **propriétés de navigation** ou des **propriétés complexes intégrées**.

Je peux prendre pour exemple de propriété de type complexe intégré au sein d'une classe un type Adresse qui serait constitué de plusieurs attributs Rue, Code postal, Ville intégré à un type Stagiaire.

Pour récupérer la valeur de la rue, il faudrait être en mesure de lier la propriété du contrôle à Adresse.Rue. Hors, la liaison de données n'est pas autorisée avec un identificateur en deux parties.

Nous allons donc devoir proposer une solution pour dépasser cette limite.

Il existe plusieurs possibilités.

J'ai choisi dans l'exemple page suivante l'option la plus simple à mettre en œuvre.

Nous retenons ici l'exemple d'une classe d'association, SuivreFormation, entre les classes Offre et Beneficiaire.

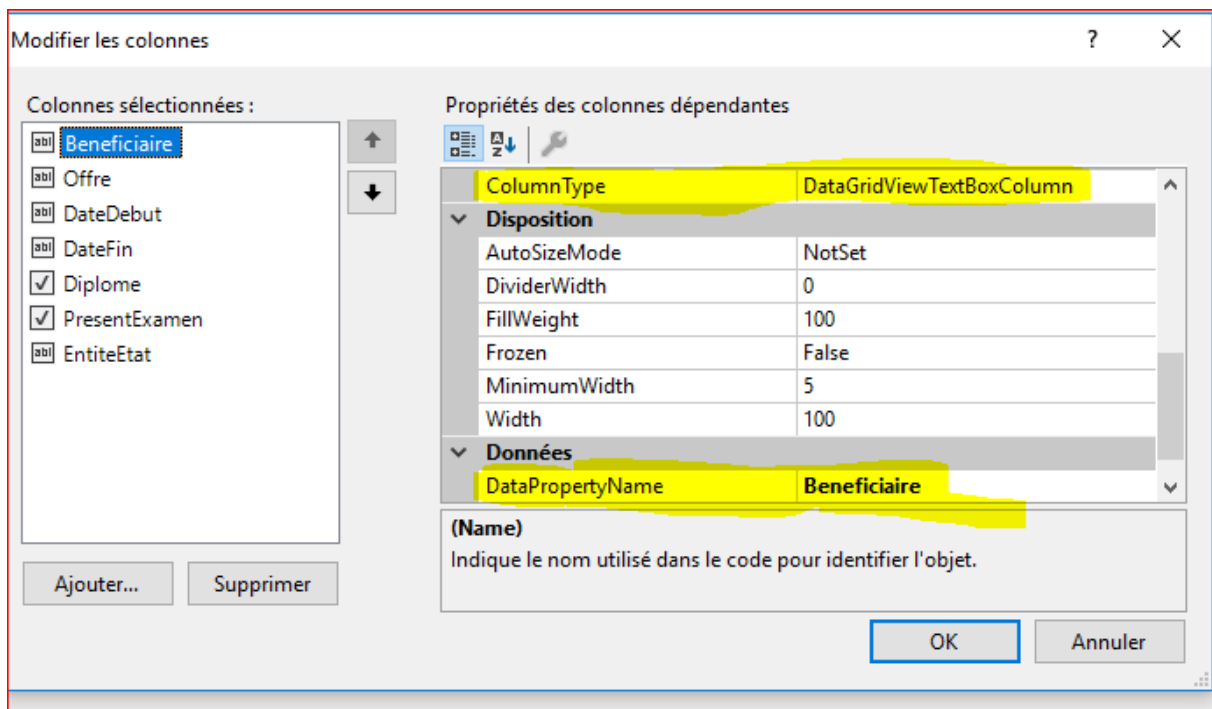
Si nous lions une grille à cette source de données du type SuivreFormation, nous obtenons une grille avec cette forme (extrait) :

	Beneficiaire	Offre	DateDebut	DateFin	Diplome	PresentExamen

Les deux premières colonnes sont liées à des objets respectivement de type Beneficiaire et Offre. A l'exécution, lors du chargement des offres suivies par un stagiaire, nous obtenons le résultat suivant :

	Beneficiaire	Offre	DateDebut	DateFin	Diplome	PresentExame
▶	11167205;Frugier;Ni...	18155 ;Concepte...	02/07/2018		<input type="checkbox"/>	<input type="checkbox"/>
<						>

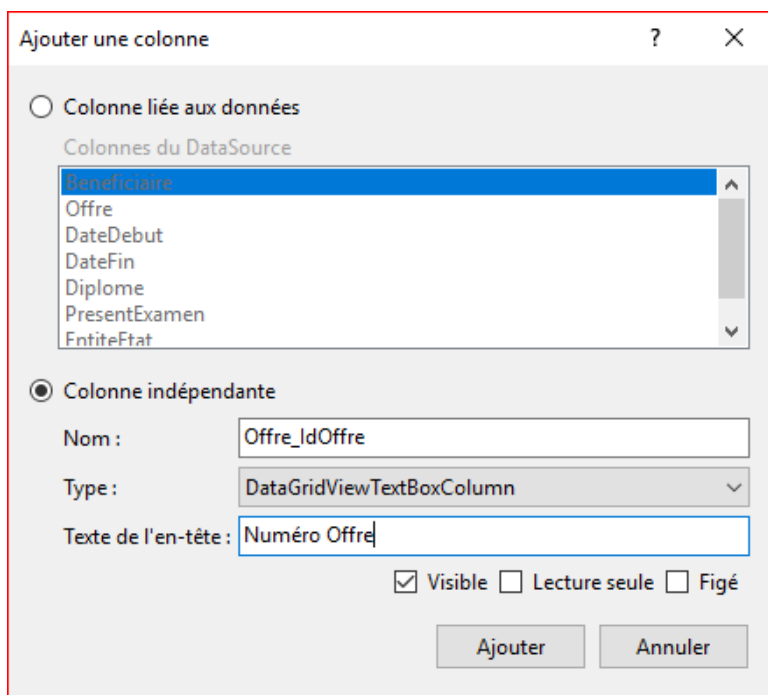
Les colonnes liées à Offre et Beneficiaire produisent l'exécution de la méthode ToString de ces objets (logique car c'est le mécanisme de conversion par défaut pour assigner des valeurs aux propriétés Text des TextBox).



Nous allons ici supprimer la liaison de données pour ces colonnes car le problème ne saurait être résolu en utilisant un identificateur en deux parties comme Beneficiaire.Nom, ...

Nous supprimons les 2 colonnes liées et ajoutons deux colonnes indépendantes qui représenteront respectivement le code et le libellé de l'offre de formation.

Inutile d'afficher les infos du stagiaire qui sont déjà affichées à l'écran.



Ces colonnes n'étant plus liées, il convient de les alimenter par code....

Nous allons intervenir lors de l'événement de mise en forme de la cellule.

Cet événement est très utile et fait l'objet de sollicitations diverses pour appliquer des formats de présentation, réaliser des conversions, ...

Cet événement transporte des données.

Nous allons utiliser 3 d'entre elles pour notre fonctionnalité.

- l'index de la colonne traitée
- l'index de la ligne de la grille traitée
- l'objet lié par la source de données à la grille

Gestionnaire d'événement mis en œuvre pour s'exécuter lors du formatage de la cellule.

J'utilise ici des variables intermédiaires pour stocker les références des objets manipulés afin de permettre une meilleure compréhension du code.

Un test est effectué sur le nom de la colonne. La valeur de la propriété de l'objet lié est alors affectée à la cellule en fonction du résultat du test. Nous en profitons pour indiquer à l'utilisateur si la formation est terminée ou non.

```
private void DGVSuivreFormations_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    DataGridView grille = sender as DataGridView;

    DataGridViewColumn colonne = grille.Columns[e.ColumnIndex];

    SuivreFormation objetLie = grille.Rows[e.RowIndex].DataBoundItem as SuivreFormation;

    if (objetLie != null)
    {
        if (colonne.Name == "Offre_IdOffre")
        {
            e.Value = objetLie.Offre.IdOffre;
            if (objetLie.Offre.DateFin < DateTime.Now)
            {
                e.CellStyle.BackColor = Color.Red;
                e.CellStyle.SelectionBackColor = Color.Red;
            }
            else
            {
                e.CellStyle.BackColor = Color.Green;
                e.CellStyle.SelectionBackColor = Color.Green;
            }
        }
        else if (colonne.Name == "Offre_Libelle")
        {
            e.Value = objetLie.Offre.Libelle;
        }
    }
}
```

	Numéro Offre	Désignation	DateDebut	DateFin	Diplome	PresentExamen
▶	18155	Concepteur Dév...	02/07/2018		<input type="checkbox"/>	<input type="checkbox"/>
<						>

6.2. Valider ou invalider les modifications

Il est possible de disposer de mécanismes permettant d'abandonner les modifications réalisées sur un objet. Ainsi, vous pouvez mettre en place un processus de validation ou annulation des modifications en cours via la mise en œuvre de 2 boutons, annuler et valider,

Le bindingsource dispose de 2 méthodes :

- EndEdit : elle marque la validation des modifications effectuées
- CancelEdit : elle précise que les modifications en cours doivent être abandonnées/

Si ces méthodes fonctionnent avec des sources de données de type DataTable et DataRow qui gèrent les différentes versions d'une ligne (ancienne et nouvelle pour faire simple), avec une source de données de type objet ou collection d'objets vous devez programmer les mécanismes sous-jacents.

Pour cela, les concepteurs de .Net ont développé une Interface que vous devrez implémenter dans tout type proposant un mécanisme d'invalidation des modifications en cours.

Cette interface se nomme IEditableObject

Elle prévoit l'implémentation de 3 méthodes :

```
public void BeginEdit()
{
    throw new NotImplementedException();
}

public void EndEdit()
{
    throw new NotImplementedException();
}

public void CancelEdit()
{
    throw new NotImplementedException();
}

#endregion
```

La méthode BeginEdit doit créer un objet de sauvegarde des données : Backup à programmer selon une méthode similaire au constructeur de copie : création d'un clone. Il faut penser à sauvegarder l'état de l'objet (EntityState)

La méthode EndEdit permet de valider les modifs : destruction de la sauvegarde et appel de la méthode EndEdit de l'objet de liaison. Il est possible d'envisager la modification de l'état à ce moment précis mais il faut être capable de déterminer si un changement a eu lieu entre l'appel de cette méthode et le BeginEdit.

La méthode CancelEdit doit restaurer les données sauvegardées du clone au niveau de l'objet manipulé. Il faut remettre l'objet dans son état initial et appeler la méthode CancelEdit de l'objet de liaison.

Ce sont ces méthodes qui peuvent être à l'origine du déclenchement ou non de la mise à jour de la base de données.

Vous trouverez un exemple illustrant la mise en place de ce mécanisme à l'adresse suivante : <https://docs.microsoft.com/fr-fr/dotnet/api/system.componentmodel.ieditableobject?view=netframework-4.7.2>

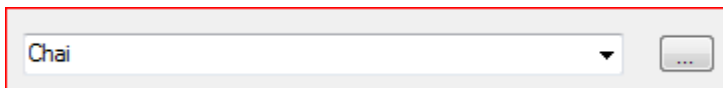
6.3. Partager une même source de liaison entre 2 fenêtres

Il est possible de partager une source de données entre un formulaire comportant une grille de données ou une liste déroulante et un formulaire de détail. Ce n'est toutefois pas sans poser de problèmes.

Pour cela, vous allez passer la référence de la source de liaison au constructeur du formulaire de détail.

Dans l'exemple qui suit, une demande d'ajout d'un nouveau produit doit être traitée sur click du bouton à droite de la liste déroulante.

Cette liste appartient au formulaire « Maître » ou « Parent ».



```
private void btnNouveauProduit_Click(object sender, EventArgs e)
{
    productsBindingSource.AddNew();
    dialogueProduit(this.productsBindingSource);
}
```

Nous invoquons le constructeur du formulaire « Esclave » ou « Enfant » en lui communiquant la **référence** de la source de données et commençons l'édition de la ligne courante (ici, le nouvel élément ajouté).

```
private void dialogueProduit(BindingSource productBS)
{
    FrmProducts dialProduct = new FrmProducts(productBS);
    if (dialProduct.ShowDialog() == System.Windows.Forms.DialogResult.Cancel)
        productBS.CancelEdit();
    else
        productBS.EndEdit();
}
```


Dans la fenêtre enfant, nous allons procéder à la liaison des contrôles avec la source de données partagée. La liaison doit se faire alors par code.

Avec le concepteur visuel, les éléments de liaison doivent être déposés sur le formulaire....

```
public FrmProducts(BindingSource productsbs):this()
{
    this.productIDTextBox.DataBindings.Add(new System.Windows.Forms.
        Binding("Text", productsbs, "ProductID", true));
    this.productNameTextBox.DataBindings.Add(new System.Windows.Forms.
        Binding("Text", productsbs, "ProductName", true));
}
```

A noter pour rappel : appel du constructeur par défaut this() en charge de l'initialisation du formulaire depuis un constructeur surchargé. Vous pouvez générer les contrôles en les déposant depuis la source de données et supprimer cette-ci ensuite.

Cette approche permet de conserver synchronisée la boîte de liste de la fenêtre maître.