Manipuler des données de fichier

Sommaire

1 Introduction

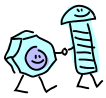
1.1 Contexte fonctionnel

Rappel du contexte fonctionnel du tutorial du coach VB

L'objectif du tutorial du Coach VB est d'accompagner les développeurs à la découverte et la prise en main du langage Visual Basic (VB) pour la construction d'applications avec une approche orientée objet.

Pour rappel, vous pouvez repérer facilement deux caractéristiques importantes du langage à l'aide des logos suivants en marge :

Ce logo marque une fonctionnalité de VB ou de Visual Studio qui permet de développer vite (et juste ☺).

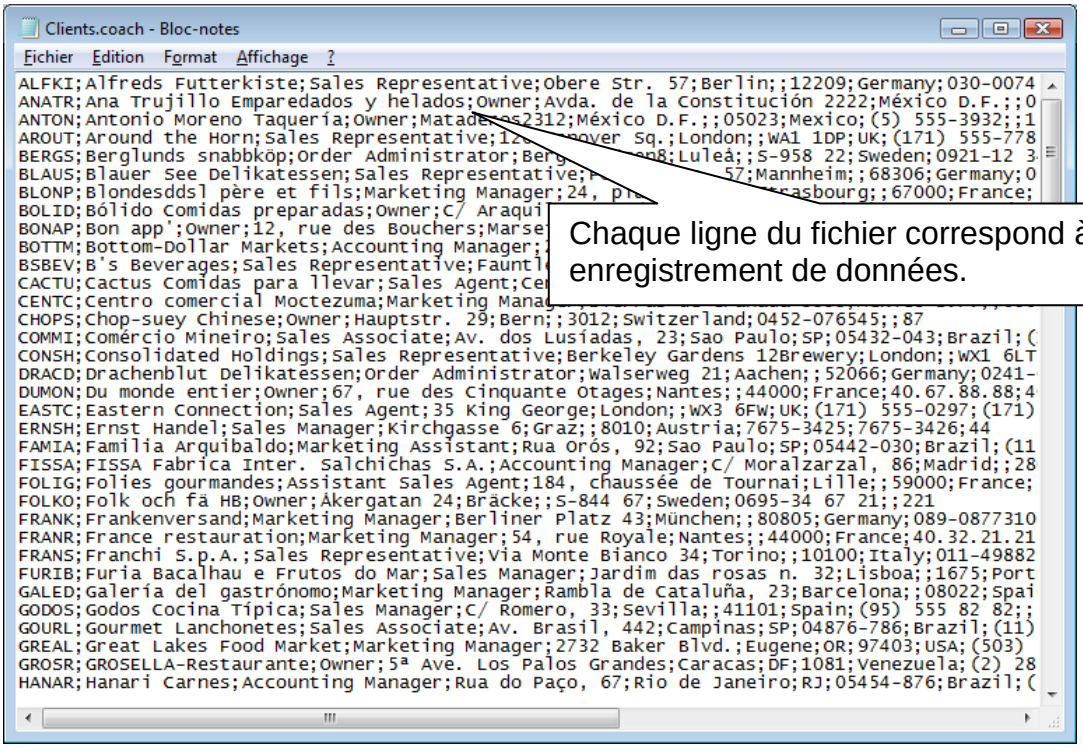


Ce logo met en évidence une caractéristique de la programmation orientée objet.

Contexte fonctionnel du quatrième atelier

Dans ce quatrième atelier, nous allons commencer à manipuler des données. Pour l'instant celles-ci sont chargées et sauvegardées sous la forme d'un fichier délimité CSV d'extension *.coach. Nous verrons dans la suite de ce tutorial comment manipuler ces mêmes données à partir d'une base de données.

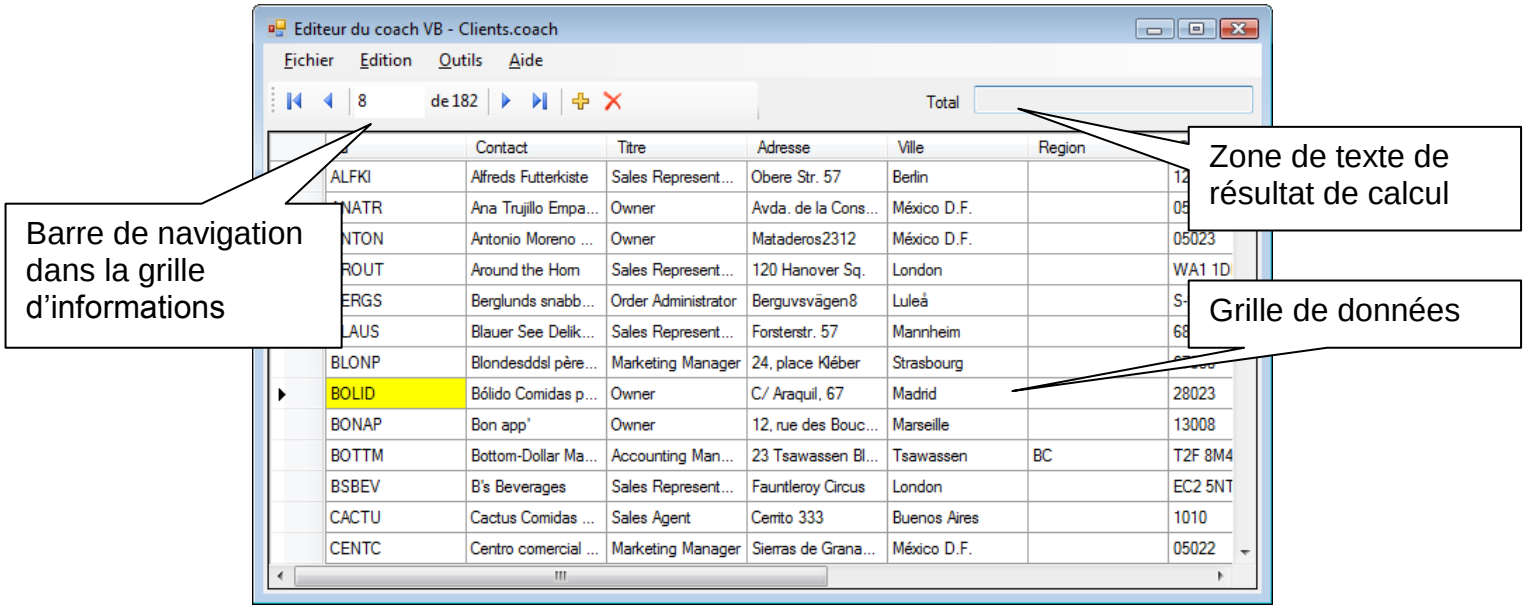
Exemple d'un fichier **Clients.coach** :



Il s'agit d'une liste de contacts client comprenant les informations suivantes :

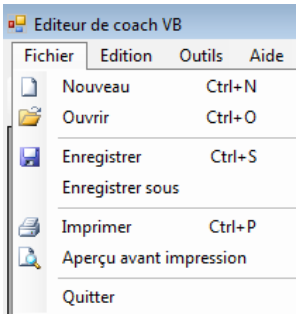
Nom	*Description*	*Type*
Id	Code d'indentification du client	string
Contact	Nom du client	string
Titre	Fonction du client	string
Adresse	Adresse du client	string
Ville	Ville de résidence du client	string
Region	Région de résidence du client	string
CodePostal	Code postal du bureau postal distributeur	string
Pays	Pays de résidence du client	string
Telephone	Numéro de téléphone du client	string
Telecopie	Numéro de la télécopie du client	string
CA	Chiffre d'affaire arrondi, en millier d'euros, que vous réalisez avec ce client	int

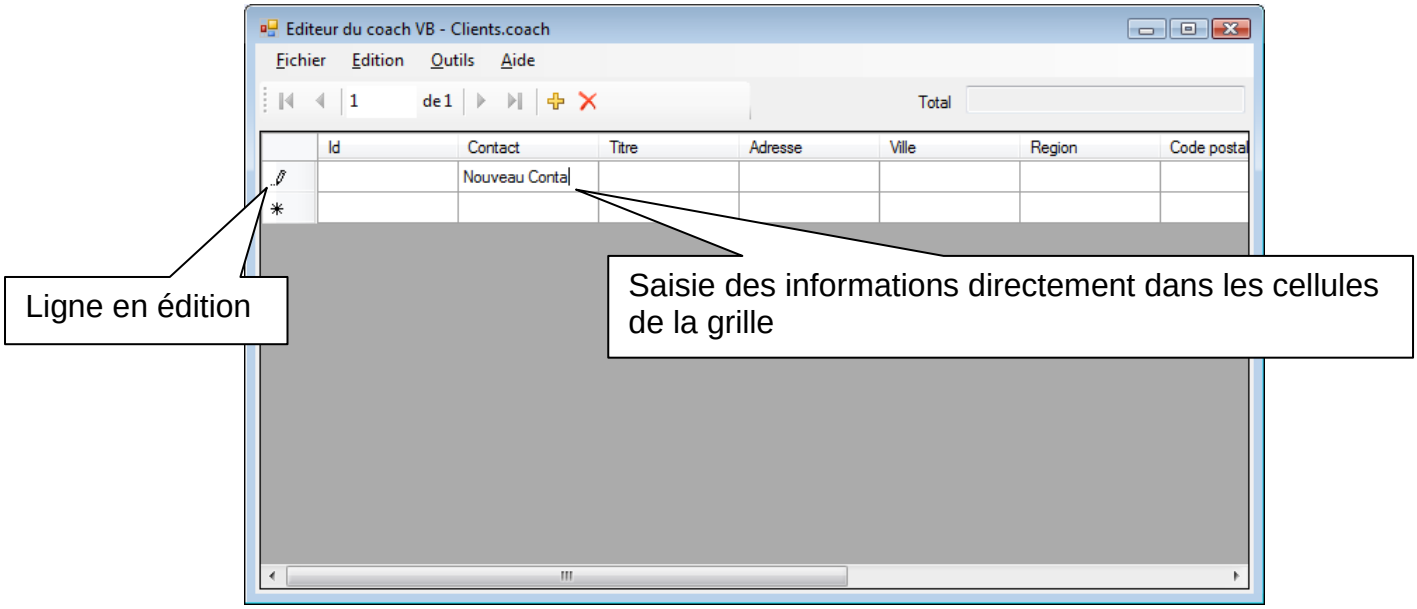
L'objectif est d'éditer les données dans notre application Editeur sous la forme d'une grille de données :



L'utilisateur doit pouvoir :

- Ouvrir un fichier existant via le menu **Fichier** > **Ouvrir** :
  - Ajouter de nouveaux contacts,
  - Supprimer et modifier des contacts existants,
  - Enregistrer ses modifications dans le fichier initial via le menu **Fichier** > **Enregistrer**
  - ou enregistrer sous un nouveau nom de fichier via le menu **Fichier** > **Enregistrer sous**,
- Ou créer un nouveau fichier via le menu **Fichier** > **Nouveau** :





Contexte technique
L'objectif de cet atelier est d'introduire les principes de gestion de données en commençant avec une approche « à l'ancienne », c'est-à-dire sans exploiter la puissance de la programmation orientée objet, et sans base de données.

A quoi bon programmer comme ça si ce n'est pas ce qu'il faut faire ?
Ne vous méprenez pas, cela va être très utile. C'est un peu comme si on allait apprendre à nager la brasse avant de se lancer à nager en brasse coulée. Nous allons travailler sur deux axes :

- Le premier concerne l'affichage des données au moyen de contrôles Windows Form adaptés. Nous verrons que l'affichage d'une source de données dans un contrôle d'affichage se fait au moyen du mécanisme de *databinding (liaison de données)*.
- Le second concerne la gestion de fichiers texte au format CSV. Nous allons apprendre à ouvrir, lire puis à écrire et enregistrer un fichier.

A la fin de cet atelier, vous saurez comment :

- Lire et écrire dans un fichier texte,
- Utiliser le databinding,
- Manipuler un tableau de données en mémoire,
- Positionner des contrôles d'affichage à l'aide de conteneur,
- Utiliser les principaux contrôles de données.

La solution de cet atelier est disponible dans le répertoire **..\Atelier 4\Solution**. Les fichiers utiles, auxquels font référence les exercices sont disponibles dans le répertoire **..Atelier 4\Fichiers utiles**.

2 Travailler avec les contrôles de données

Dans cet exercice, vous allez apprendre à :

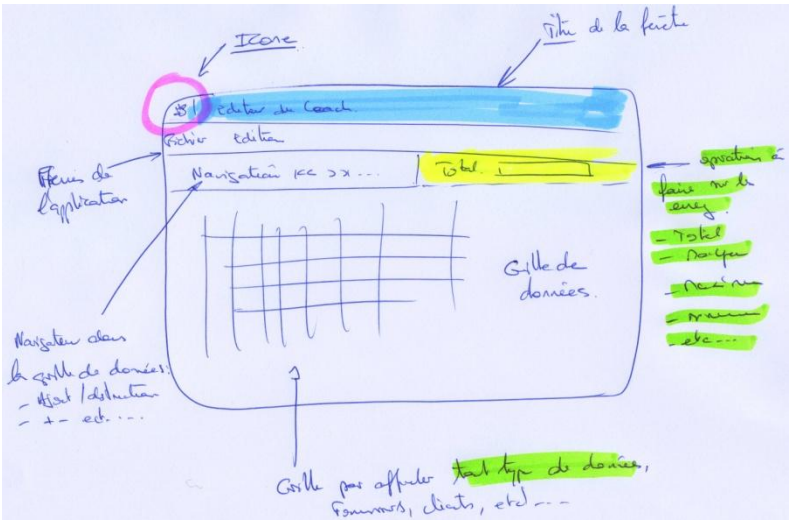
- Utiliser le contrôle conteneur TableLayoutPanel,
- Dessiner une grille de données et une barre de navigation,
- Positionner des contrôles dans un conteneur.

Objectif
L'objectif de ce premier exercice est de se familiariser avec quelques contrôles standards d'affichage et de manipulation de données.

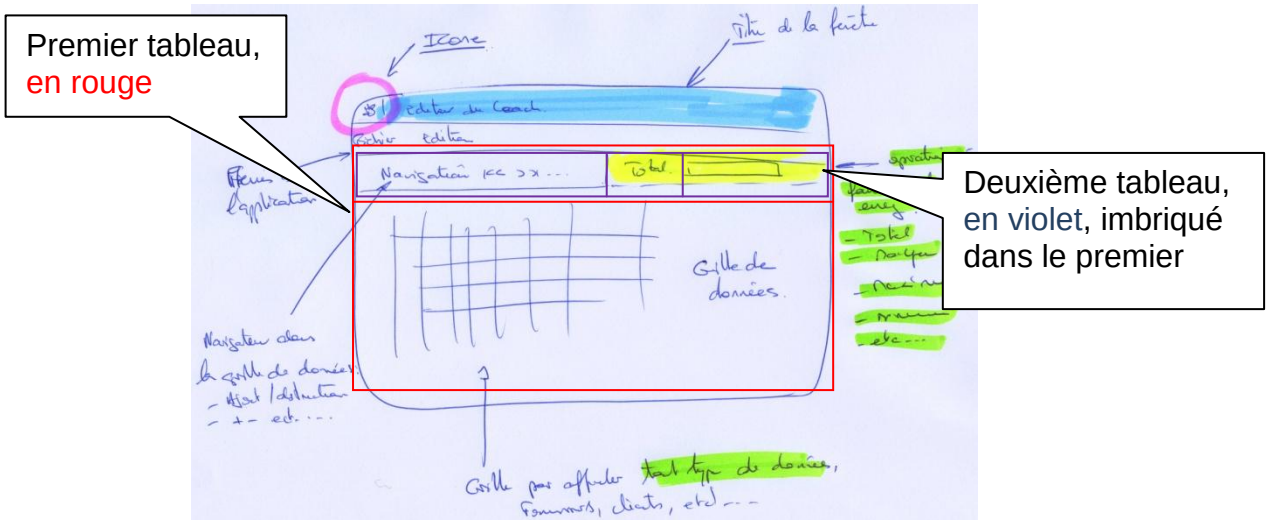
2.1 Préparer le positionnement des contrôles à l'aide d'une structure de tableau

Avant de dessiner les contrôles de données, il s'agit de les positionner. Pour cela nous allons utiliser un contrôle conteneur qui permet de structurer des contrôles sous forme de tableau.

Contexte fonctionnel
Vous vous rappelez le cahier des charges ? C'est ce petit bout de papier qui est ressorti d'une réunion d'analyse :



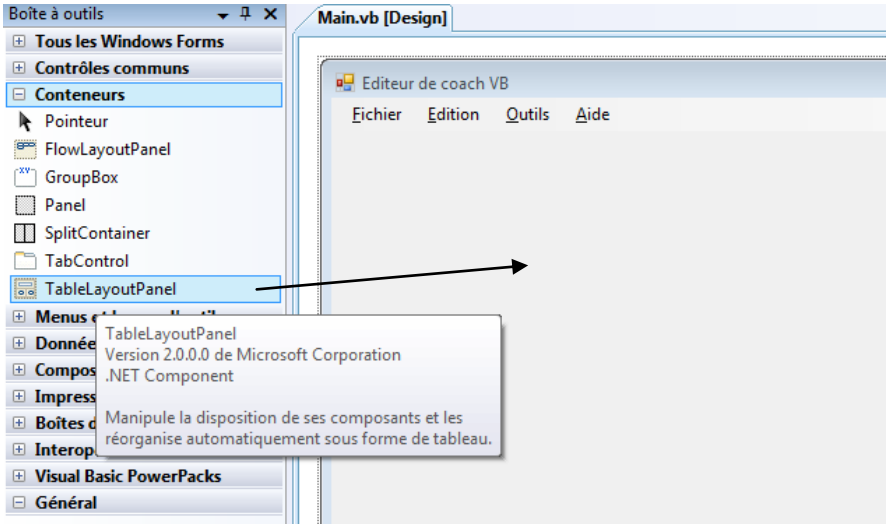
Oui, je sais…cela demande pas mal d'imagination ☹.
N'empêche qu'en y regardant bien, on peut voir que les différentes parties du formulaire sont organisées sous forme de tableaux imbriqués :



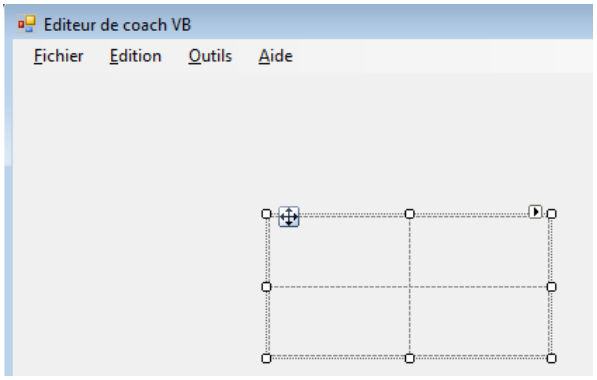
Alors pourquoi ne pas carrément dessiner des tableaux (dont les bordures seraient transparentes) pour nous aider à positionner les contrôles de manière alignée ?
Le premier tableau contiendrait une colonne et deux lignes. Le second, imbriqué dans la première ligne du précédent, comprendrait une seule ligne et trois colonnes.

Déroulement de l'exercice :

1. Ouvrez le projet précédent réalisé lors de l'atelier 3 :
   - Lancez Visual Studio à partir du menu **Démarrer** > **Tous les programmes** > **Microsoft Visual Basic 2008 Express Edition**.
   - Menu **Fichier** > **Ouvrir un projet**.
   - Retrouvez le fichier **Atelier 3.sln** que vous avez créé lors de l'atelier 3 ou, si vous n'avez pas fait l'atelier précédent, récupérez le projet de démarrage fourni avec le code de cet atelier dans le répertoire : **..\Atelier 4\Démarrage\Atelier 4.sln**.
2. Ajoutez un premier tableau (le rouge) au formulaire :
   - Ouvrez le fichier **Main.vb** en mode **Design**.
   - Faites un glisser déplacer de la **Boîte à outils** > rubrique **Conteneurs** > du contrôle **TableLayoutPanel** sur la surface centrale du formulaire.



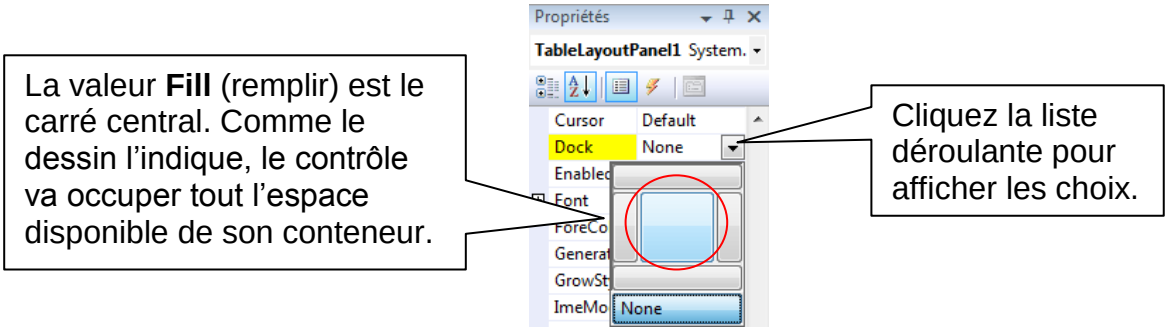
- Vous obtenez un tableau comprenant par défaut deux lignes et deux colonnes :



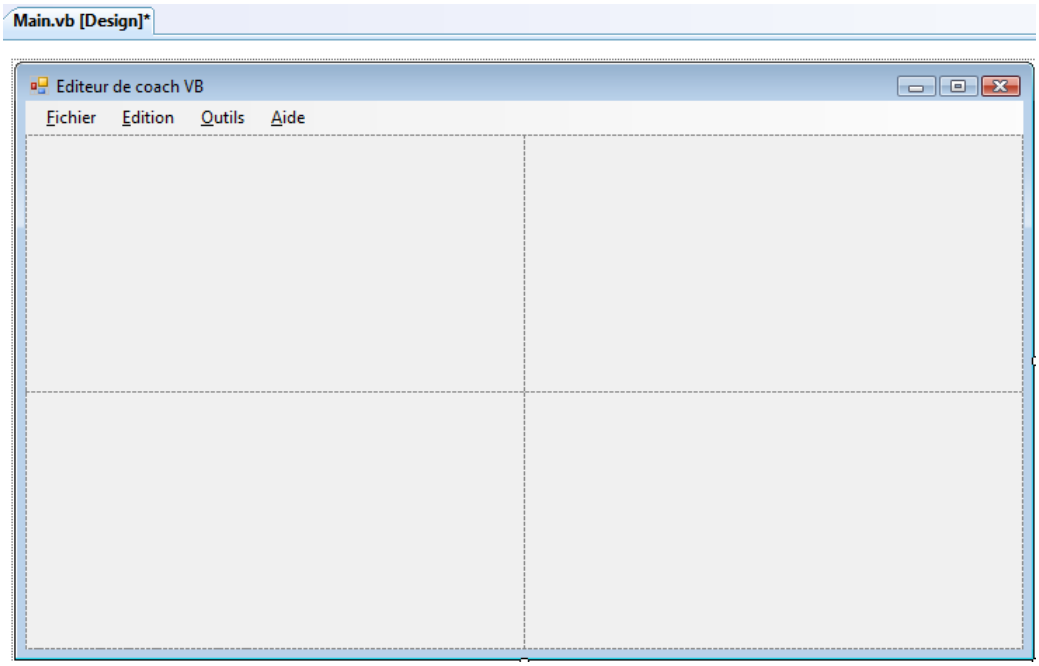
Vous pouvez déplacer le contrôle en cliquant l'icône en haut à gauche du contrôle. L'idéal serait de positionner le contrôle de façon à ce qu'il occupe toute la surface disponible du formulaire. Rappelez-vous, il s'agit du mécanisme *d'ancrage* des contrôles dont nous avons déjà parlé dans l'atelier précédent. Il se configure notamment à l'aide de la propriété **Dock** des contrôles.

- Affichez la fenêtre de **Propriétés** du contrôle (**F4**).
- Configurez la propriété **Dock** à la valeur **Fill** pour remplir toute la surface disponible du formulaire.

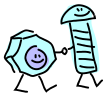
Visual Studio vous propose un éditeur visuel pour configurer cette propriété plus intuitivement :



Vous devez obtenir :



Le principal travail sur ce type de contrôles consiste à définir les lignes et les colonnes. Cela revient à configurer les propriétés **Columns** et **Rows** du contrôle **TableLayoutPanel** qui représentent respectivement une collection de colonnes et une collection de lignes.

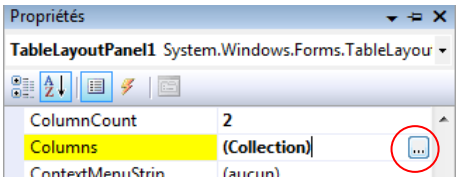


Qu'est ce qu'on entend par *collection* ?
Une collection est un objet qui sert à regrouper des objets apparentés. C'est en quelque sorte un tableau d'objets. Vous ne vous en êtes peut-être pas aperçu mais il y en a partout. Le Framework par exemple, utilise et fournit massivement des collections d'objet.

msdn

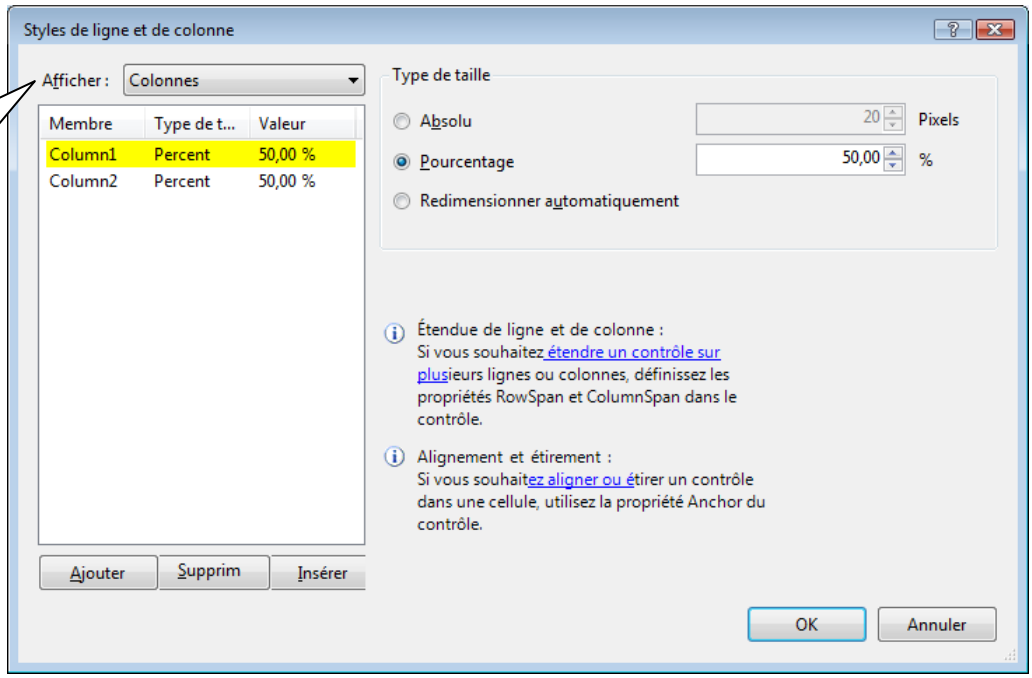
Pour tout savoir sur les collections en Visual Basic :
http://msdn.microsoft.com/fr-fr/library/a1y8b3b3(VS.80).aspx

3. Configurez le tableau pour qu'il ait deux lignes et une seule colonne :
   - Toujours dans la fenêtre de **Propriétés** du contrôle **TableLayoutPanel1**, sélectionnez la propriété **Columns** qui représente la collection de colonnes du tableau. Cliquez sur le bouton qui s'affiche en face de la propriété :

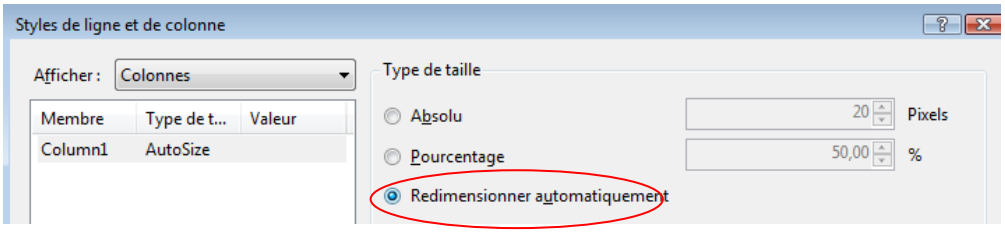


Là encore Visual Studio vous propose un éditeur pour vous assister. Il s'agit d'une boîte de dialogue commune aux propriétés **Columns** et **Rows** dans laquelle vous pouvez configurer à la fois les colonnes et les lignes du tableau. Utilisez la liste déroulante **Afficher** pour basculer des unes aux autres.

Affichage des colonnes ou des lignes

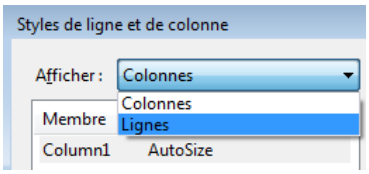


- Sélectionnez la colonne **Column2**, et cliquez le bouton **Supprimer** de l'éditeur de propriétés pour détruire la deuxième colonne qui nous est inutile.
- Sélectionnez la colonne **Column1**, et indiquez que sa dimension est automatiquement calculée en cliquant le radio-bouton **Redimensionner Automatiquement** :

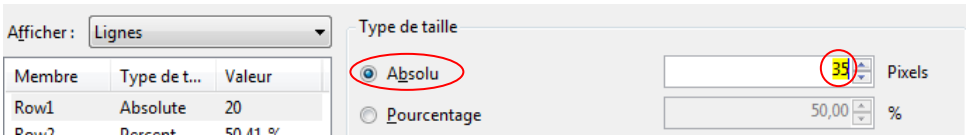


Pour voir comment fonctionne le redimensionnement automatique dans le contrôle **TableLayoutPanel** :
http://msdn.microsoft.com/fr-fr/library/ms171690.aspx

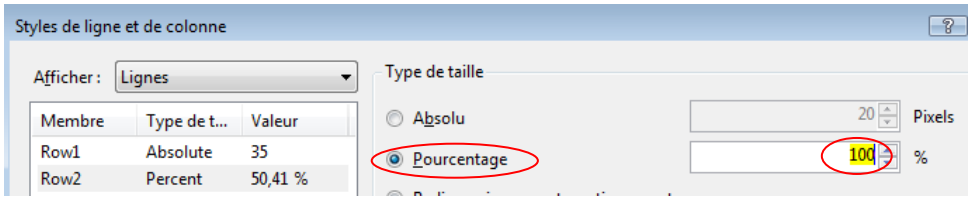
- Basculez maintenant sur la configuration des lignes en sélectionnant **Lignes** dans la liste déroulante **Afficher** :



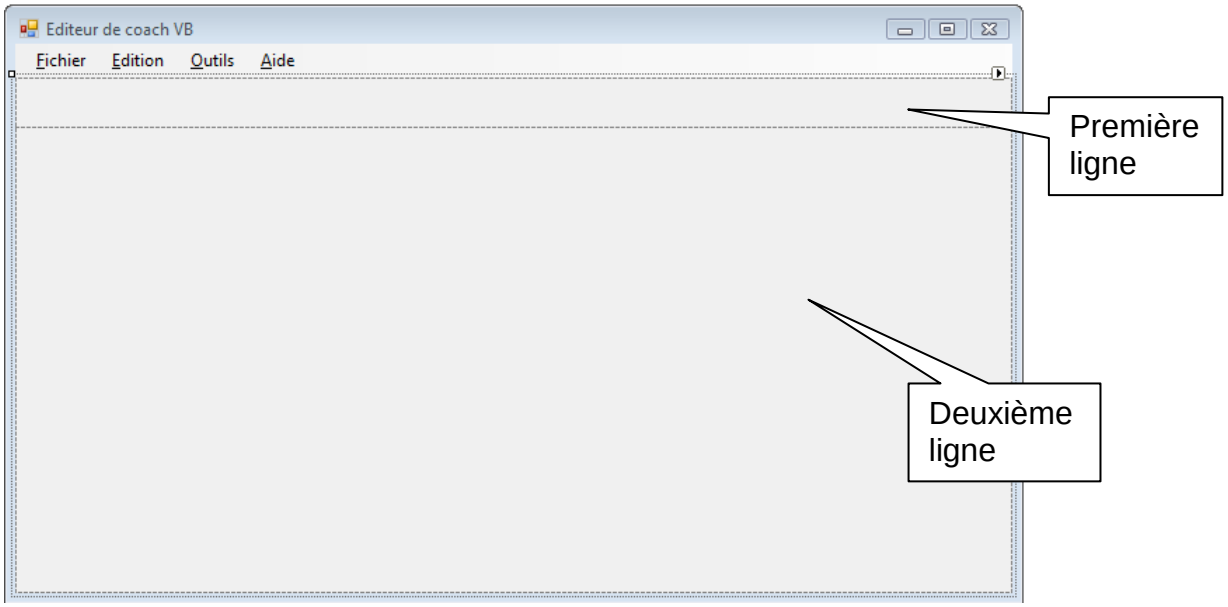
- Pour la première ligne, dont le nom est **Row1**, indiquez une taille de type **Absolu** et de **35 pixels** :



- Pour la deuxième ligne, dont le nom est **Row2**, indiquez une taille de type **Pourcentage** à **100 %**, afin d'occuper tout l'espace restant disponible :

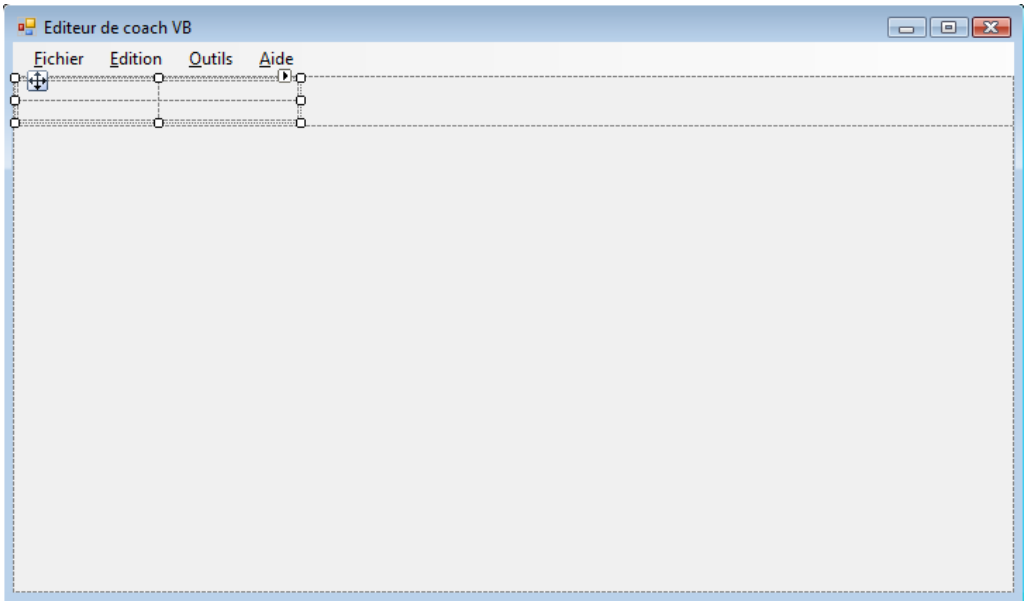


- **Cliquez** maintenant le bouton **Ok** pour fermer la boite de dialogue **Styles de ligne et de colonne**. Vous devez obtenir le tableau suivant :

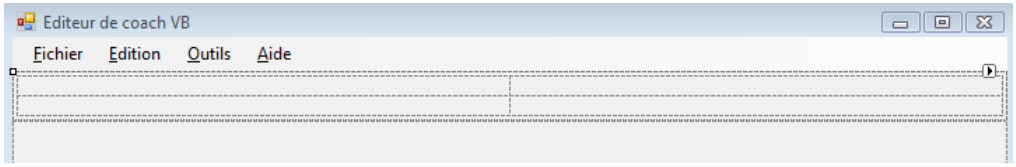


4. Ajoutez à l'intérieur de la première ligne du tableau un deuxième tableau (le violet):

- Faites un **glisser-déplacer** d'un deuxième contrôle **TableLayoutPanel** à l'intérieur de la première ligne du tableau précédent.

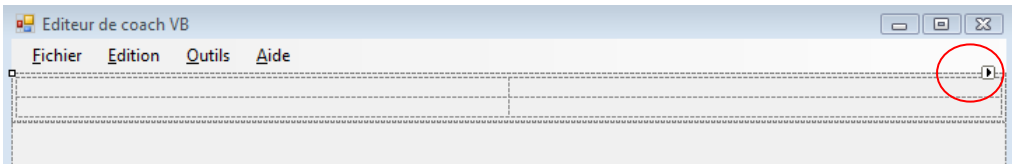


- Affichez la fenêtre de **Propriétés** du nouveau contrôle **tableLayoutPanel2** ;
- Configurez la propriété **Dock** du contrôle à **Fill** de façon à ce que le contrôle remplisse toute la surface disponible de son conteneur parent (c'est-à-dire de la ligne du premier tableau).

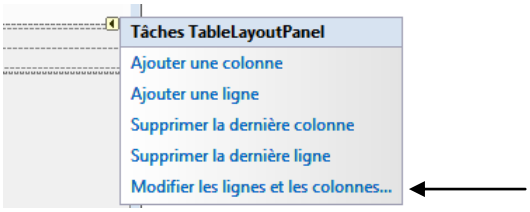


5. Configurez le tableau pour qu'il ait une ligne et trois colonnes :

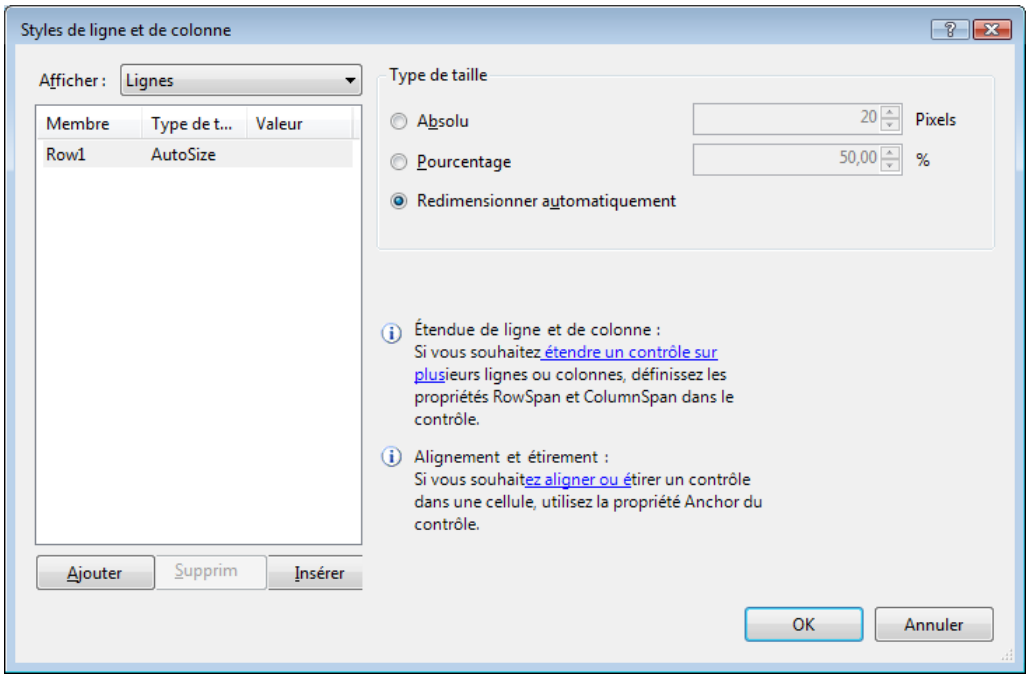
Il suffit, comme précédemment de configurer la propriété **Columns** (ou **Rows**) du contrôle pour obtenir les lignes et colonnes souhaitées.
Mais Visual Studio propose aussi un raccourci vers les principales actions possibles sur le contrôle en cliquant sa *balise active*. Il s'agit de l'icône représentant une petite flèche que l'on trouve en général en haut à droite du contrôle lorsqu'il est sélectionné.



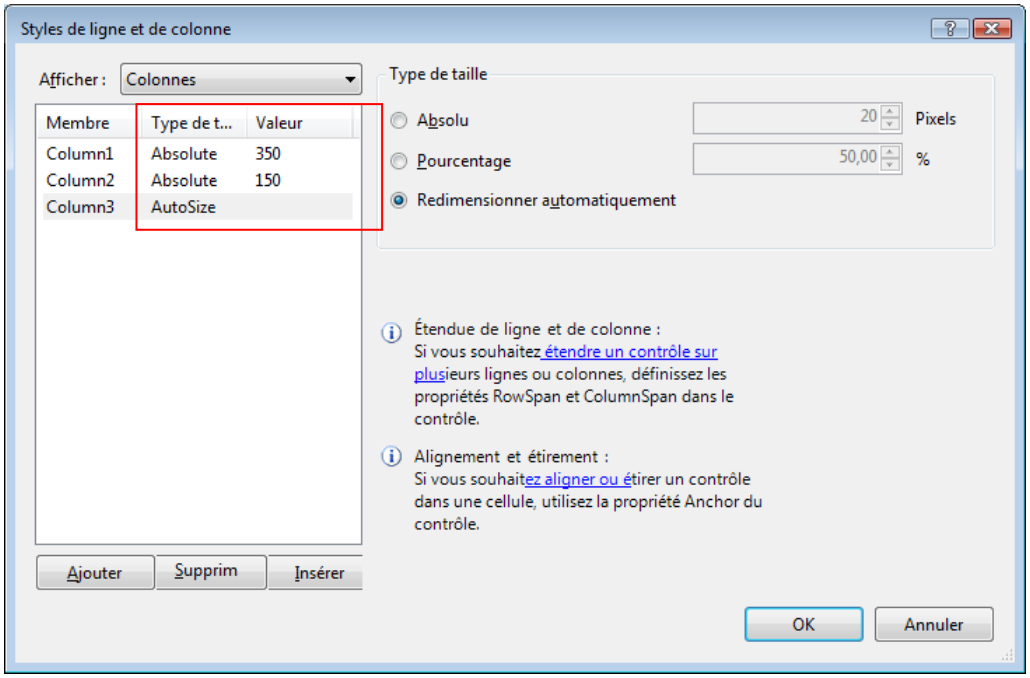
Par exemple , avec l'option **Modifier les lignes et les colonnes…**, vous retrouvez exactement la même boîte de dialogue de configuration que précédemment :



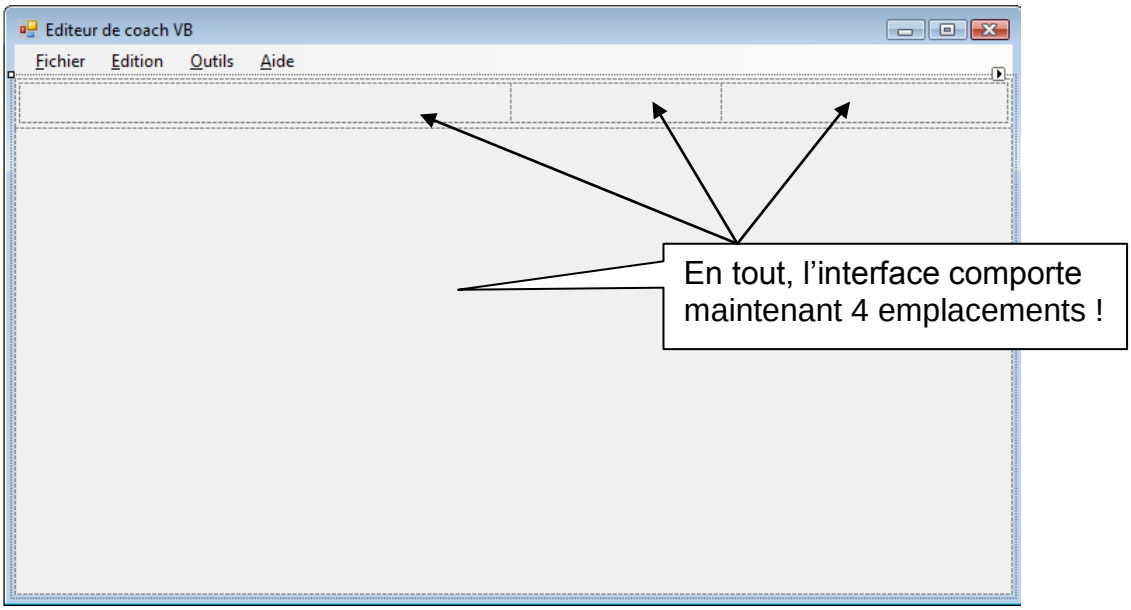
- Sélectionnez **Modifier les lignes et les colonnes…** pour afficher la boite de dialogue **Styles de ligne et de colonne**.
- Dans la liste déroulante **Afficher**, sélectionnez **Lignes**.
- Supprimez la ligne **Row2**.
- Indiquez un redimensionnement automatique pour la ligne **Row1** :



- Dans la liste déroulante **Afficher**, sélectionnez **Colonnes**.
- Ajoutez une nouvelle colonne an cliquant sur le bouton **Ajouter**.
- Indiquez une taille absolue de **350 pixels** pour la colonne **Column1**.
- Indiquez une taille absolue de **150 pixels** pour la colonne **Column2**.
- Indiquez un redimensionnement automatique pour la colonne **Column3**.



- **Cliquez** maintenant le bouton **Ok** pour fermer la boîte de dialogue **Styles de ligne et de colonne**. Vous obtenez la structure suivante :



Pour tout savoir sur le contrôle **TableLayoutPanel** :
http://msdn.microsoft.com/fr-fr/library/3a1tbfwd.aspx
Pour apprendre à disposer des contrôles dans les Windows Forms :
http://msdn.microsoft.com/fr-fr/library/ty26a068.aspx

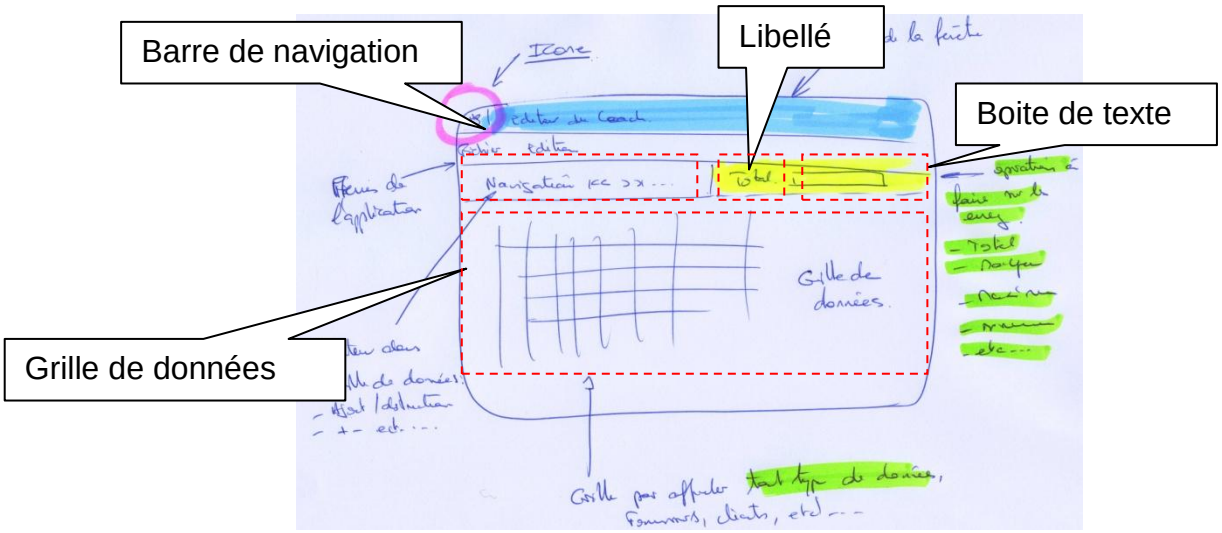
2.2 Dessiner les contrôles de données

Nous allons positionner dans les quatre cellules de tableau, quatre contrôles d'affichage que nous alimenterons avec les données d'un fichier CSV.

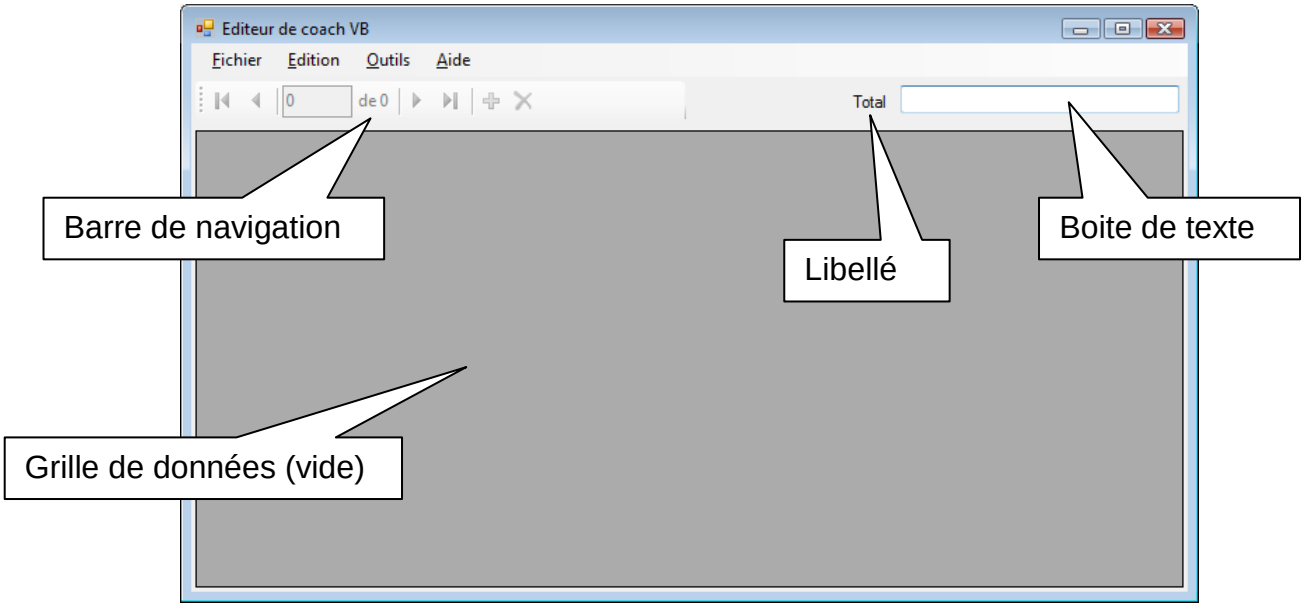
Contexte fonctionnel

Toujours en s'inspirant du document d'analyse (avec beaucoup d'imagination), il semble que l'interface utilisateur soit composée des contrôles suivants :

- Une barre de navigation,
- Une grille d'affichage des données,
- Une boite de texte d'affichage de résultat,
- Un libellé.



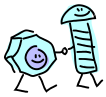
On devrait obtenir ceci :



Déroulement de l'exercice :

1. Ajoutez une grille d'affichage de données sur la deuxième ligne du tableau principal :

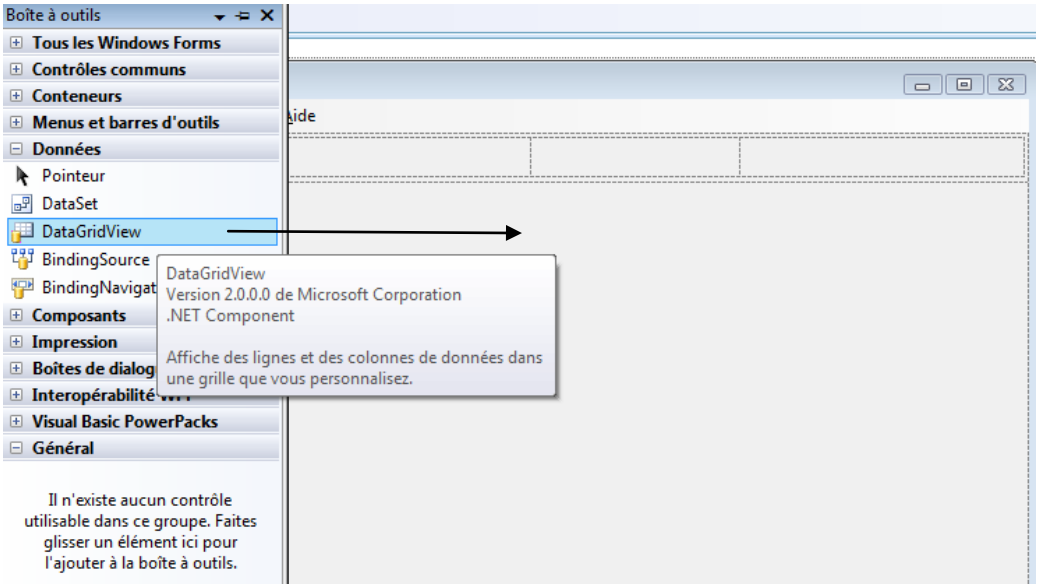
Le contrôle de grille de données standard des Windows Forms est le contrôle **DataGridView**. Il offre un moyen puissant et flexible pour afficher des données sous forme d'un tableau.



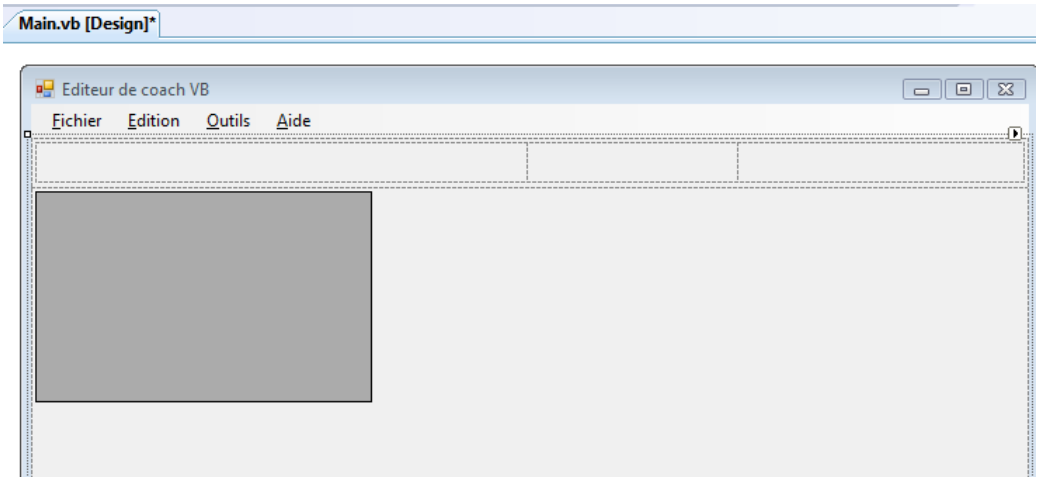
Nous verrons qu'avec les principes de programmation orienté objet tel que *l'héritage*, il est toujours possible d'étendre le comportement de base d'un objet pour le personnaliser en fonction de vos besoins.
Du coup, ne vous arrêtez pas au comportement standard d'un contrôle **DataGridView** qui ne remplit peut-être pas toutes les fonctionnalités dont vous avez besoin. Vous pouvez étendre par exemple son comportement en développant vos propres algorithmes de tri ou en confectionnant vos propres types de cellule.

Pour tout ce qui concerne l'utilisation du contrôle **DataGridView** :
http://msdn.microsoft.com/fr-fr/library/e0ywh3cz.aspx

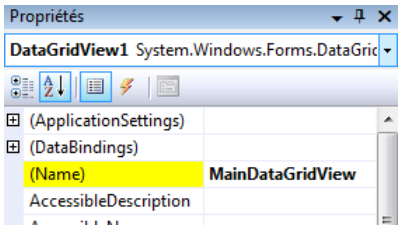
- Faites un glisser déplacer de la **Boîte à outils** > rubrique **Données** > du contrôle **DataGridView** sur la deuxième ligne du contrôle **TableLayoutPanel1** :



Vous obtenez :



- Affichez la fenêtre de **Propriétés** du contrôle.
- Configurez la propriété **(Name)** avec la valeur : **MainDataGridView**.



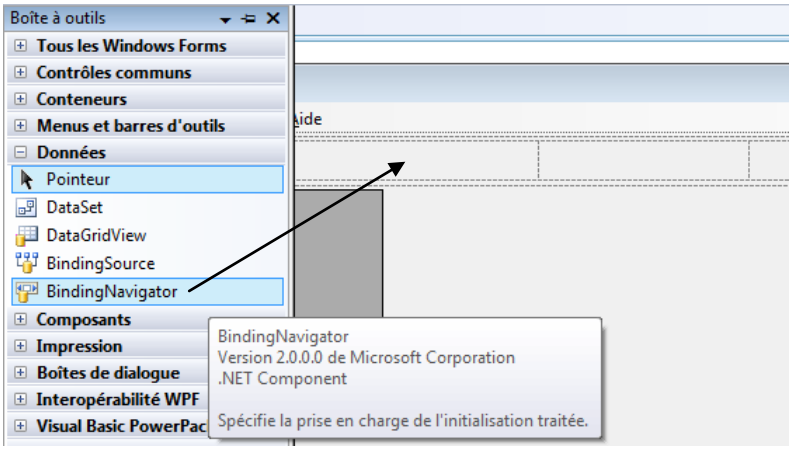
2. Ajoutez une barre de navigation de données dans la première cellule du second tableau :

Qu'est ce qu'on entend par barre de navigation ?
L'idée est d'assister l'utilisateur dans la manipulation du jeu de données qui est affiché dans l'Editeur. Comme son nom l'indique, une barre de navigation va lui permettre de *naviguer* d'un enregistrement à l'autre c'est-à-dire de faire avancer ou reculer le curseur dans la grille de données.

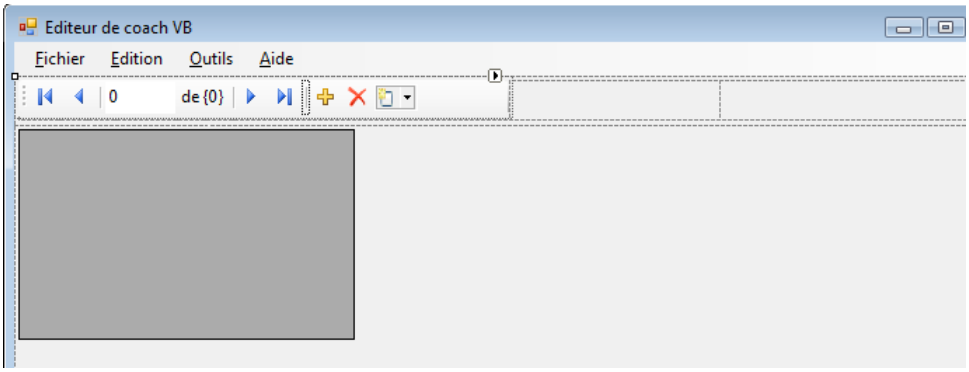
Les Windows Forms proposent un contrôle de navigation appelé **BindingNavigator**. Il permet non seulement de naviguer dans les données mais aussi d'interagir avec celles-ci. La barre de navigation peut par exemple afficher un bouton de création d'une nouvelle ligne de données, un bouton suppression etc…

Pour tout ce qui concerne l'utilisation du contrôle **BindingNavigator** :
http://msdn.microsoft.com/fr-fr/library/system.windows.forms.bindingnavigator.aspx

- Faites un glisser déplacer de la **Boîte à outils** > rubrique **Données** > du contrôle **BindingNavigator** sur la première cellule du contrôle **TableLayoutPanel2** :



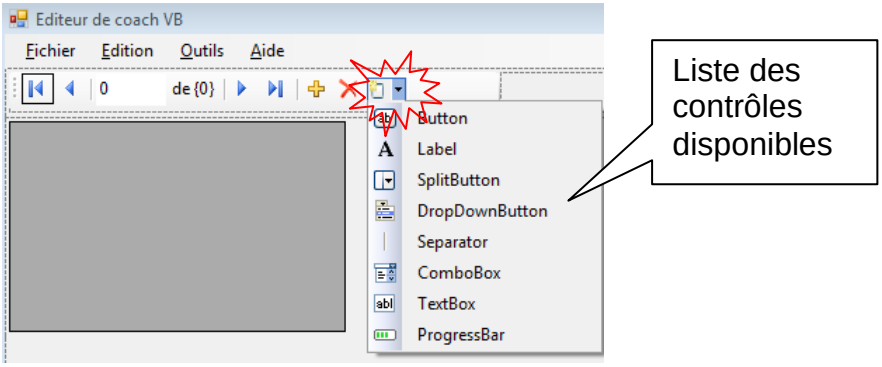
Vous obtenez :



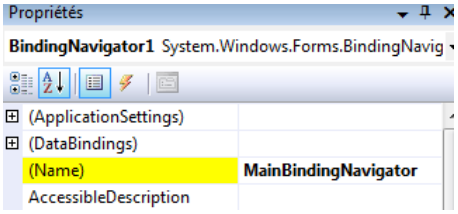
Notez les différents boutons que présente le contrôle :

-  : pour la navigation classique,
-  pour supprimer un enregistrement,
-  pour créer un nouvel enregistrement.

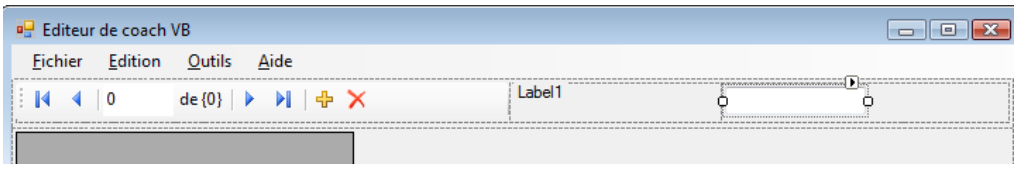
Vous pouvez agrémenter cette barre, exactement comme pour une barre d'outils, de nouveaux boutons ou autres contrôles personnalisés en cliquant  (n'oubliez pas de sélectionner le contrôle BindingNavigator pour voir apparaître cet icône à droite) :



- Affichez la fenêtre de **Propriétés** du contrôle.
- Configurez la propriété **(Name)** avec la valeur : **MainBindingNavigator**.

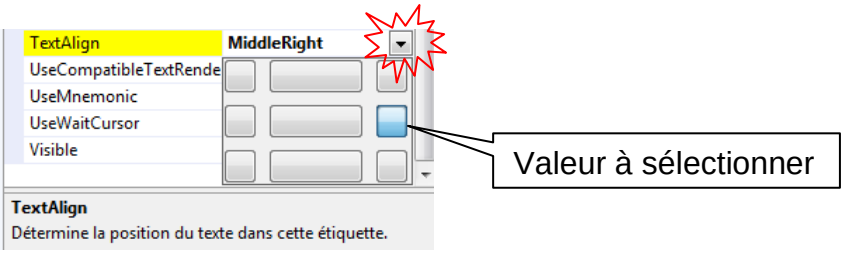


3. Ajoutez maintenant les deux derniers contrôles dans les deux autres cellules du second tableau :
   - Faites un glisser déplacer de la **Boîte à outils** > rubrique **Contrôles Communs** > d'un contrôle **Label** dans la deuxième cellule du tableau **TableLayoutPanel2**.
   - Faites un glisser déplacer de la **Boîte à outils** > rubrique **Contrôles Communs** > d'un contrôle **TextBox** dans la troisième et dernière cellule du tableau **TableLayoutPanel2** :

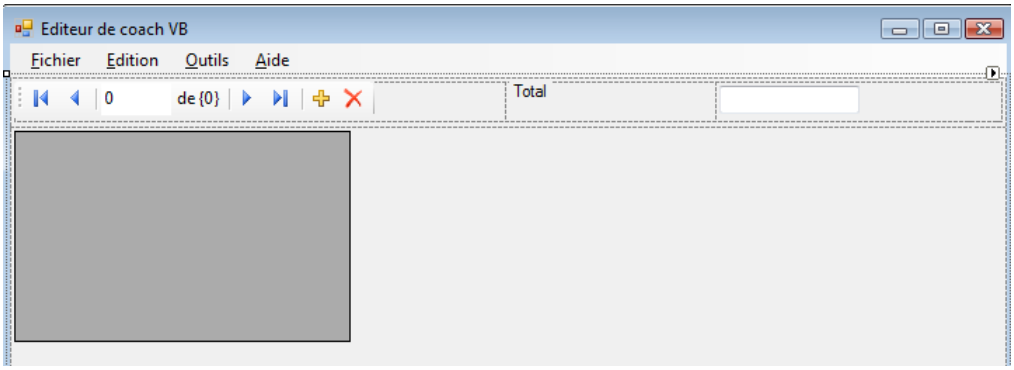


   - Affichez la fenêtre de **Propriétés** du contrôle **TextBox1**.
   - Configurez sa propriété **(Name)** avec la valeur : **TotalTextBox**.
   - Configurez sa propriété **ReadOnly** avec la valeur : **True** pour indiquer que la zone de texte sera en lecture uniquement.
   - Sélectionnez le contrôle **Label1**.
   - Configurez sa propriété **(Name)** avec la valeur : **TotalLabel**.
   - Configurez sa propriété **Text** avec la valeur : **Total**.
   - Configurez sa propriété **TextAlign** avec la valeur **MiddleRight** pour aligner le libellé à droite avec un centrage vertical.

Utilisez l'éditeur de propriété associé en cliquant sur la flèche à droite de la ligne de propriété.

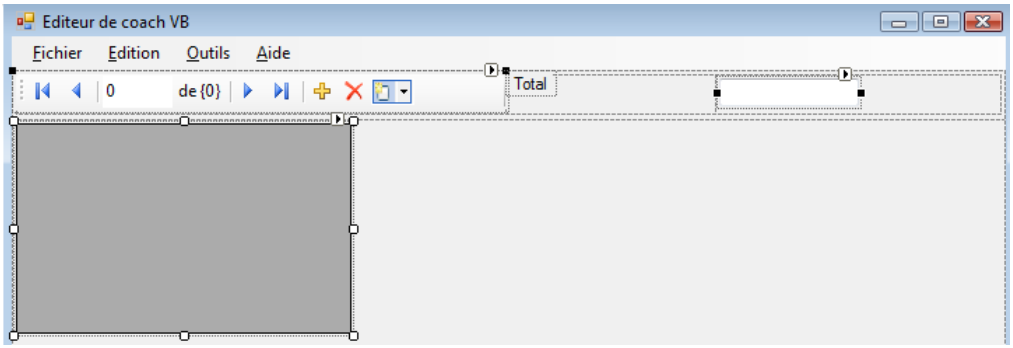


Vous obtenez :

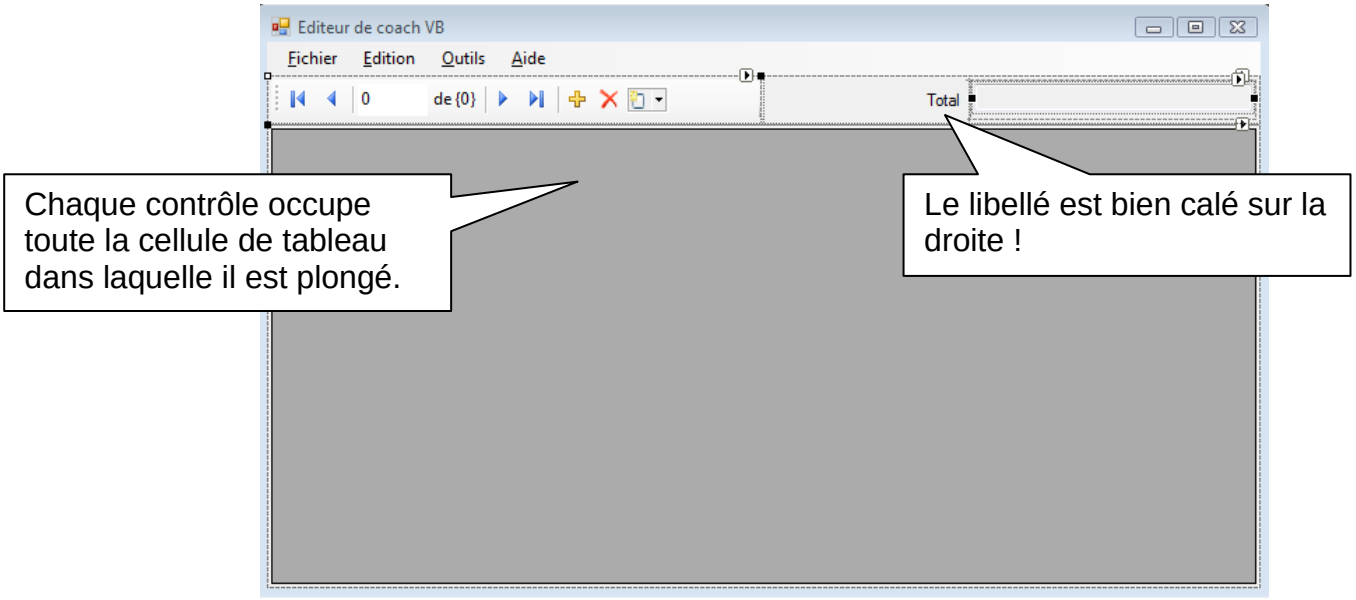


Le résultat n'est pas vraiment satisfaisant car chaque contrôle reste mal dimensionné. C'est là qu'il faut penser aux propriétés d'ancrage (**Dock**, **Anchor**) des contrôles vis-à-vis de leur conteneur parent.

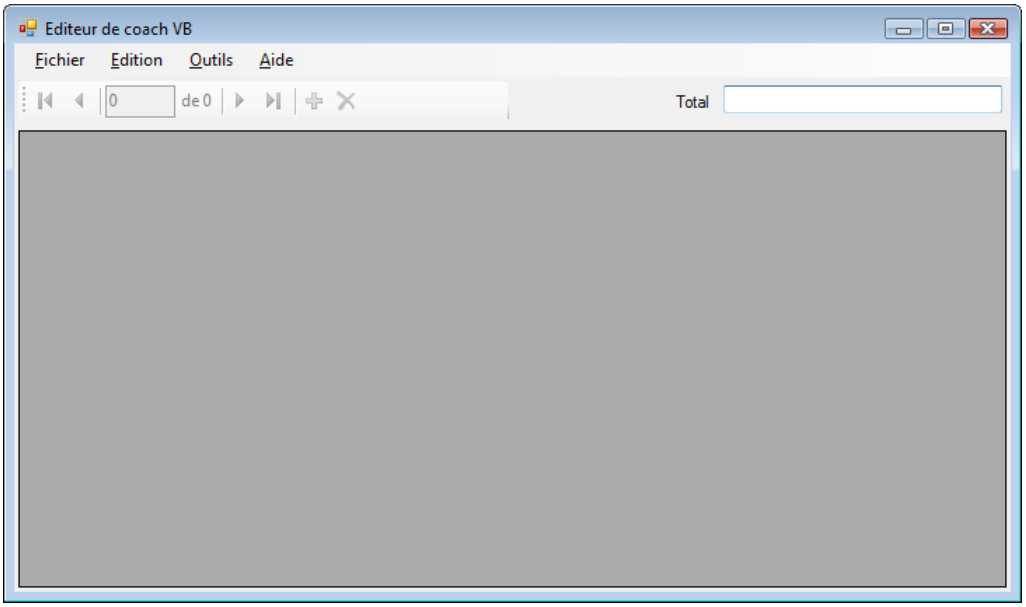
- Tout en maintenant la touche **Shift**  appuyée, cliquez sur chacun des quatre contrôles, **TotalTextBox**, **TotalLabel**, **MainBindingNavigator** et **MainDataGridView** pour effectuer une sélection multiple de contrôles.
- Relâchez la touche **Shift**.



- Affichez la fenêtre de propriétés qui vous propose toutes les propriétés communes des contrôles de la sélection.
- Configurez la propriété **Dock** à la valeur **Fill** de façon à ce que chaque contrôle remplisse toute la surface disponible de son conteneur. Vous obtenez :



4. Testez l'affichage de notre interface à l'exécution :
   - Enregistrez tous vos changements en cliquant sur dans la barre d'outils de Visual Studio.
   - Exécutez l'application en cliquant sur (ou touche F5).



Pour l'instant la barre de navigation est inactive car aucune donnée n'est affichée dans la grille mais l'interface fonctionne. Maintenant il ne reste plus qu'à programmer le code pour alimenter la grille avec le contenu d'un fichier CSV.

3 Alimenter les contrôles de données avec un fichier CSV

Dans cet exercice, vous allez apprendre à :

- Lire et enregistrer un fichier au format CSV,
- Alimenter des contrôles d'affichage avec des données,
- Utiliser une table mémoire,
- Mettre en œuvre le mécanisme de liaison de données (databinding).

Objectif
L'objectif de cet exercice est d'apprendre à manipuler des données en provenance d'un fichier texte délimité (*.csv).

La démarche que nous allons adopter consiste à utiliser un objet mémoire appelé **DataTable**, qui va servir d'intermédiaire entre le fichier et les contrôles d'affichage de notre application.

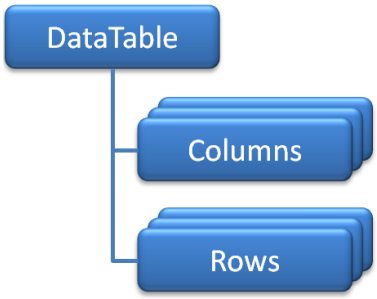
En effet, nous allons voir qu'il est très simple de faire interagir des contrôles d'affichage de données et une source de données en mémoire via le mécanisme dit de liaison de données (databinding). Un tel mécanisme n'existe pas directement sur une ressource de type fichier.

3.1 Créer une table mémoire (DataTable)

Dans ce premier exercice, nous allons préparer la structure de données mémoire dans laquelle nous chargerons les données de fichier.

Pour stocker les données en mémoire, nous vous proposons d'utiliser un objet de type **DataTable**.

C'est quoi exactement une **DataTable** ?
DataTable est une classe de l'espace de noms **System.Data** fourni par le Framework .NET, qui représente une table de données en mémoire. Qui dit table, dit **colonnes** et bien sûr **lignes**. La structure globale d'une **DataTable** est en effet la suivante :



- La propriété **Columns** d'un objet **DataTable** réfère à la collection des colonnes qui appartiennent à la table. Les objets de cette collection sont de type **DataColumn** ;
- La propriété **Rows** réfère à la collection des lignes qui appartiennent à la table. Les objets de cette collection sont de type **DataRow**.

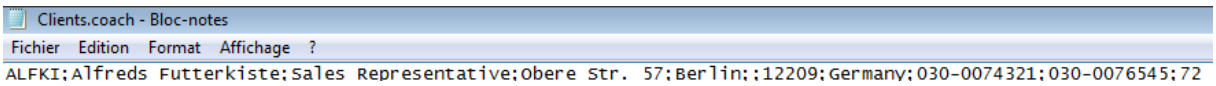
Autrement dit, ce que nous vous proposons de faire dans cet exercice, est de définir la structure de la table en mémoire en utilisant sa collection **Columns**. Nous manipulerons par la suite les données à l'aide de sa collection **Rows**.

Pour tout savoir sur l'espace de nom **System.Data** :
http://msdn2.microsoft.com/fr-fr/library/ax3wd0k9.aspx

Pour tout savoir sur la classe **DataTable** :
http://msdn2.microsoft.com/fr-fr/library/9186hy08.aspx

Contexte fonctionnel

Dans notre cas, les données sont stockées dans des fichiers texte délimités dont un enregistrement est composé de douze champs comme suit :



Nom	*Description*	*Type*
Id	Code d'indentification du client	string
Contact	Nom du contact principal du client	string
Titre	Fonction du contact principal du client	string
Adresse	Adresse de l'entreprise	string
Ville	Ville de résidence de l'entreprise	string
Region	Région de résidence de l'entreprise	string
CodePostal	Code postal du bureau postal distributeur	string
Pays	Pays de résidence de l'entreprise	string
Telephone	Numéro de téléphone du standard de l'entreprise	string
telecopie	Numéro de la télécopie principale de l'entreprise	string
CA	Chiffre d'affaire arrondi, en millier d'euros, que vous réalisez avec cette entreprise	int

La table de données en mémoire devra donc présenter la même structure soit douze colonnes.

Déroulement de l'exercice :

1. Définissez la structure de la table mémoire dans une fonction isolée de la classe Main appelée par exemple CreateDataTable :
   - Faites un clic droit dans l'**Explorateur de solutions** sur le fichier **Main.vb** > **Afficher le code** pour afficher le code de la classe **Main**.
   - Juste après la fin du constructeur **New** de la classe, ajoutez la définition d'une nouvelle fonction **CreateDataTable** comme suit :

Code VB

```
Public Class Main
   Public Sub New()
      …
   End Sub
   Function CreateDataTable() As DataTable
   End Function
   …
End Class
```

L'objectif de cette fonction est de créer un objet représentant la table en mémoire et de retourner une référence vers celui-ci à l'appelant. Le type de la valeur de retour est donnée par le mot clé **As** à la fin de la ligne de déclaration de la fonction.

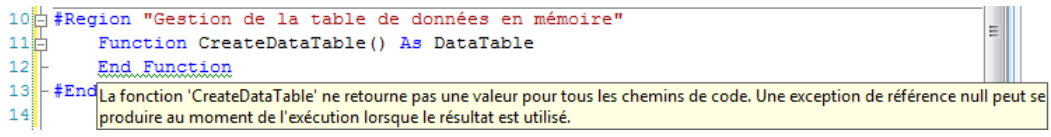
Function <NomDeLaFonction>(<Liste des paramètres>) As <TypeDeLaValeurRetour>

- Entourez la fonction d'une région nommée **Gestion de la table de données en mémoire** :

Code VB

```
#Region "Gestion de la table de données en mémoire"
   Function CreateDataTable() As DataTable
   End Function
#End Region
```

A priori, vous devriez voir apparaître un trait vert en dessous de **End Function**. Rappelez-vous que ces codes couleurs sont des indications précieuses de Visual Studio. Pour avoir le détail du problème dans une info bulle, il suffit de stopper la souris sur la ligne de couleur.
Visual Studio vous informe ici qu'il ne détecte pour l'instant aucune valeur de retour dans votre fonction et qu'il y a donc lieu de s'en préoccuper.

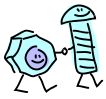


- Ajoutez la définition d'une variable de type **DataTable** puis préparer la ligne de retour de la fonction avec l'objet ainsi créé :

Code VB

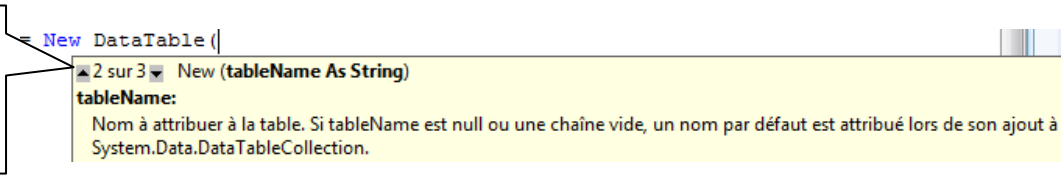
```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Return result
    End Function
#End Region
```

Avec ces deux lignes, le trait vert en dessous de **End Function** doit disparaître !

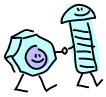


Quel est l'objet de la première ligne ?
Nous avons vu dans l'atelier précédent que le mot clé **New** permet de créer un nouvel objet d'un type donné et qu'il est suivi du *constructeur* de la classe concernée. Lorsque vous tapez la première parenthèse du constructeur, l'IntelliSense vous donne l'ensemble des constructeurs disponibles sur ce type d'élément. En fait il y a différents constructeurs d'un objet car il y a tout simplement plusieurs façons différentes d'initialiser l'objet. Par exemple, ici nous avons choisi le second qui permet d'initialiser le nom de la table que nous allons nommée : **Coach**.

Utiliser les flèches pour balayer les constructeurs disponibles



2. Ajoutez une première colonne à la table correspondant à l'Id d'un enregistrement de données dans notre fichier :



Une colonne est elle-même un objet, dont le type est donné par la classe **DataColumn**. Donc pour créer une colonne, il suffit d'utiliser le mot clé **New** suivi du constructeur de l'objet comme précédemment.

- Ajoutez la création d'un objet de type **DataColumn** comme suit :

Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Dim idColumn As DataColumn = New DataColumn()
        Return result
    End Function
#End Region
```

Pour être défini précisément, vous devez indiquer le type de données qui seront contenues dans la colonne ainsi que son nom.

Pourquoi est-ce que c'est si important de définir le type des données ?
Parce que si vous imposez un maximum de restrictions à l'entrée de votre table, vous évitez d'autant les erreurs intempestives. Les types des données doivent par exemple correspondre avec ceux de la source de données, à savoir notre fichier plat.
Il faut savoir qu'un objet de type **DataColumn** peut être extrêmement riche et que son rôle est d'être capable de reproduire les mêmes contraintes qu'un schéma de base de données (contraintes d'unicité, colonne auto incrémentale, valeur nulle autorisée etc…)

Pour en savoir plus sur la classe **DataColumn** :
http://msdn.microsoft.com/fr-fr/library/system.data.datacolumn.aspx

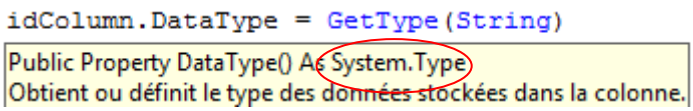
- Ajoutez les lignes suivantes pour configurer le nom et le type de la colonne :

Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Dim idColumn As DataColumn = New DataColumn()
        idColumn.ColumnName = "Id"
        idColumn.DataType = GetType(String)
        Return result
    End Function
#End Region
```

Qu'est ce que *GetType* ?
GetType est un opérateur du langage Visual Basic. Il est utilisé pour récupérer l'objet correspondant à un type du langage.

Qu'est ce que c'est que l'objet correspondant à un type de données ?
En fait, la propriété **DataType** de notre objet **idColumn** doit référencer un objet qui définisse le type de données de la colonne. Ce n'est pas le type de données en lui-même tel que **String** ou **Integer**, non, non… Ce doit être un objet qui *décrive* le type. C'est un peu comme au restaurant. Pour commander, vous ne demandez pas les plats mais la carte du menu qui décrit chaque plat.



Chaque type du système de type communs du Framework .NET est décrit dans un objet de type **System.Type**.

Et comment est-ce qu'on fait pour dégoter un objet qui décrive par

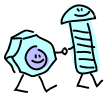
exemple le type de données **String** ?
C'est tout simple, il suffit de faire appel à l'opérateur **GetType** en lui donnant le type de données spécifié ☺ !

Pour en savoir plus sur l'opérateur **GetType** :
http://msdn.microsoft.com/fr-fr/library/tay4kywk.aspx

- Ajoutez une dernière ligne pour ajouter la colonne que vous avez créée à la collection des colonnes de la table **Coach** :

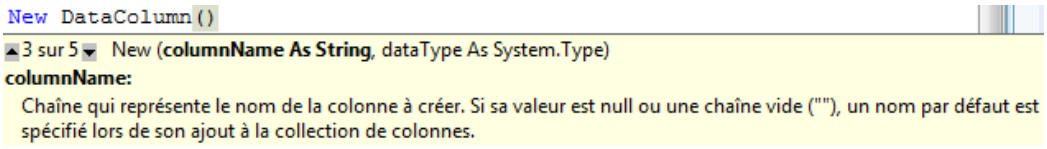
Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Dim idColumn As DataColumn = New DataColumn()
        idColumn.ColumnName = "Id"
        idColumn.DataType = GetType(String)
        result.Columns.Add(idColumn)
        Return result
    End Function
#End Region
```



Au final, toutes ces lignes pour créer une seule colonne, cela fait pas mal de lignes en perspectives pour créer les dix colonnes restantes…
Heureusement il existe un moyen de réduire le code :

- D'abord en utilisant la notion de *constructeur* d'objet pour initialiser le nom et le type de la colonne en une seule ligne. Si vous utilisez l'IntelliSense, vous constaterez qu'en effet la classe **DataColumn** propose un constructeur qui permet d'initialiser les deux informations en une seule instruction :

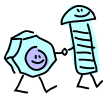


- Supprimez les deux lignes d'initialisation de l'objet et modifiez la ligne de déclaration et de construction de l'objet comme suit :

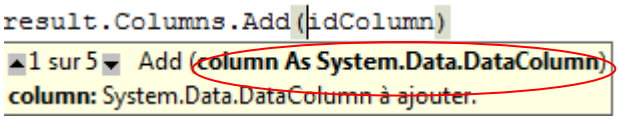
Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Dim idColumn As DataColumn = New DataColumn("Id", GetType(String))
        idColumn.ColumnName = "Id"
        idColumn.DataType = GetType(String)
        result.Columns.Add(idColumn)
        Return result
    End Function
```

```
#End Region
```



- Ensuite il faut savoir que l'opérateur **New** peut être utilisé directement partout où l'on attend une référence d'objet.
  Par exemple, c'est le cas de la méthode **Add** qui ajoute un objet de type **DataColumn** à la collection **DataColumns** de l'objet **DataTable**. Il lui faut simplement en paramètre une référence vers l'objet à ajouter :



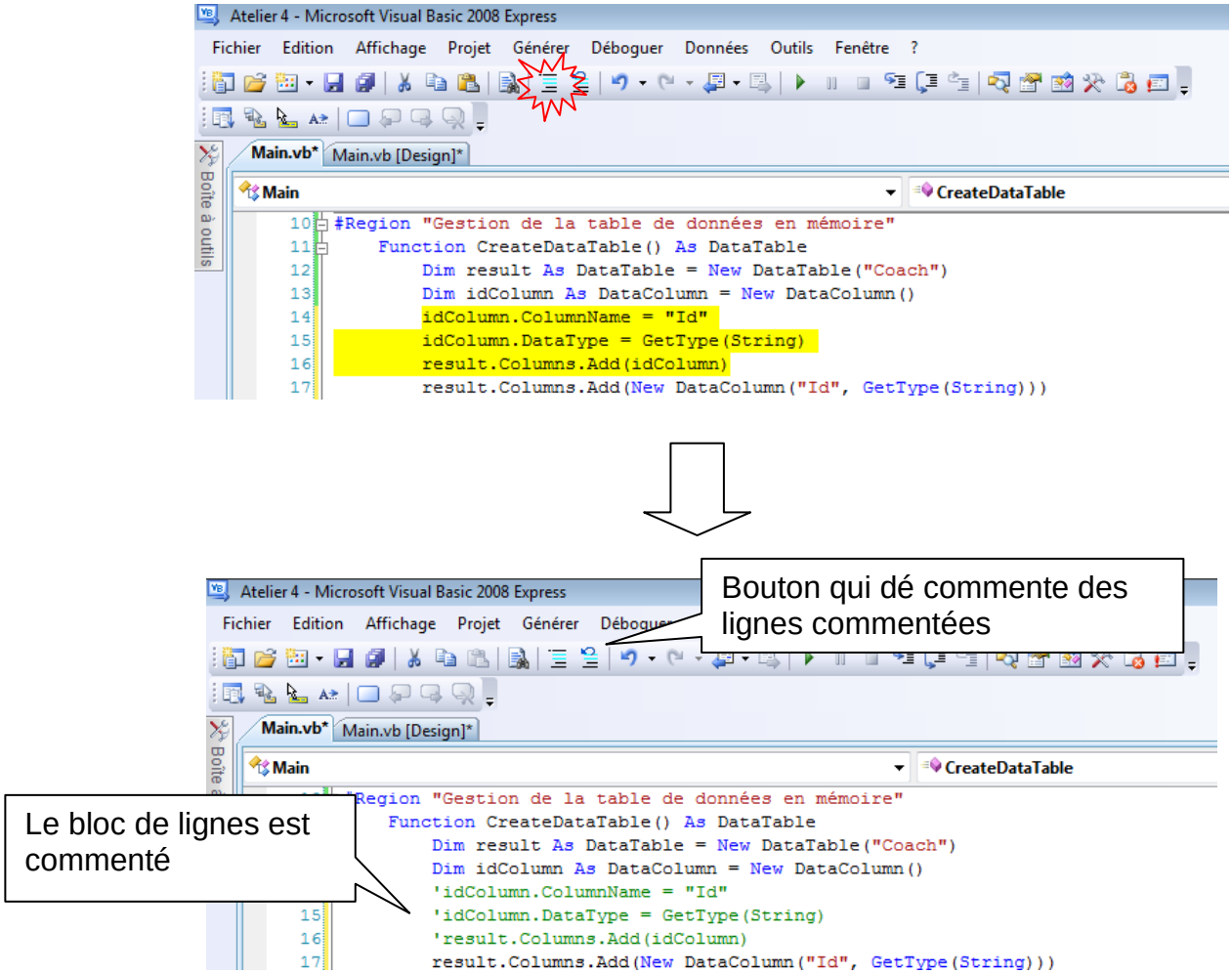
  Cette référence, c'est celle que nous avons appelée **idColumn** et définie comme étant de type **DataColumn**. On va pouvoir se passer de déclarer cette variable !

- Supprimez la déclaration de la variable **idColumn** et insérez directement le code de création de la colonne en paramètre de la méthode **Add** en utilisant l'opérateur **New** :

Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        Dim idColumn As DataColumn = New DataColumn("Id", GetType(String))
        result.Columns.Add(New DataColumn("Id", GetType(String)))
        Return result
    End Function
#End Region
```

Plutôt que de supprimer les lignes de code, vous pouvez également les mettre en commentaire. Pour mettre un bloc de plusieurs lignes en commentaire, il suffit de sélectionner (grossièrement) les lignes avec la souris puis de cliquer le bouton de la barre d'outils standard de Visual Studio.
A l'inverse, pour dé commenter ces même lignes, il suffit de cliquer le bouton .



3. Ajoutez maintenant les dix autres colonnes à la table :
   - A la suite dans la fonction **CreateDataTable**, ajoutez les autres colonnes qui structurent la table :

Code VB

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable() As DataTable
        Dim result As DataTable = New DataTable("Coach")
        result.Columns.Add(New DataColumn("Id", GetType(String)))
        result.Columns.Add(New DataColumn("Contact", GetType(String)))
        result.Columns.Add(New DataColumn("Titre", GetType(String)))
        result.Columns.Add(New DataColumn("Adresse", GetType(String)))
        result.Columns.Add(New DataColumn("Ville", GetType(String)))
        result.Columns.Add(New DataColumn("Region", GetType(String)))
        result.Columns.Add(New DataColumn("Code postal", GetType(String)))
        result.Columns.Add(New DataColumn("Pays", GetType(String)))
        result.Columns.Add(New DataColumn("Telephone", GetType(String)))
        result.Columns.Add(New DataColumn("Telecopie", GetType(String)))
        result.Columns.Add(New DataColumn("CA", GetType(Integer)))
        Return result
    End Function
```

Attention à la dernière colonne qui est de type Integer (et non String)

```
#End Region
```

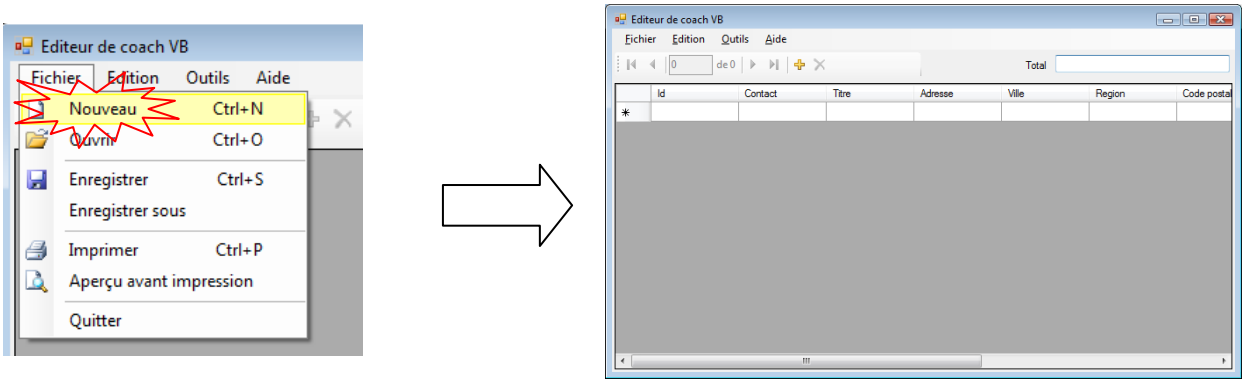
Parfait ! De cette façon, vous disposez maintenant d'une table en mémoire reflétant le schéma de votre source de données. Vous allez donc pouvoir la charger avec des données en provenance de fichier. Vous pouvez également l'utiliser dès maintenant comme une structure de données vierge que l'utilisateur peut enrichir avec de nouvelles données.
Dans les deux cas, nous avons besoin de lier la table mémoire avec des contrôles de données d'affichage qui vont permettre d'interagir avec l'utilisateur.

3.2 Comprendre la liaison de données (DataBinding)

L'objectif de cet exercice est de présenter le mécanisme de liaison de données (ou DataBinding en anglais).

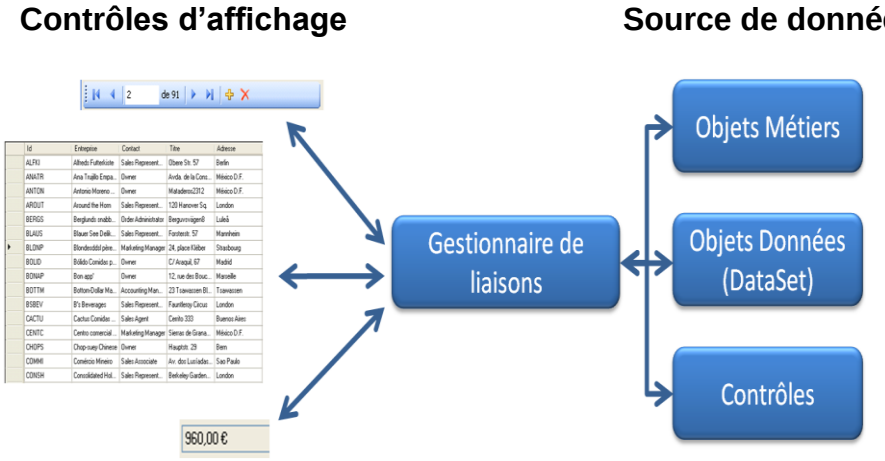
Contexte fonctionnel

Avant de se lancer dans la manipulation d'un fichier, nous vous proposons d'exploiter la table de données mémoire à l'état vierge pour créer de nouvelles données (en vue bien entendu de les sauvegarder par la suite sur une structure fichier).
L'utilisateur clique le menu **Fichier** > **Nouveau** de l'application. Une grille vide doit apparaître dans le formulaire de façon à ce qu'il puisse saisir des données.



Déroulement de l'exercice :

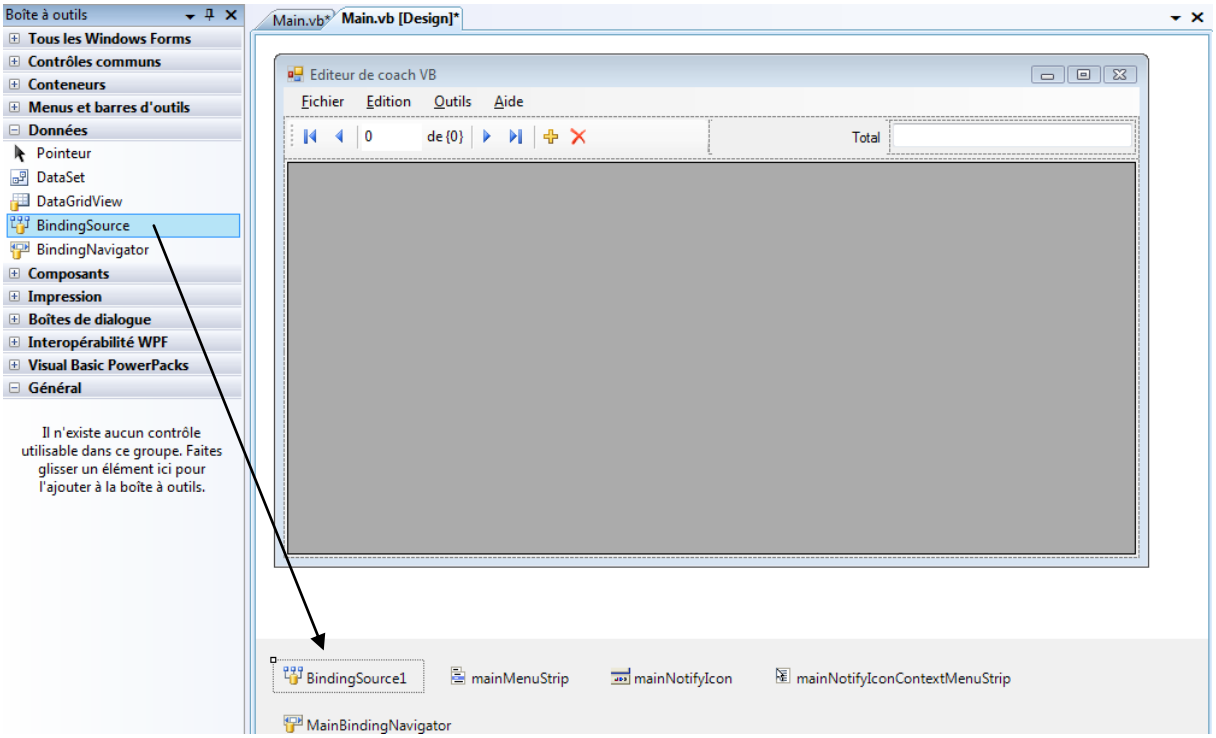
Pourquoi est-ce qu'on a besoin du *DataBinding* ?
Le **DataBinding** est le mécanisme par lequel la valeur d'une propriété d'un contrôle d'affichage est automatiquement maintenue à jour en fonction de la valeur de la propriété d'un objet fournissant des données. Le contrôle d'affichage et l'objet source de données sont étroitement liés (d'où le terme *binding*) si bien que la modification de l'un entraîne immédiatement la mise à jour en correspondance de l'autre et vice versa.
On peut schématiser le fonctionnement comme suit :



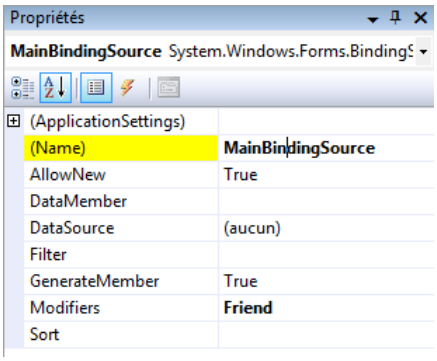
Qu'est ce qu'un *gestionnaire de liaisons* ?
Pour simplifier la liaison de données, le modèle des Windows Forms fournit un composant appelé **BindingSource** qui agit comme un intermédiaire entre la source de données et les contrôles d'affichage dépendants. On parle de *gestionnaire de liaisons* car il gère la liaison, telle que la notification des modifications de part et d'autre.

Pour tout savoir sur le modèle de liaison de données avec les Windows Forms :
http://msdn.microsoft.com/fr-fr/library/ef2xyb33.aspx

1. La première étape consiste donc à créer un gestionnaire de liaison pour gérer l'interconnexion entre nos contrôles d'affichage de données et la table mémoire :
   - Basculez sur le formulaire **Main.vb** en mode **Design**.
   - Faites un glisser-déplacer de la **Boîte à outils** > rubrique **Données** > du contrôle **BindingSource** sur la zone de dépôt de contrôles du formulaire :



- Affichez les propriétés du contrôle **BindingSource1** (F4).
- Configurez la propriété **(Name)** avec la valeur : **mainBindingSource** :



Pour tout savoir sur la classe **BindingSource** :
http://msdn.microsoft.com/fr-fr/library/h974h4y2.aspx

2. La seconde étape consiste à créer un gestionnaire d'évènement en réponse à l'évènement clic du menu Fichier > Nouveau de l'application :
   - Toujours sur le formulaire **Main.vb** en mode **Design**, sélectionnez le menu **Fichier** puis double cliquez sur l'option de menu **Nouveau** pour générer la procédure **NouveauToolStripMenuItem_Click** :

```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click


End Sub
```

3. Créez la table mémoire à l'aide de la fonction CreateDataTable écrite dans l'exercice précédent :

- Dans la procédure **NouveauToolStripMenuItem _Click**, ajoutez le code de création d'une table mémoire comme suit :

Code VB

```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
End Sub
```

4. Indiquez au gestionnaire de liaisons mainBindingSource que la source de données à prendre en charge est la table mémoire newDataTable :
    - Attribuez la nouvelle table à la propriété **DataSource** de l'objet **mainBindingSource** :

Code VB

```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
    'Configuration du gestionnaire de liaison sur la source de données
    MainBindingSource.DataSource = newDataTable
End Sub
```

5. Liez les contrôles d'affichage de données avec le gestionnaire de liaisons :
    - D'abord le contrôle de grille de données **mainDataGridView** avec sa propriété **DataSource** :

Code VB

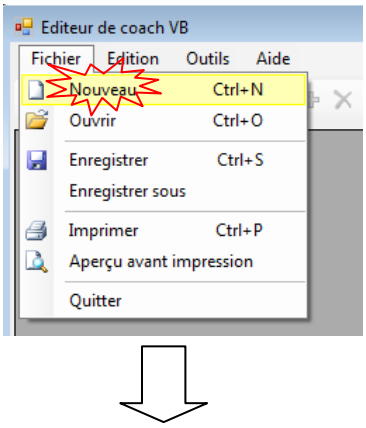
```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
    'Configuration du gestionnaire de liaison sur la source de données
    MainBindingSource.DataSource = newDataTable
    'Liaison du gestionnaire avec les contrôles d'affichage de données
    MainDataGridView.DataSource = MainBindingSource
End Sub
```

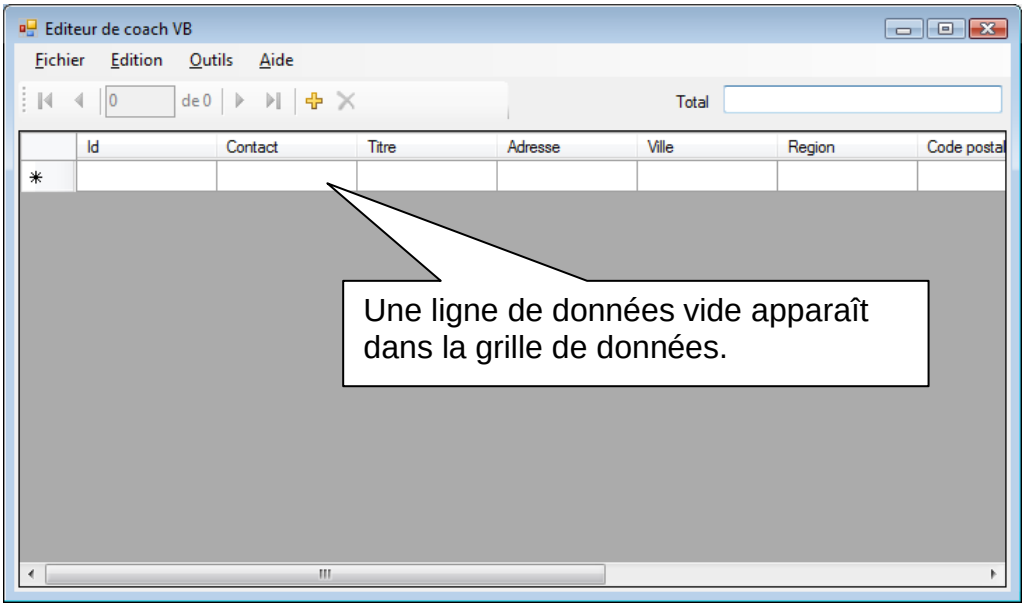
    - Ensuite le contrôle de navigation **mainBindingNavigator** avec sa propriété **BindingSource** :

Code VB

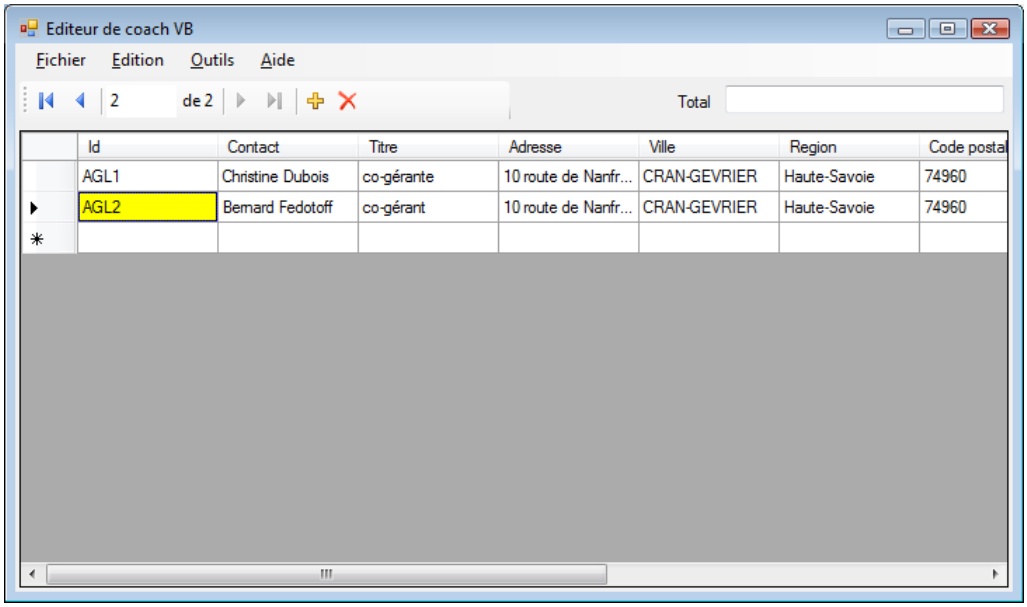
```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
    'Configuration du gestionnaire de liaison sur la source de données
    MainBindingSource.DataSource = newDataTable
    'Liaison du gestionnaire avec les contrôles d'affichage de données
    MainDataGridView.DataSource = MainBindingSource
    MainBindingNavigator.BindingSource = MainBindingSource
End Sub
```

6. Testez le fonctionnement des contrôles et de la table mémoire :
   - Enregistrez tous vos changements en cliquant sur dans la barre d'outils de Visual Studio.
   - Exécutez l'application en cliquant sur (ou touche F5).
   - Cliquez le menu **Fichier > Nouveau** :



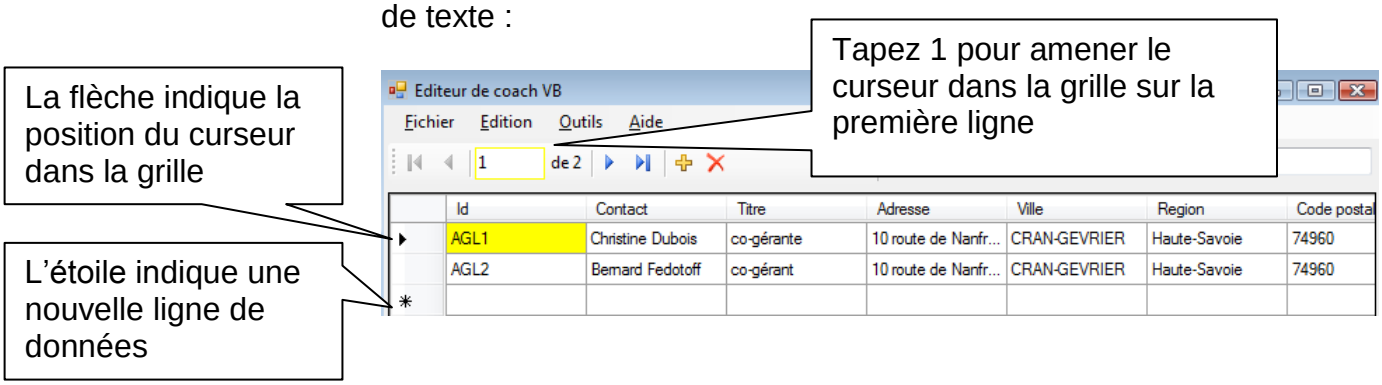


   - Amusez-vous à saisir d'autres informations.



Vérifiez que la barre de navigation fonctionne également. Par exemple :

- sélectionnez une ligne de données et naviguez d'une ligne à l'autre en cliquant sur les boutons ou ,
- ou entrez directement le numéro de la ligne souhaitée dans la zone de texte :



- Ajoutez une nouvelle ligne en cliquant ou en vous positionnant sur la grille en face de la ligne marquée d'une étoile (*),
- Supprimez une ligne en cliquant .

Bravo ! La grille fonctionne, liée à la table mémoire.

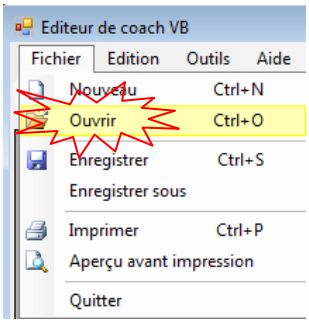
3.3 Manipuler le fichier CSV

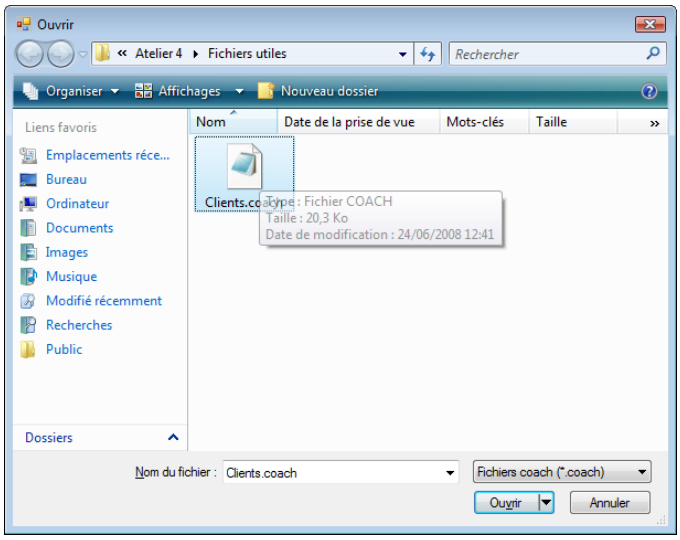
L'objectif de cet exercice est de lire les données de la grille à partir d'un fichier au format CSV (texte délimité), et bien sur de sauvegarder de nouveau ces données, après modification, dans le même fichier CSV (ou un autre d'ailleurs)

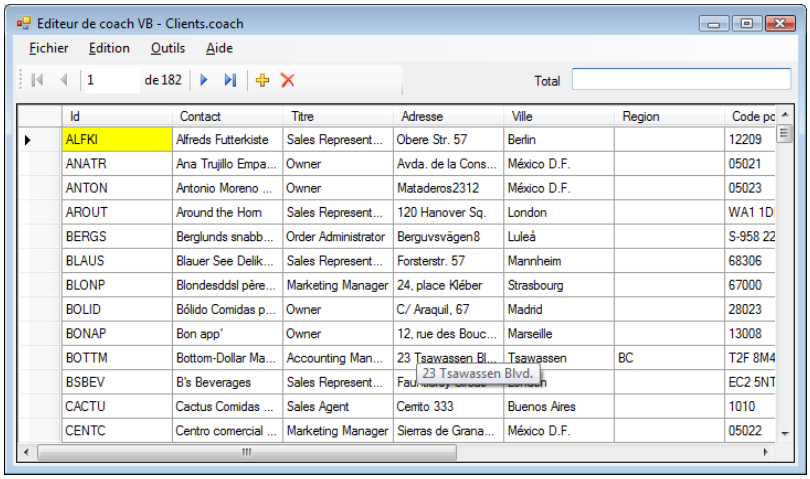
3.3.1 Lire le fichier

Contexte fonctionnel

Pour charger un fichier dans la grille de données de l'application, l'utilisateur clique le menu **Fichier** > **Ouvrir** puis sélectionne le fichier de données dans la structure de répertoires de la machine :







Il peut ensuite travailler sur les données : ajouter de nouveaux enregistrements, supprimer des enregistrements, mettre à jour des informations existantes etc…

Un fichier CSV est composé de ligne de valeurs, séparées par un séparateur qui sera dans notre cas le « **;** », et dont l'ordre des valeurs est toujours le même d'une ligne à l'autre. Chaque ligne est finie par un retour-chariot (**CR**).

L'algorithme à développer pour lire un tel fichier et le charger dans la grille de données du formulaire est composé de quatre étapes :

1. Sélectionner le fichier,
2. Ouvrir le fichier,
3. Lire le fichier CSV ligne à ligne,

4. Pour chaque ligne de texte lue, générer un enregistrement dans la table de données.

Etape 1 : La première étape consiste donc à écrire le code de sélection du fichier sur le disque.

Nous allons coder l'ouverture du fichier sur le menu **Fichier** > **Ouvrir** du formulaire **Main**. Cela revient donc à créer un gestionnaire d'évènement associé à l'évènement **Click** de l'option de menu correspondante.

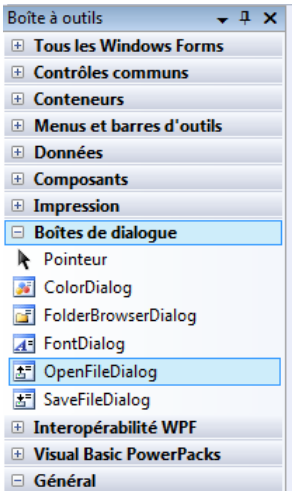
1. Créez un gestionnaire d'évènement associé au clic du menu Ouvrir :
   - Affichez le formulaire **Main** en mode **Design**.
   - Sélectionnez le menu **Fichier** puis double cliquez sur l'option de menu **Ouvrir** pour générer la procédure **OuvrirToolStripMenuItem_Click** :

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click

End Sub
```

2. Utilisez la boîte de dialogue standard d'ouverture de fichier de Windows pour permettre à l'utilisateur de faire sa sélection :

Vous vous souvenez des boîtes de dialogue communes fournies par le Framework .NET ? Nous en avons utilisé une dans l'atelier 3 de ce tutorial pour sélectionner un chemin de répertoire sur le disque. Nous allons utiliser ici la boîte **OpenFileDialog** pour sélectionner un fichier sur le disque.
Pour rappel, ces boîtes de dialogue se manipulent en tant qu'objets que Visual Studio peut créer pour vous via les composants de la **Boîte à outils** > rubrique **Boîtes de dialogue**.
Vous pouvez aussi créer vos propres objets dans le code directement à partir des classes du Framework .NET. C'est cette deuxième approche que nous allons adopter.



- Ajoutez à la procédure le code de déclaration et d'instanciation d'un nouvel objet de type **OpenFileDialog** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
End Sub
```

Les principales propriétés à configurer sur cet objet sont :
- **Filter** : Cette propriété sert à filtrer la liste des fichiers de la boîte

de dialogue, typiquement sur l'extension de fichier pour orienter l'utilisateur directement sur le type de fichier autorisé. Nous pouvons donc filtrer tous les fichiers d'extension ***.coach**.

- **InitialDirectory** : Cette propriété configure le répertoire affiché par défaut lors de l'ouverture de la fenêtre.

- Complétez le code pour filtrer le type de fichier sur l'extension ***.coach** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
End Sub
```

- Configurez le répertoire par défaut sur le dossier préférentiel de l'utilisateur configuré à l'aide de la fenêtre **Options** à l'atelier 3 dans la variable privée **SaveDirectoryPath** de la classe **Main** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
End Sub
```

Reste à afficher la boîte de dialogue. Toujours dans le même atelier 3 vous avez appris à utiliser la méthode **ShowDialog**. Le code d'ouverture du fichier doit être exécuté si l'utilisateur clique **OK** pour fermer la boîte d'ouverture.

- Affichez la boîte de dialogue et testez le code de fermeture avec une condition **If…Then…End If** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog =  _
            Windows.Forms.DialogResult.OK Then
       'Code d'ouverture du fichier sélectionné
    End If
End Sub
```

Etape 2 : La seconde étape consiste à ouvrir le fichier.

Comment retrouve-t-on le fichier sélectionné par l'utilisateur ?
C'est la propriété **FileName** de l'objet **openDataFileDialog** qui donne le nom du fichier sélectionné par l'utilisateur dans la boîte de dialogue.
En réalité le nom de cette propriété (FileName) est un faux ami car elle indique non seulement le nom du fichier mais aussi le chemin complet pour y accéder.

Une bonne pratique de développement est de créer une méthode ou fonction par étape d'algorithme, et ce afin d'augmenter la maintenabilité et la lisibilité du code. Autrement dit, plutôt que d'écrire le code d'ouverture du fichier directement dans le gestionnaire d'évènement précédent, nous allons coder une nouvelle procédure. Et nous procèderons de même pour les autres étapes de notre algorithme.

3. Ajoutez une fonction dont l'objectif est d'ouvrir et lire le contenu d'un fichier :
   - Toujours dans le fichier de code **Main.vb**, juste après la fin du constructeur **New** de la classe **Main**, ajoutez la définition d'une nouvelle fonction **ReadFile** comme suit :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
End Function
```

Notez que la fonction a évidemment besoin du chemin du fichier qu'il lui faudra ouvrir. C'est l'objet du paramètre **FileName**.

Pourquoi prévoir une **valeur de retour** de type **DataTable** ?
Pour rappel, notre objectif est de lire le fichier pour charger son contenu dans la table mémoire. Voilà pourquoi le résultat de l'opération que nous vous proposons est un objet de type **DataTable** qu'il suffira de lier aux contrôles d'affichage de données comme nous l'avons vu dans l'exercice précédent.

- Codez aussitôt la déclaration d'une variable de retour et renvoyez-la à l'aide du mot clé **Return** :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    Dim table As DataTable = Nothing
    'Renvoi de la valeur de retour
    Return table
End Function
```

Pensez à encadrer votre code à l'aide des régions !
Construisez par exemple une région pour retrouver facilement tous les codes en rapport avec le traitement de fichier !

- Entourez la fonction d'une région nommée **Traitement des fichiers de données** :

Code VB

```
#Region "Traitement des fichiers de données"
 Function ReadFile(ByVal FileName As String) As DataTable
     …
 End Function
#End Region
```

Nous allons avoir besoin de définir deux variables au sein de la fonction :

- **readRow**, de type **string,** dans laquelle nous récupérerons une à une les lignes de fichier lues. Chaque ligne devra être chargée dans la table mémoire.
- **isFirstRow**, de type **booléen**, pour différencier la lecture de la première ligne des autres lignes du fichier. En effet, au moment de la lecture de la première ligne il faudra prévoir de créer la table mémoire avant de penser à y charger la ligne lue.

- Ajoutez à la fonction **ReadFile** le code de définition des variables **readLine** et **isFirstLine** comme suit :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    ‘Renvoi de la valeur de retour
    Return table
End Function
```

Pensez à initialiser vos variables pour éviter les erreurs (surprises) intempestives à l’exécution.

- Ajoutez à la suite le code suivant :

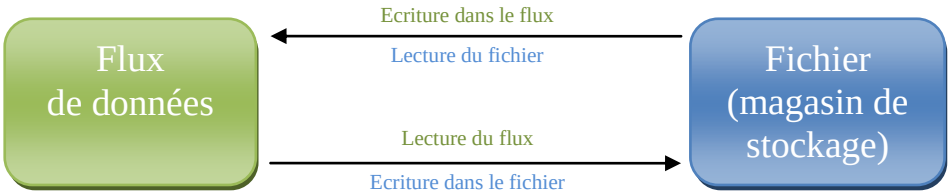
Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    'Parcours du fichier à l’aide d’un StreamReader
    Dim sr As StreamReader = New StreamReader(FileName)

    ‘Renvoi de la valeur de retour
    Return table
End Function
```

C'est quoi un **StreamReader** ?
C'est un type d'objet qui lit des caractères à partir d'un flux d'octets dans un codage particulier. Il appartient à l'espace de nommage **System.IO**, qui fournit tous les objets nécessaires pour réaliser tout type d'Entrées/Sorties avec le système.

Qu'est ce qu'on entend par *flux* ?
Pour toute opération d'entrée/sortie de fichier de base, le principe consiste à passer par un flux. C'est lui qui gère la lecture ou l'écriture des octets stockés dans le fichier. Le fichier joue le rôle de magasin de stockage des données du flux.



C'est un peu comme si vous aviez besoin d'un *lecteur* pour lire le fichier à votre place, tout comme nous aurons besoin d'un *écrivain* pour écrire le fichier.

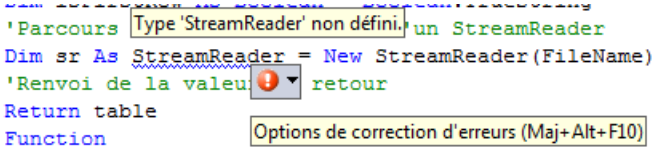
Pour tout savoir sur les flux de données :
http://msdn.microsoft.com/fr-fr/library/k3352a4t.aspx

Pour en savoir plus sur la classe **StreamReader** :
http://msdn2.microsoft.com/fr-fr/library/6aetdk20.aspx

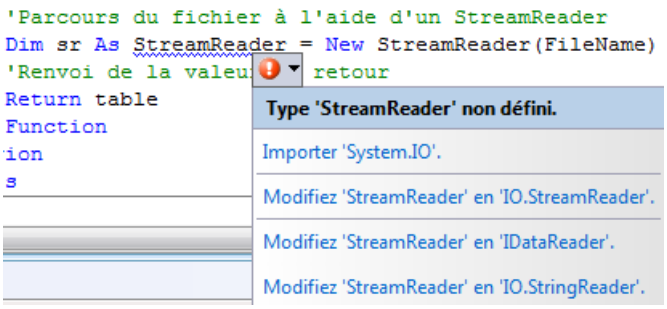
Avez-vous remarqué le surlignement bleu et le petit trait rouge en dessous de **StreamReader** ?

```
'Parcours du fichier à l'aide d'un StreamReader
Dim sr As StreamReader = New StreamReader(FileName)
```

Positionnez la souris sur le petit trait rouge en fin de mot pour faire apparaître la balise suivante :

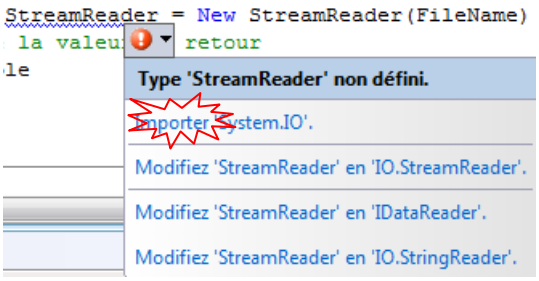


Visual Studio a un souci avec le type **StreamReader** qui ne lui dit rien du tout… Cliquez sur la petite flèche de la balise pour lister les corrections de l'erreur proposées :

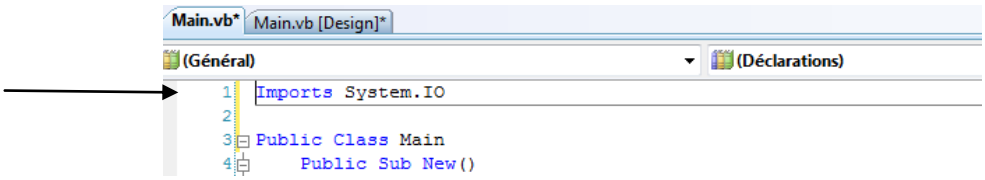


Visual Studio a bien trouvé une référence dans votre projet à la bibliothèque de classes du Framework .NET qui contient l'espace de nom **System.IO**. Il s'agit d'une référence automatiquement incluse par le modèle de projet que nous avons utilisé à la création du projet.
En revanche, il vous demande d'utiliser le nom qualifié complet de l'objet demandé pour être sûr de ne pas le confondre avec un objet de même nom d'une autre bibliothèque référencée.

Heureusement, il est possible d'éviter l'écriture du nom qualifié complet d'un type (par exemple System.IO.StreamReader) en définissant un **alias d'importation**.
C'est l'objet de la première suggestion de Visual Studio **Importer 'System.IO'**. Cliquez cette option parmi la liste des propositions.



Remontez au début du fichier **Main.vb** pour observer la ligne de code que Visual Studio a ajouté automatiquement :

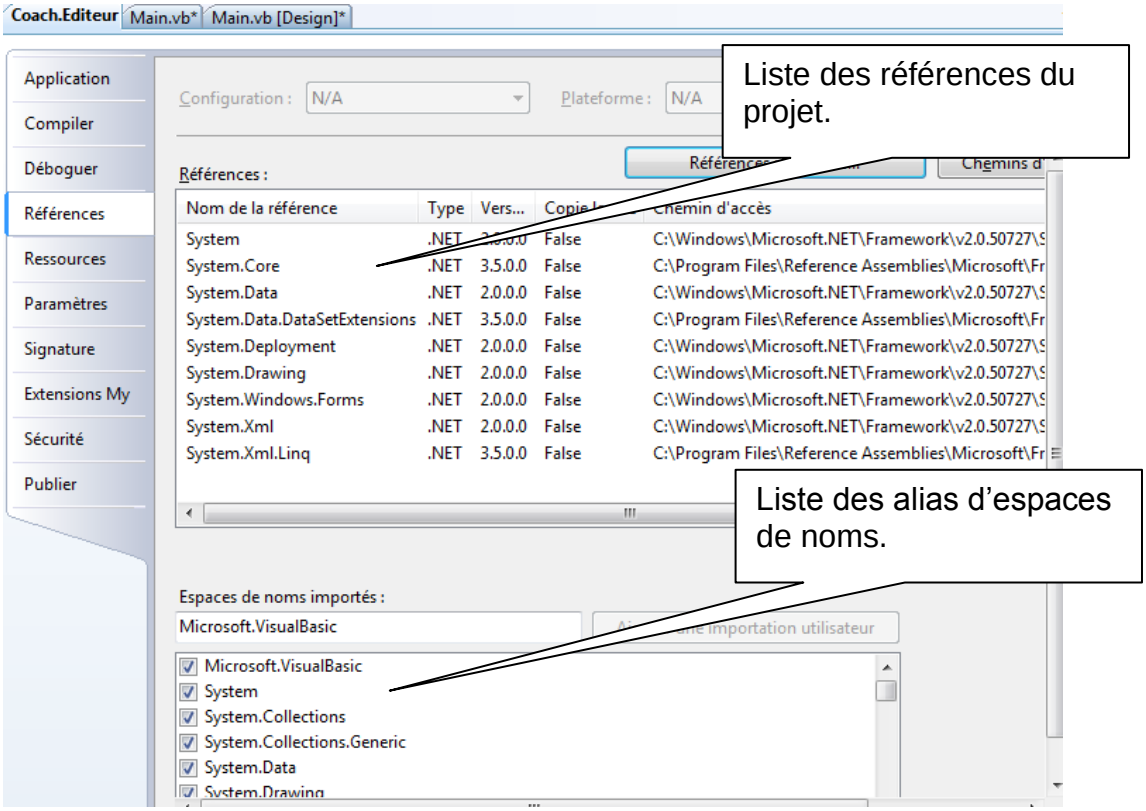


A partir de maintenant vous êtes dispensé d'écrire le nom complet des types de l'espace de noms **System.IO** au sein de ce fichier !

On aurait tout aussi bien pu définir cet alias au niveau du projet pour qu'il soit valable dans tous les fichiers de l'application. Les espaces de noms les plus courants sont d'ailleurs déjà enregistrés au niveau du projet pour vous faciliter l'utilisation des types standards du Framework .NET. C'est pourquoi nous ne nous étions encore jamais posé la question jusqu'à maintenant alors que nous avons déjà eu l'occasion d'utiliser à maintes reprises des classes du Framework .NET…

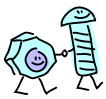
Pour retrouver les alias d'importation du projet :

- Cliquez **My Project** dans l'**Explorateur de solutions**.
- Sélectionnez l'onglet **Références**.



Il faut bien comprendre que la définition d'un alias ne remplace pas la définition de la référence. Ce sont deux étapes bien différentes. La définition de la référence est absolument nécessaire pour utiliser le type d'objet souhaité dans votre code alors que la définition d'un alias n'est pas du tout obligatoire. Il s'agit uniquement d'un confort d'écriture que vous pouvez décider de vous octroyer ou non.

Pour en savoir plus sur la définition d'un alias d'importation à l'aide du mot clé **Imports** :
http://msdn.microsoft.com/fr-fr/library/7f38zh8x.aspx



Fermons la parenthèse sur les alias et rouvrons une autre parenthèse ☺.
D'un point de vue objet, nous avions déjà évoqué le fait qu'il est très important de toujours veiller à supprimer les références d'objet que vous utilisez. Pour tout objet géré (*managé*) par le Framework.NET, vous savez que le GC (Garbage Collector) s'occupe de tout.
Mais ce n'est pas le cas pour les ressources non gérées (*non managées*), comme par exemple les ressources de type fichier.
Heureusement le langage Visual Basic fournit une structure de contrôle très pratique pour garantir la suppression de vos ressources à partir du moment où le code a fini de les utiliser. Le principe consiste à limiter la portée de la variable pointant sur la ressource à un bloc de code

déterminé par le bloc **Using…End Using**.

- Remplacez l'instruction **Dim** par **Using** pour gérer la portée de l'objet **StreamReader** comme suit :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    'Parcours du fichier à l'aide d'un StreamReader
    ~~Dim~~ Using sr As StreamReader = New StreamReader(FileName)

    End Using
    'Renvoi de la valeur de retour
    Return table
End Function
```

La variable **sr** est utilisable à l'intérieur du bloc **Using…End Using**. Au-delà, elle sera détruite automatiquement.

msdn Pour tout savoir sur la structure de contrôle **Using** :
http://msdn.microsoft.com/fr-fr/library/ms172863.aspx

Etape 3 : La troisième étape consiste à lire les lignes du fichier une à une.

La méthode de la classe **StreamReader** que nous allons utiliser pour lire une ligne de donnée texte est **ReadLine()**. Pour traiter le fichier ligne à ligne nous allons également devoir utiliser une structure de boucle. Le bloc **Do…Loop** est idéal pour balayer un nombre d'élément dont on ne connait pas le nombre.

- Ajouter le code de lecture des lignes du fichier une à une :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    'Parcours du fichier à l'aide d'un StreamReader
    Using sr As StreamReader = New StreamReader(FileName)
      'Lecture des lignes du fichier une à une jusqu'à la dernière ligne
      Do
        readRow = sr.ReadLine()
      Loop While readRow IsNot Nothing
    End Using
    'Renvoi de la valeur de retour
    Return table
End Function
```

Il existe plusieurs possibilités pour gérer la sortie d'une boucle **Do…Loop**, soit en début soit en fin de boucle.
La clause **While** positionnée en fin de boucle conditionne l'exécution d'au moins une fois du bloc de code. Si la référence **readRow** ne pointe sur aucun objet, alors la dernière ligne du fichier a été atteinte donc on peut arrêter le traitement.

msdn

Pour tout savoir sur la structure de boucle **Do…Loop** :
http://msdn.microsoft.com/fr-fr/library/eked04a7.aspx

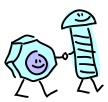
Il nous reste à programmer le traitement des lignes une à une. L'algorithme est le suivant :

- Dans le cas de la première ligne, créer la table mémoire puis ajouter la ligne à celle-ci.
- Dans le cas de toutes les autres lignes, ajouter la ligne à la table mémoire.

- Utiliser la variable **isFirstRow** (que nous avons initialisée à True) pour détecter la première ligne et créer la table mémoire dans la variable de retour **table** à l'aide de la fonction **CreateDataTable** :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    'Parcours du fichier à l'aide d'un StreamReader
    Using sr As StreamReader = New StreamReader(FileName)
      'Lecture des lignes du fichier une à une jusqu'à la dernière ligne
      Do
        readRow = sr.ReadLine()
        If isFirstRow Then
            table = CreateDataTable()
            isFirstRow = False
        End If
        'Ajouter la ligne à la table mémoire
      Loop While readRow IsNot Nothing
    End Using
    'Renvoi de la valeur de retour
    Return table
End Function
```



Attention ! Vous vous souvenez que le runtime n'apprécie pas beaucoup que vous utilisiez une référence qui ne référencie aucun objet.
Que va-t-il se passer lorsque la dernière ligne est atteinte lors du dernier passage dans la boucle ?
PAF !!!!
Une bonne pratique est de toujours tester votre référence avant de vous

lancer à l'utiliser.

- Rajoutez le test de la référence **readRow** pour le cas où la dernière ligne aura été atteinte :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
        'Définition des variables locales à la fonction
        Dim table As DataTable = Nothing
        Dim readRow As String = String.Empty
        Dim isFirstRow As Boolean = Boolean.TrueString
        'Parcours du fichier à l'aide d'un StreamReader
        Using sr As StreamReader = New StreamReader(FileName)
            'Lecture des lignes du fichier une à une jusqu'à la dernière ligne
            Do
                readRow = sr.ReadLine()
                If Not readRow Is Nothing Then
                    If isFirstRow Then
                        table = CreateDataTable()
                        isFirstRow = False
                    End If
                    'Ajouter la ligne à la table mémoire
                End If
            Loop While readRow IsNot Nothing
        End Using
        'Renvoi de la valeur de retour
        Return table
End Function
```

Etape 4 : La dernière étape est l'ajout de chacune des lignes à la table mémoire.

Qui dit nouvelle grande étape dit nouvelle procédure…

4. Créez une nouvelle procédure nommée par exemple AddRowToDataTable pour ajouter une ligne de fichier lue à la table mémoire.
    - Retrouvez la région **« Gestion de la table de données en mémoire »** que vous avez créée au point 1 de l'exercice 3.1 dans le fichier de code **Main.vb** :

```
#Region "Gestion de la table de données en mémoire"
    Function CreateDataTable ...
#End Region
```

Elle doit contenir la fonction **CreateDataTable** qui crée la structure de la table mémoire.

- Ajoutez à la suite de la fonction **CreateDataTable**, une nouvelle procédure nommée **AddRowToDataTable** comme suit :

Code VB

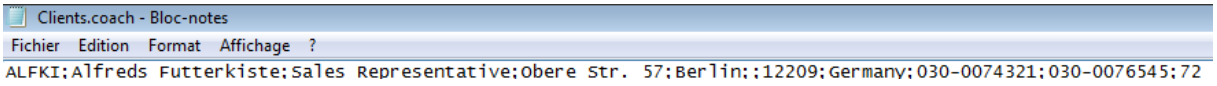
```
#Region "Gestion de la table de données en mémoire" Function CreateDataTable() As
```

```
DataTable
…
End Function
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)

End Sub
#End Region
```

Cette procédure ne retourne aucune valeur et doit récupérer en paramètres la ligne en cours lue dans le fichier ainsi que l'objet table mémoire dans lequel celle-ci doit être insérée.

Pour rappel, chaque ligne du fichier que nous avons récupérée est une chaîne de caractères comprenant un enregistrement de données où les valeurs des champs sont séparées par un point virgule.



Nous avions vu à l'atelier 3 qu'il existe de nombreuses fonctions pour traiter les chaînes de caractères, dont la fonction **Split** qui permet de découper une chaîne selon un séparateur donné et de ranger le résultat dans un tableau.

- Ajoutez le code pour découper la ligne de texte lue en tableau de chaînes de caractères en fonction du séparateur **« ; »** à l'aide de la fonction **Split** de la chaîne **ReadRow** :

Code VB

```
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)
    'Récupération de toutes les valeurs de la ligne dans un tableau
    Dim readValues As String() = ReadRow.Split(SEPARATOR_SEMICOLON)
End Sub
```

Réutiliser la constante que nous avions créée dans l'atelier 3 pour définir le séparateur.

Pour ajouter une ligne à une table de type **DataTable**, vous pouvez utiliser la méthode **NewRow** qui crée un enregistrement vide. Il suffit ensuite de renseigner chaque champ avec la valeur correspondante en utilisant le format **<LigneDeLaTableMemoire>(<IndexDeLaColonne>) = <Valeur>** puis d'ajouter l'enregistrement à la collection des lignes de la table à l'aide de la méthode **Add** de la collection.

Pour comprendre le fonctionnement de la méthode **NewRow** :
http://msdn.microsoft.com/fr-fr/library/system.data.datatable.newrow.aspx

- Créez une nouvelle ligne vide dans la table mémoire à l'aide de la méthode **NewRow** de l'objet **Table** passé en paramètres de la procédure:

Code VB

```
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)
    'Récupération de toutes les valeurs de la ligne dans un tableau
     Dim readValues As String() = ReadRow.Split(SEPARATOR_SEMICOLON)
     'Création d'un enregistrement vide dans la table mémoire
     Dim dataRow As DataRow = Table.NewRow()
End Sub
```

Comment est-ce qu'on peut parcourir le tableau des valeurs d'une ligne ? Il s'agit bien évidemment d'une structure de boucle. Lorsqu'on travaille sur la base d'un tableau ou d'une collection, le plus idéal est la structure **For Each…Next** que nous avons vu à l'atelier 3.

- Ajoutez le code de balayage du tableau de valeurs d'une ligne de données à l'aide de l'instruction **For Each** :

Code VB

```
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)
    'Récupération de toutes les valeurs de la ligne dans un tableau
     Dim readValues As String() = ReadRow.Split(SEPARATOR_SEMICOLON)
     'Création d'un enregistrement vide dans la table mémoire
     Dim dataRow As DataRow = Table.NewRow()
     'Balayage du tableau de valeurs d'une ligne de données
     For Each Value As String In readValues

     Next
End Sub
```

Pour insérer la valeur dans la colonne correspondante, nous allons utiliser un **index** et la structure de décision multiple **Switch**.

- Ajoutez la définition d'un index en début de procédure et le code de remplissage de la ligne de données dans la table :

Code VB

```
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)
     'Définition d'un index pour repérer les colonnes de valeurs
     Dim index As Integer = 0
     'Récupération de toutes les valeurs de la ligne dans un tableau
     Dim readValues As String() = ReadRow.Split(SEPARATOR_SEMICOLON)
     'Création d'un enregistrement vide dans la table mémoire
     Dim dataRow As DataRow = Table.NewRow()
     'Balayage du tableau de valeurs d'une ligne de données
     For Each Value As String In readValues
       Select Case index
         Case 0
           dataRow("Id") = Value.Trim()
         Case 1
           dataRow("Contact") = Value.Trim()
```

```
            Case 2
               dataRow("Titre") = Value.Trim()
            Case 3
               dataRow("Adresse") = Value.Trim()
            Case 4
               dataRow("Ville") = Value.Trim()
            Case 5
               dataRow("Region") = Value.Trim()
            Case 6
               dataRow("Code Postal") = Value.Trim()
            Case 7
               dataRow("Pays") = Value.Trim()
            Case 8
               dataRow("Telephone") = Value.Trim()
            Case 9
               dataRow("Telecopie") = Value.Trim()
            Case 10
               dataRow("CA") = Value.Trim()
         End Select
         index += 1
      Next
End Sub
```

A chaque valeur de la ligne on incrémente l'index pour passer à la colonne de données suivante.

Remarquez l'utilisation d'une autre fonction de traitement de chaîne de caractères, **Trim**, qui permet de supprimer tout espace blanc intempestif pouvant traîner au début et à la fin de la chaîne.

Pour en savoir plus sur la méthode **Trim** de la classe **String** :
http://msdn.microsoft.com/fr-fr/library/t97s7bs3.aspx

- Il ne vous reste plus qu'à ajouter l'enregistrement à la fin du chargement de toutes les colonnes dans la collection de lignes de données de la table **Table** :

Code VB

```
Sub AddRowToDataTable(ByVal ReadRow As String, ByVal Table As DataTable)
      'Définition d'un index pour repérer les colonnes de valeurs
      Dim index As Integer = 0
      'Récupération de toutes les valeurs de la ligne dans un tableau
      Dim readValues As String() = ReadRow.Split(SEPARATOR_SEMICOLON)
      'Création d'un enregistrement vide dans la table mémoire
      Dim dataRow As DataRow = Table.NewRow()
      'Balayage du tableau de valeurs d'une ligne de données
      For Each Value As String In readValues
         Select Case index
            …
         End Select
         index += 1
      Next
```

```
        Table.Rows.Add(dataRow)
End Sub
```

Dernière étape : Nous allons maintenant compléter chacune des procédures avec l'appel au traitement correspondant.

5. Revenez sur le code de la fonction **ReadFile** pour ajouter l'appel à la procédure **AddRowToDataTable** qui traite chaque ligne et l'ajoute à la table mémoire :

Code VB

```
Function ReadFile(ByVal FileName As String) As DataTable
    'Définition des variables locales à la fonction
    Dim table As DataTable = Nothing
    Dim readRow As String = String.Empty
    Dim isFirstRow As Boolean = Boolean.TrueString
    'Parcours du fichier à l'aide d'un StreamReader
    Using sr As StreamReader = New StreamReader(FileName)
      'Lecture des lignes du fichier une à une jusqu'à la dernière ligne
      Do
        readRow = sr.ReadLine()
        'si c'est la première ligne lue alors créer la table en mémoire
        If Not readRow Is Nothing Then
          If isFirstRow Then
            table = CreateDataTable()
            isFirstRow = False
          End If
          'Ajouter la ligne à la table mémoire
          AddRowToDataTable(readRow, table)
        End If
      Loop While readRow IsNot Nothing
    End Using
    'Renvoi de la valeur de retour
    Return table
End Function
```

6. Revenez sur le code du gestionnaire d'évènement **OuvrirToolStripMenuItem_Click** pour ajouter l'appel à la procédure **ReadFile** qui lit le fichier et renvoie la table mémoire chargée :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
```

```
        'Code d'ouverture du fichier sélectionné
        Dim newDataTable As DataTable
        newDataTable = ReadFile(openDataFileDialog.FileName)
    End If
End Sub
```

N'oubliez pas de programmer la liaison de données entre la table mémoire et les contrôles d'affichage comme dans l'exercice précédent en utilisant le gestionnaire de liaison **mainBindingSource**.

- Ajoutez le code de mise en place de la liaison de données :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        'Code d'ouverture du fichier sélectionné
        Dim newDataTable As DataTable
        newDataTable = ReadFile(openDataFileDialog.FileName)
        'Configuration du gestionnaire de liaison sur la source de données
        MainBindingSource.DataSource = newDataTable
        'Liaison du gestionnaire avec les contrôles d'affichage de données
        MainDataGridView.DataSource = MainBindingSource
        MainBindingNavigator.BindingSource = MainBindingSource
    End If
End Sub
```

Une dernière petite chose avant de tester le fonctionnement de l'ouverture d'un fichier de donnée : il est fort probable que l'utilisateur veuille réenregistrer ses modifications de données dans le même fichier au moment de la sauvegarde sur disque.
Dans cette optique, il faut donc prévoir de conserver le nom du fichier à l'ouverture de celui-ci. Nous allons donc rajouter une variable dont la portée est la classe **Main** pour y stocker le chemin complet du fichier.

- Ajouter la définition d'une variable **DataFilePath** de type **String** dans les déclarations de variable privée de la classe **Main** :

Code VB

```
Public Class Main

#Region "Déclaration des variables privées de la classe"
    'Déclaration des variables de fichier
    Dim DataFilePath As String = String.Empty
```

```
    'Déclaration des propriétés de la boîte des options
    Dim RecentFileListNumber As Short = 0
    Dim ConfirmBeforeSave As Boolean = Boolean.FalseString
    Dim SaveDirectoryPath As String = String.Empty
    Dim TraceEnabled As Boolean = Boolean.FalseString
    Dim SaveDirectoryType As DirectoryType = DirectoryType.MyDocuments
    Dim AuthorInfo() As String
    'Déclaration des énumérations
    Enum DirectoryType
        MyDocuments = 0
        Other = 1
    End Enum
    'Déclaration des constantes
    Const SEPARATOR_SEMICOLON As String = ";"
#End Region
…
End Class
```

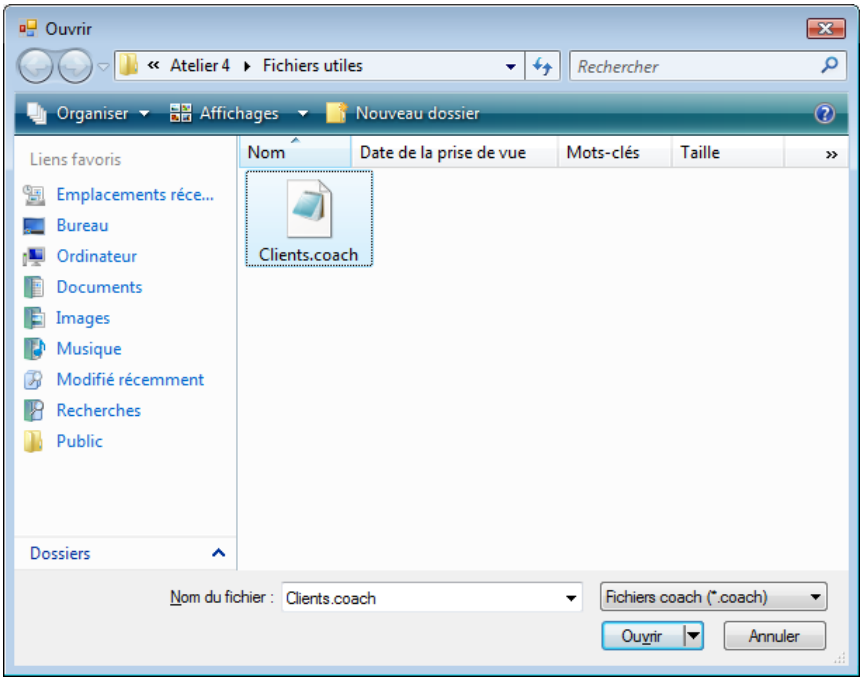
- Sauvegardez le chemin du fichier sélectionné par l'utilisateur, donné par la propriété **FileName** de la boîte de dialogue **openDataFileDialog**, dans la variable **DataFilePath** à la suite du gestionnaire d'évènement **OuvrirToolStripMenuItem_Click** :

Code VB

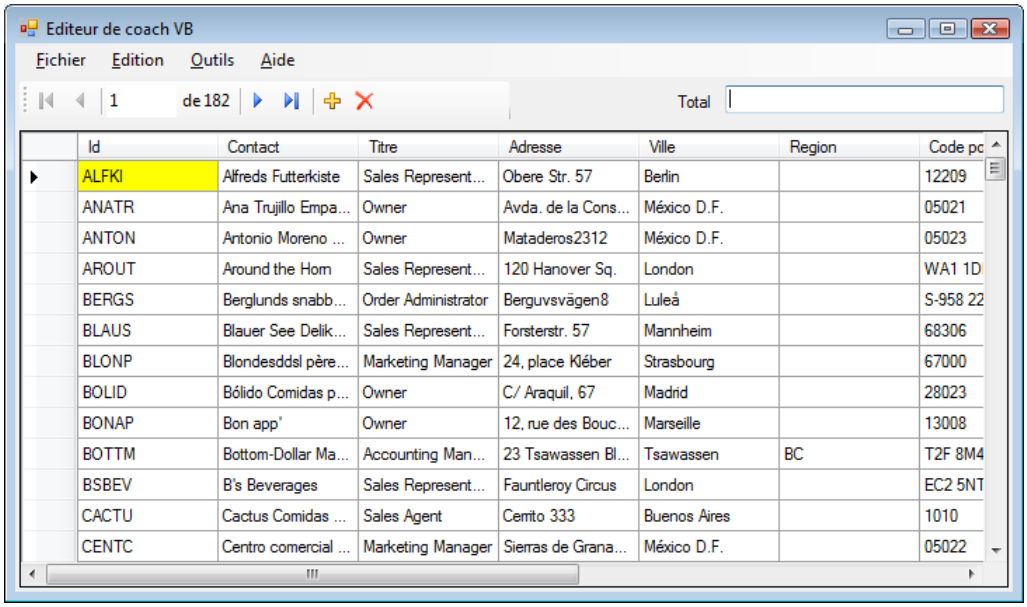
```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        'Code d'ouverture du fichier sélectionné
        Dim newDataTable As DataTable
        newDataTable = ReadFile(openDataFileDialog.FileName)
        'Configuration du gestionnaire de liaison sur la source de données
        MainBindingSource.DataSource = newDataTable
        'Liaison du gestionnaire avec les contrôles d'affichage de données
        MainDataGridView.DataSource = MainBindingSource
        MainBindingNavigator.BindingSource = MainBindingSource
        'Mémorisation du chemin du fichier pour une éventuelle sauvegarde
        DataFilePath = openDataFileDialog.FileName
    End If
End Sub
```

7. Testez le fonctionnement de l'ouverture d'un fichier de données d'extension *.coach :
    - Enregistrez tous vos changements en cliquant sur  dans la barre d'outils de Visual Studio.

- Exécutez l'application en cliquant sur (ou touche F5).
- Cliquez le menu **Fichier** > **Ouvrir**.
- Sélectionnez le fichier **Clients.coach** fourni dans le dossier **…Atelier 4\Fichiers Utiles** de cet atelier.



- Cliquez **Ouvrir**. Vous devez obtenir la grille chargée avec le jeu de données contenu dans le fichier proposé.



Id	Contact	Titre	Adresse	Ville	Region	Code pc
ALFKI	Alfreds Futterkiste	Sales Represent...	Obere Str. 57	Berlin		12209
ANATR	Ana Trujillo Empa...	Owner	Avda. de la Cons...	México D.F.		05021
ANTON	Antonio Moreno ...	Owner	Mataderos2312	México D.F.		05023
AROUT	Around the Hom	Sales Represent...	120 Hanover Sq.	London		WA1 1D
BERGS	Berglunds snabb...	Order Administrator	Berguvsvägen8	Luleå		S-958 22
BLAUS	Blauer See Delik...	Sales Represent...	Forsterstr. 57	Mannheim		68306
BLONP	Blondesddsl père...	Marketing Manager	24, place Kléber	Strasbourg		67000
BOLID	Bólido Comidas p...	Owner	C/ Araquil, 67	Madrid		28023
BONAP	Bon app'	Owner	12, rue des Bouc...	Marseille		13008
BOTTM	Bottom-Dollar Ma...	Accounting Man...	23 Tsawassen Bl...	Tsawassen	BC	T2F 8M4
BSBEV	B's Beverages	Sales Represent...	Fauntleroy Circus	London		EC2 5NT
CACTU	Cactus Comidas ...	Sales Agent	Cerrito 333	Buenos Aires		1010
CENTC	Centro comercial ...	Marketing Manager	Sierras de Grana...	México D.F.		05022

- Quitter l'application.

Bravo ! Encore un petit exercice pour apprendre à faire le processus inverse de sauvegarde des données de la table mémoire sur disque et l'éditeur fonctionne !

Une dernière bricole…
Et si vous affichiez le nom du fichier dans l'intitulé de la fenêtre de l'application (un peu comme le fait Word) ?
Pour cela, vous pouvez utiliser une classe de l'espace de noms **System.IO** très utile appelée **FileInfo** qui, comme son nom l'indique, permet de gérer des informations de fichier (dont son nom qui nous intéresse ici, le chemin du répertoire dans lequel se trouve le fichier, le chemin complet etc…). C'est une classe qui contient également des méthodes intéressantes pour copier, supprimer, déplacer un fichier etc…

- A la suite du gestionnaire d'évènement **OuvrirToolStripMenuItem_Click**, ajoutez le code de définition d'un objet de type **FileInfo** sur la base du chemin enregistré dans **DataFilePath** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        …
        'Mémorisation du chemin du fichier pour une éventuelle sauvegarde
        DataFilePath = openDataFileDialog.FileName
        'Affichage du nom du fichier dans la barre de titre du formulaire
        Dim fileInformation As IO.FileInfo = New IO.FileInfo(DataFilePath)
    End If
End Sub
```

- Utiliser la méthode **Concat** de la classe **String** pour concaténer le nom du fichier avec le titre initialement configuré pour le formulaire :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPath
    If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        …
        'Mémorisation du chemin du fichier pour une éventuelle sauvegarde
        DataFilePath = openDataFileDialog.FileName
        'Affichage du nom du fichier dans la barre de titre du formulaire
        Dim fileInformation As IO.FileInfo = New IO.FileInfo(DataFilePath)
```

```
        Me.Text = String.Concat("Editeur de coach VB", _
                                " - ", fileInformation.Name)
    End If
End Sub
```

Comme nous allons certainement avoir besoin de mettre à jour ce titre ultérieurement, par exemple lorsque l'utilisateur décide d'enregistrer ses données sous un nouveau nom de fichier, nous pouvons extraire ce code et le factoriser dans une procédure séparée.

- Ajoutez une nouvelle méthode **UpdateFormTitle** à la classe **Main** comme suit :

Code VB

```
Private Sub UpdateFormTitle(ByVal FilePath As String)
        Dim fileInformation As IO.FileInfo = New IO.FileInfo(FilePath)
        Me.Text = String.Concat("Editeur de coach VB", _
                                " - ", fileInformation.Name)
End Sub
```

Attention ! Le constructeur de l'objet **FileInfo** ne traite pas le cas d'une chaîne vide en paramètre. Or nous allons en avoir besoin pour traiter le cas d'un nouveau fichier pour lequel la barre de titre ne comprend pas de nom de fichier.

- Ajoutez un test pour le cas où le chemin de fichier n'est pas connu en utilisant la méthode **IsNullOrEmpty** de la classe **String** :

Code VB

```
Private Sub UpdateFormTitle(ByVal FilePath As String)
    If String.IsNullOrEmpty(FilePath) Then
        Me.Text = String.Concat("Editeur de coach VB")
    Else
        Dim fileInformation As IO.FileInfo = New IO.FileInfo(FilePath)
        Me.Text = String.Concat("Editeur de coach VB", _
                                " - ", fileInformation.Name)
    End If
End Sub
```

- Remplacez les lignes de la procédure **OuvrirToolStripMenuItem_Click** avec l'appel à la méthode **UpdateFormTitle** :

Code VB

```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
```

```
        openDataFileDialog.Filter = "Fichiers coach|*.coach"
        openDataFileDialog.InitialDirectory = SaveDirectoryPath
        If openDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
            …
            'Mémorisation du chemin du fichier pour une éventuelle sauvegarde
            DataFilePath = openDataFileDialog.FileName
            'Affichage du nom du fichier dans la barre de titre du formulaire
            Dim fileInformation As IO.FileInfo = New IO.FileInfo(DataFilePath)
            Me.Text = String.Concat(Me.Text, " - ", fileInformation.Name)
            UpdateFormTitle(DataFilePath)
        End If
    End Sub
```

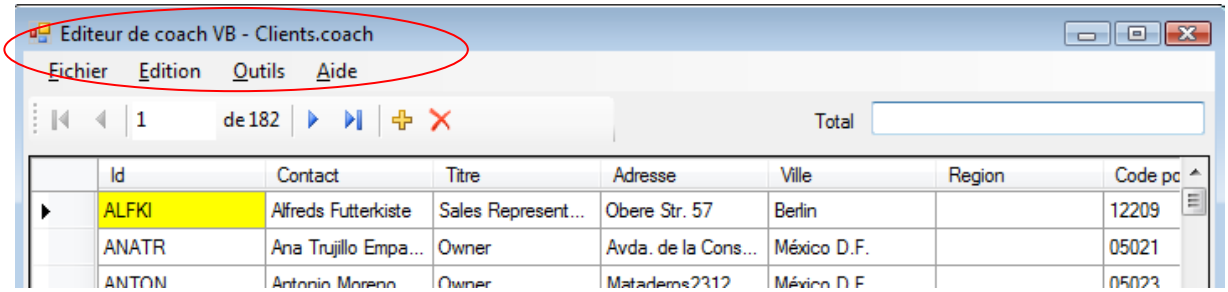
Profitons en pour tout de suite remettre à jour le titre du formulaire lorsque l'utilisateur créé un nouveau fichier pour lequel il n'existe encore aucun nom :

- Retrouvez la procédure **NouveauToolStripMenuItem_Click.**
- Ajoutez la mise à jour du chemin du fichier et du titre du formulaire comme suit :

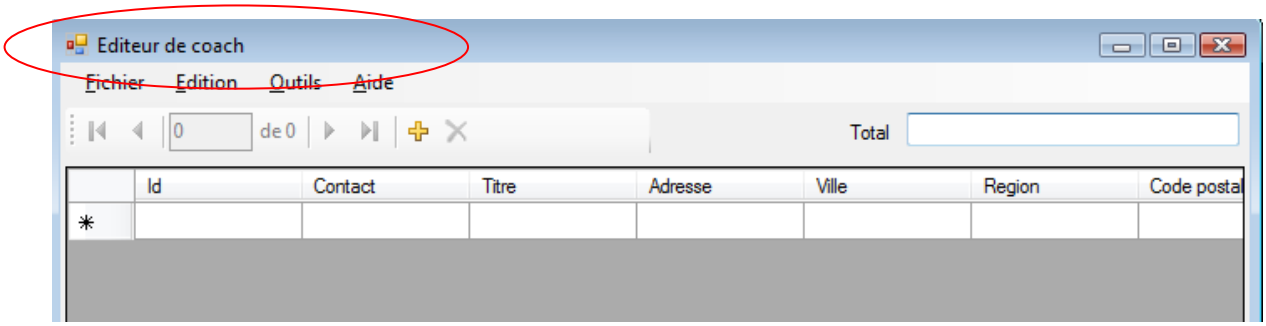
Code VB

```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
    'Configuration du gestionnaire de liaison sur la source de données
    MainBindingSource.DataSource = newDataTable
    'Liaison du gestionnaire avec les contrôles d'affichage de données
    MainDataGridView.DataSource = MainBindingSource
    MainBindingNavigator.BindingSource = MainBindingSource
    'Mettre à jour le chemin de fichier à vide
    DataFilePath = String.Empty
    'Remettre à jour le titre du formulaire
    UpdateFormTitle(DataFilePath)
  End Sub
```

8. Testez l'affichage du titre :
   - Enregistrez tous vos changements en cliquant sur dans la barre d'outils de Visual Studio.
   - Exécutez l'application en cliquant sur (ou touche F5).
   - Cliquez le menu **Fichier** > **Ouvrir**.
   - Sélectionnez le fichier **Clients.coach** fourni dans le dossier **…Atelier 4\Fichiers Utiles** de cet atelier.
   - Cliquez **Ouvrir**. Vérifiez que le nom du fichier est concaténé au titre du formulaire :



- Cliquez le menu **Fichier** > **Nouveau**. Vérifiez que la barre de titre est remise à jour :

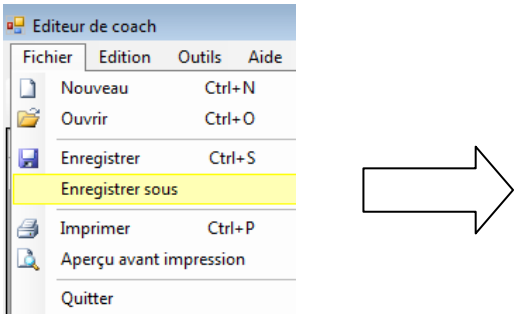


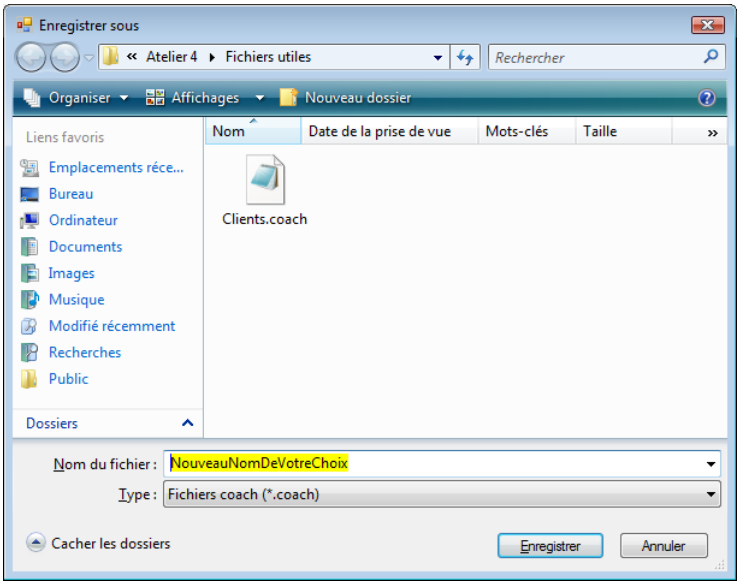
- Quitter l'application.

3.3.2 Ecrire dans le fichier

Contexte fonctionnel
L'application comporte les deux menus traditionnels **Enregistrer** et **Enregistrer sous** pour enregistrer les données :

- si l'utilisateur sélectionne le menu **Fichier** > **Enregistrer sous**, la boîte de dialogue standard **Enregistrer sous** de Windows apparaît pour qu'il choisisse un nom de fichier et le chemin de sauvegarde.





- si l'utilisateur sélectionne le menu **Fichier** > **Enregistrer**.
  Soit le nom du fichier est connu : par exemple parce que l'utilisateur travaille sur un fichier existant. Dans ce cas, la sauvegarde est réalisé sur le nom du fichier existant.
  Soit il ne l'est pas : par exemple pour un nouveau fichier. Dans ce deuxième cas, la boîte de dialogue **Enregistrer sous** de Windows doit s'afficher pour que l'utilisateur puisse déterminer le nom et le chemin du fichier de sauvegarde (on retombe sur le premier scénario).

L'algorithme à développer pour écrire dans un fichier est composé de trois étapes :

1. Sélectionner le chemin de sauvegarde et le nom du fichier (uniquement dans le cas de l'option de menu **Enregistrer sous**),
2. Balayer les enregistrements de la table mémoire un à un,
3. Ecrire chaque enregistrement sur une ligne de fichier.

Etape 1 : La première étape consiste donc à écrire le code de sélection du chemin et du fichier sur le disque.

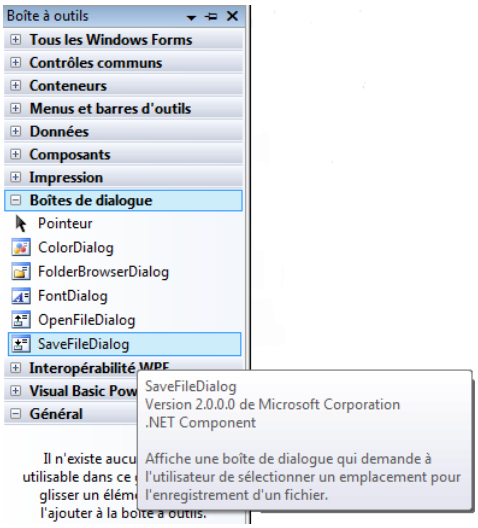
Cette étape concerne l'option de menu **Fichier** > **Enregistrer sous** du formulaire **Main**.

1. Créez un gestionnaire d'évènement associé au clic du menu Enregistrer sous :
   - Affichez le formulaire **Main** en mode **Design**.
   - Sélectionnez le menu **Fichier** puis double cliquez sur l'option de menu **Enregistrer sous** pour générer la procédure **EnregistrersousToolStripMenuItem_Click** :

```
247     Private Sub EnregistrersousToolStripMenuItem_Click(ByVal sender As System.Object, _
248                                                        ByVal e As System.EventArgs) _
249                                                        Handles EnregistrersousToolStripMenuItem.Click
250
251     End Sub
252 End Class
```

2. Utilisez la boîte de dialogue standard de sauvegarde de fichier de Windows pour permettre à l'utilisateur de faire sa sélection :

La boîte de dialogue qui nous intéresse cette fois-ci est **SaveFileDialog**. Elle est utilisable comme précédemment à partir de la Boîte à outils ou directement via la classe correspondante du Framework .NET.



- Ajoutez à la procédure le code de déclaration et d'instanciation d'un nouvel objet de type **SaveFileDialog** :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                            ByVal sender As System.Object, _
                            ByVal e As System.EventArgs) _
                            Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog As SaveFileDialog = New SaveFileDialog()
End Sub
```

Les principales propriétés à configurer sur cet objet sont les mêmes que celles de la boîte **OpenFileDialog**, à savoir : **Filter** et **InitialDirectory**.

- Complétez le code pour filtrer le type de fichier sur l'extension ***.coach** :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                            ByVal sender As System.Object, _
                            ByVal e As System.EventArgs) _
                            Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog As SaveFileDialog = New SaveFileDialog()
    saveDataFileDialog.Filter = "Fichiers coach|*.coach"
End Sub
```

- Configurez le répertoire par défaut sur le dossier préférentiel de l'utilisateur configuré à l'aide de la fenêtre **Options** à l'atelier 3 dans la variable privée **SaveDirectoryPath** de la classe **Main** :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                            ByVal sender As System.Object, _
                            ByVal e As System.EventArgs) _
                            Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog As SaveFileDialog = New SaveFileDialog()
    saveDataFileDialog.Filter = "Fichiers coach|*.coach"
```

```
        saveDataFileDialog.InitialDirectory = SaveDirectoryPath
End Sub
```

Pour éviter de taper à chaque ligne le nom relativement long de notre objet de type **SaveFileDialog**, une bonne pratique est d'utiliser la structure **With…End With** que nous avons déjà rencontrée dans ce tutorial.

- Ajoutez un bloc **With…End With** pour simplifier l'écriture du code :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                    ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog As SaveFileDialog = New SaveFileDialog()
    With saveDataFileDialog
        saveDataFileDialog.Filter = "Fichiers coach|*.coach"
        saveDataFileDialog.InitialDirectory = SaveDirectoryPath
    End With

  End Sub
End Class
```

Avec la toute dernière version du langage Visual Basic fournie avec Visual Studio 2008, il existe même une écriture encore plus concise qui permet de déclarer, instancier et initialiser les propriétés de l'objet en une seule instruction.

- Retransformez le code de la façon suivante :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                    ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog = New SaveFileDialog() With _
                {.Filter = "Fichiers coach|*.coach", _
                 .InitialDirectory = SaveDirectoryPath}
  End Sub
End Class
```

Deux remarques :

- Le mot clé **As** qui indique le type de données de la variable a disparu ! Comment le système va-t-il connaître le type de l'objet si on omet de le préciser ?
  C'est une nouveauté du langage ☺. On parle d'**Inférence de type avec les types nommés**. Le type de donnée de la variable est déduit du type de l'objet créé par l'assignation (c'est-à-dire par ce qui se trouve à droite du signe **=**. Ici, il s'agit du constructeur de la classe).

- Pour cette nouvelle écriture plus concise utilisant le mot clé **With**, on parle d'**Initialiseur d'objet**.

Pour tout savoir sur les **initialiseurs d'objets** de Visual Basic 2008: http://msdn.microsoft.com/fr-fr/library/bb385125.aspx

- Affichez maintenant la boîte de dialogue avec la méthode **ShowDialog** et testez le code de fermeture avec une condition **If…Then…End If** :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog = New SaveFileDialog() With _
                                        {.Filter = "Fichiers coach|*.coach", _
                                         .InitialDirectory = SaveDirectoryPath}
    If saveDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then

    End If
End Sub
```

- Récupérez le chemin du fichier à partir de la propriété **FileName** de l'objet **saveDataFileDialog** et stockez-le dans la variable **DataFilePath** :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog = New SaveFileDialog() With _
                                        {.Filter = "Fichiers coach|*.coach", _
                                         .InitialDirectory = SaveDirectoryPath}
    If saveDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        'Sauvegarde du chemin du fichier sélectionné
        DataFilePath = saveDataFileDialog.FileName
    End If
End Sub
```

- Mettez à jour le nouveau nom de fichier dans l'intitulé du formulaire à l'aide de la méthode **UpdateFormTitle** créé à l'exercice précédent :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog = New SaveFileDialog() With _
                                        {.Filter = "Fichiers coach|*.coach", _
                                         .InitialDirectory = SaveDirectoryPath}
    If saveDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
```

```
            'Sauvegarde du chemin du fichier sélectionné
            DataFilePath = saveDataFileDialog.FileName
            'Mise-à-jour du titre du formulaire avec le nouveau nom de fichier
            UpdateFormTitle(DataFilePath)
        End If
End Sub
```

Etapes 2 et 3 : Les deux autres étapes consistent à sauvegarder les données de la grille dans le fichier.

3. Ajoutez une méthode à la classe Main dont l'objectif est de sauvegarder les données dans un fichier :
   - Retrouvez la région de code **Traitement des fichiers de données** contenant déjà la fonction **ReadFile** :

```
#Region "Traitement des fichiers de données"
    Function ReadFile ...
#End Region
```

   - Juste après la fonction **ReadFile**, ajoutez la définition d'une nouvelle procédure **WriteFile** comme suit :

Code VB

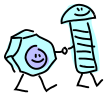
```
Sub WriteFile(ByVal FileName As String)
End Sub
```

Notez que la procédure a évidemment besoin du chemin du fichier qu'il lui faudra ouvrir. C'est l'objet du paramètre **FileName**.
En revanche, nous n'avons pas besoin de valeur de retour ici.

Pour lire le fichier, nous avons eu besoin d'un objet **StreamReader** qui faisait office de *lecteur*. Pour écrire dans le fichier nous allons avoir besoin à l'inverse d'un *écrivain*, c'est-à-dire un objet ... ? **StreamWriter** bien sûr !

C'est quoi un **StreamWriter** ?
C'est un type d'objet qui *écrit* des caractères à partir d'un flux d'octets dans un codage particulier. Tout comme le **StreamReader**, il appartient à l'espace de nommage **System.IO**. La méthode que nous allons utilisée sur cet objet pour écrire une ligne de données texte est **WriteLine** (on s'en serait douté puisque c'était **ReadLine** pour l'opération inverse avec le **StreamReader**).

Pour en savoir plus sur la classe **StreamWriter** :
http://msdn.microsoft.com/fr-fr/library/system.io.streamwriter.aspx



N'oublions pas la structure de contrôle **Using…End Using** pour maitriser la durée de vie de notre objet !

- Ajoutez la déclaration d'un objet de type **StreamWriter** comme suit :

Code VB

```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)

    End Using
End Sub
```

Plus besoin de s'embêter avec la clause **As** ☺ !

Maintenant il suffit de récupérer les données et de les charger enregistrement par enregistrement dans le fichier.

Mais où sont les données au fait ?
Tout simplement dans la propriété **DataSouce** de l'objet gestionnaire de liaison **mainBindingSource**. C'est lui qui connait la source de données.

En effet, aussi bien lors de la création d'un nouveau fichier que lors de l'ouverture d'un fichier existant, nous avons écrit respectivement les lignes :

```
Private Sub NouveauToolStripMenuItem_Click(ByVal sender As System.Object, _
                                           ByVal e As System.EventArgs) _
                                           Handles NouveauToolStripMenuItem.Click
    'Création d'une nouvelle table mémoire vide
    Dim newDataTable As DataTable
    newDataTable = CreateDataTable()
    'Configuration du gestionnaire de liaison sur la source de données
    MainBindingSource.DataSource = newDataTable
```

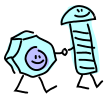
Création d'une table vide

Et :

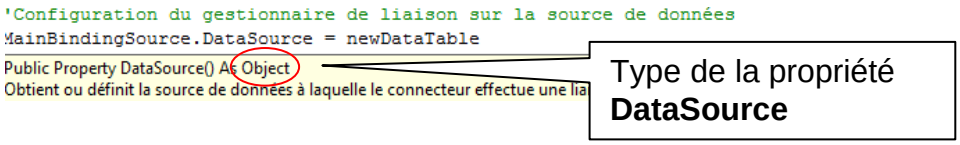
```
Private Sub OuvrirToolStripMenuItem_Click(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                          Handles OuvrirToolStripMenuItem.Click
    'Sélection du fichier à l'aide de la boîte de dialogue d'ouverture standard de Windows
    Dim openDataFileDialog As OpenFileDialog = New OpenFileDialog()
    openDataFileDialog.Filter = "Fichiers coach|*.coach"
    openDataFileDialog.InitialDirectory = SaveDirectoryPa
    If openDataFileDialog.ShowDialog = Windows.Forms.Dial
        'Code d'ouverture du fichier sélectionné
        Dim newDataTable As DataTable
        newDataTable = ReadFile(openDataFileDialog.FileName)
        'Configuration du gestionnaire de liaison sur la source de données
        MainBindingSource.DataSource = newDataTable
```

Création d'une table à partir d'un fichier existant

DataSource contient donc un objet de type **DataTable**, qui a une propriété **Rows** pointant sur la collection de ligne de la table de données. C'est ce qu'il nous faut sauvegarder !



Au passage, si dans l'une des deux procédures vous positionnez le curseur sur la propriété **DataSource**, une aide rapide apparaît vous indiquant entre autre le type de la propriété **DataSource** :



DataSource est donc de type **Object**.

Que représente le type de données **Object** ?
Il s'agit du type de données universel dans le Framework .NET et VB, ce qui signifie que tous les autres types de données sont dérivés de lui. Il est donc très utile pour typer une variable destinée à recevoir des données de différents types.

La propriété **DataSource** est de type **Object** car elle est destinée à configurer ou obtenir non seulement des tables de données (**DataTable**), mais aussi tout objet pouvant fournir des données (par exemple un **DataSet** que nous aurons l'occasion d'utiliser dans le prochain atelier). Donc le Framework ne sait pas à l'avance le type de données qu'il devra stocker.

Pour tout savoir sur le type **Object** :
http://msdn.microsoft.com/fr-fr/library/3cf69sdx.aspx

- Ajouter le code de balayage de la collection **Rows** des lignes de la table **DataTable** stockée dans la propriété **DataSource** à l'aide d'une structure de bouclage **For Each**:

Code VB

```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)
        For Each Row In MainBindingSource.DataSource.Rows

        Next
    End Using
End Sub
```

Avez-vous remarqué que l'on a omis de définir le type de la variable **Row** avec le mot clé **As** ?
C'est toujours cette même fonctionnalité d'**inférence de type** du langage que l'on utilise. Le compilateur va déduire le type de la variable **Row** à partir du type de la collection d'objets définie après **In**.

Pour tout savoir sur l'**inférence de type local** :
http://msdn.microsoft.com/fr-fr/library/bb384937.aspx

Avez-vous remarqué également que l'IntelliSense de Visual Studio ne vous propose pas la propriété **Rows** dans la liste des membres de l'objet **DataSource** ?

```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)
        For Each Row In MainBindingSource.DataSource.

        Next
    End Using
End Sub
d Region
```

Equals
GetHashCode
GetType
ReferenceEquals
ToString

En même temps, pourquoi le ferait-elle puisque Visual Studio ne connaît pas encore le type d'objet stocké dans **DataSource** ? Seul le type **DataTable** contient un membre **Rows**, pas le type **Object**…

Mais alors comment indiquer qu'un objet a un type précis ?
Visual Basic propose des instructions et des méthodes de conversion d'un type en un autre. Nous en avons vu quelques unes dans l'atelier 3 précédent. Si la conversion est possible, alors la valeur retournée est du type attendu.

Comment indiquer une conversion (explicite) dans le cas d'un *objet* ?
La fonction **CType** de Visual Basic effectue la conversion d'un type objet en un autre type de données. Il suffit de lui indiquer l'objet à convertir et le type de données cible en paramètres, et elle renvoie la donnée convertie.

Pour convertir un objet en un autre type dans Visual Basic :
http://msdn.microsoft.com/fr-fr/library/x53d8wxa.aspx
Pour vous documenter sur la conversion de types de données en Visual Basic :
http://msdn.microsoft.com/fr-fr/library/hcb26cc8.aspx

- Ajoutez la conversion de la propriété **DataSource** de type **Object** en type **DataTable** à l'aide de la fonction **CType**, avant de procéder au balayage de la collection **Rows** :

Code VB

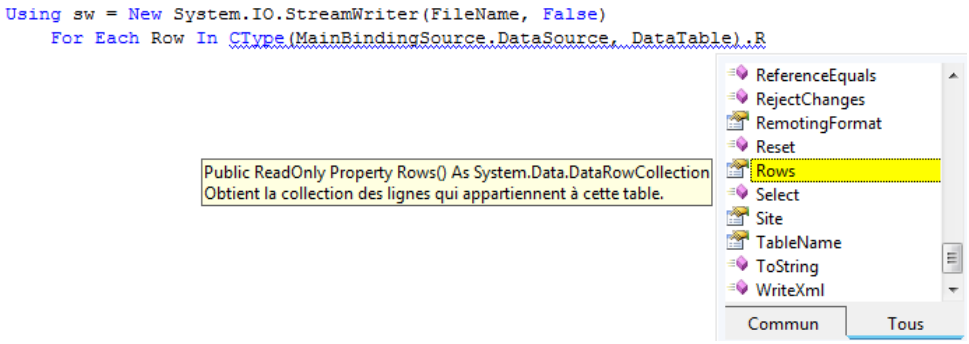
```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)
        For Each Row In CType(MainBindingSource.DataSource,DataTable).Rows

        Next
    End Using
End Sub
```

Elément à convertir

Type de données ciblé

L'IntelliSense s'y retrouve tout de suite beaucoup mieux lorsque vous cherchez à utiliser une propriété **Rows** sur l'objet retourné par la fonction **CType** !



En même temps, vous n'étiez pas obligé de faire une conversion. Se produit alors ce qu'on appelle une *liaison tardive* c'est-à-dire que le typage se produit au moment de l'exécution. Le processus est évidemment dangereux et plus couteux.

Pour comparer **la liaison tardive** à **la liaison anticipée** :
http://msdn.microsoft.com/fr-fr/library/0tcf61s1.aspx

Pour construire une ligne du fichier à partir de la ligne de données de la table mémoire, il faut concaténer à la suite les valeurs de chaque colonne de la ligne séparées par le séparateur (point virgule).

Comment allons-nous accéder à la valeur des colonnes d'une ligne de la table de données ?
La propriété **Rows** du type **DataTable** est une collection de ligne de données (**DataRow**). Pour accéder à la valeur d'une colonne (champ) de la ligne, il faut indiquer entre parenthèses, le nom du champ de données voulu. Par exemple, **Row("Id")** renvoie la valeur de l'**Id** de la ligne courante.

- Utilisez la méthode **WriteLine** de l'objet **sw** pour écrire un enregistrement de données dans le fichier. Utilisez la méthode **Concat** de la classe **String** pour concaténer chacune des colonnes de la ligne séparées par le point virgule donné par la constante **SEPARATOR_SEMICOLON** :

Code VB

```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)
        For Each Row In CType(MainBindingSource.DataSource,DataTable).Rows
            sw.WriteLine(String.Concat( _
                Row("Id"), SEPARATOR_SEMICOLON _
                , Row("Contact"), SEPARATOR_SEMICOLON _
                , Row("Titre"), SEPARATOR_SEMICOLON _
                , Row("Adresse"), SEPARATOR_SEMICOLON _
                , Row("Ville"), SEPARATOR_SEMICOLON _
                , Row("Region"), SEPARATOR_SEMICOLON _
                , Row("Code Postal"), SEPARATOR_SEMICOLON _
                , Row("Pays"), SEPARATOR_SEMICOLON _
```

```
                , Row("Telephone"), SEPARATOR_SEMICOLON _
                , Row("Telecopie"), SEPARATOR_SEMICOLON _
                , Row("CA")))
        Next
    End Using
End Sub
```

Cette fois, on tient le bon bout !
Il reste une toute petite précaution à prendre avec le type de la colonne **CA** qui est **Integer** (alors que toutes les autres colonnes sont de type **String**).
En effet, si la valeur du CA est nulle dans la grille de données, c'est-à-dire qu'aucune valeur n'a été saisie par l'utilisateur, la donnée insérée dans le fichier est une chaîne vide.
Rien de bien gênant à l'écriture dans le fichier, sauf qu'à la relecture du fichier dans notre Editeur, les choses se gâtent et (aucune conversion n'ayant été prévu dans le sens inverse) on récupère une erreur d'exécution.

Comment repère-t-on une donnée de valeur nulle ?
Grâce au Framework .NET (comme d'hab), qui fournit une classe appelée **DBNull** dans l'espace de noms **System** pour indiquer l'absence de valeur d'une information d'une source de données.

- Modifiez le code pour traiter le cas où aucune valeur n'a été saisi dans la colonne de données chiffre d'affaires (CA) :

Code VB

```
Sub WriteFile(ByVal FileName As String)
    Using sw = New System.IO.StreamWriter(FileName, False)
        For Each Row In CType(MainBindingSource.DataSource,DataTable).Rows
            Dim chiffreAffaire As String
            If Row("CA") Is System.DBNull.Value Then
                chiffreAffaire = "0"
            Else
                chiffreAffaire = Row("CA").ToString()
            End If
            sw.WriteLine(String.Concat( _
                    Row("Id"), SEPARATOR_SEMICOLON _
                    , Row("Contact"), SEPARATOR_SEMICOLON _
                    , Row("Titre"), SEPARATOR_SEMICOLON _
                    , Row("Adresse"), SEPARATOR_SEMICOLON _
                    , Row("Ville"), SEPARATOR_SEMICOLON _
                    , Row("Region"), SEPARATOR_SEMICOLON _
                    , Row("Code Postal"), SEPARATOR_SEMICOLON _
                    , Row("Pays"), SEPARATOR_SEMICOLON _
                    , Row("Telephone"), SEPARATOR_SEMICOLON _
                    , Row("Telecopie"), SEPARATOR_SEMICOLON _
                    , chiffreAffaire))
```

```
        Next
    End Using
End Sub
```

Dernière étape : Nous allons maintenant compléter le gestionnaire d'évènement de l'option **Enregistrer sous** avec l'appel au traitement correspondant.

4. Revenez sur le code de la procédure **EnregistrersousToolStripMenuItem_Click** pour ajouter l'appel à la procédure **WriteFile** qui traite chaque ligne de la table mémoire et l'ajoute au fichier :

Code VB

```
Private Sub EnregistrersousToolStripMenuItem_Click( _
                                    ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles EnregistrersousToolStripMenuItem.Click
    Dim saveDataFileDialog = New SaveFileDialog() With _
                                        {.Filter = "Fichiers coach|*.coach", _
                                         .InitialDirectory = SaveDirectoryPath}
    If saveDataFileDialog.ShowDialog = Windows.Forms.DialogResult.OK Then
        'Sauvegarde du chemin du fichier sélectionné par l'utilisateur
        DataFilePath = saveDataFileDialog.FileName
        'Mise-à-jour du titre du formulaire avec le nouveau nom de fichier
        UpdateFormTitle(DataFilePath)
        'Ecriture des données dans le fichier
        WriteFile(DataFilePath)
    End If
End Sub
```

Quel est l'algorithme derrière l'autre option de menu **Fichier** > **Enregistrer** ?
Rigoureusement le même que celui que nous venons de suivre, à ceci près qu'il faut vérifier qu'il ne s'agit pas d'un nouveau fichier auquel cas vous ne connaissez pas le nom ni le chemin du fichier de sauvegarde. Mais qu'à cela ne tienne, il suffit alors de re router l'utilisateur sur la procédure **Enregistrer Sous** précédente.

5. Coder l'option de menu Enregistrer :
   - Affichez le formulaire **Main** en mode **Design**.
   - Sélectionnez le menu **Fichier** puis double cliquez sur l'option de menu **Enregistrer** pour générer la procédure **EnregistrerToolStripMenuItem_Click** :

Code VB

```
Private Sub EnregistrerToolStripMenuItem_Click( _
                                    ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles EnregistrerToolStripMenuItem.Click

End Sub
```

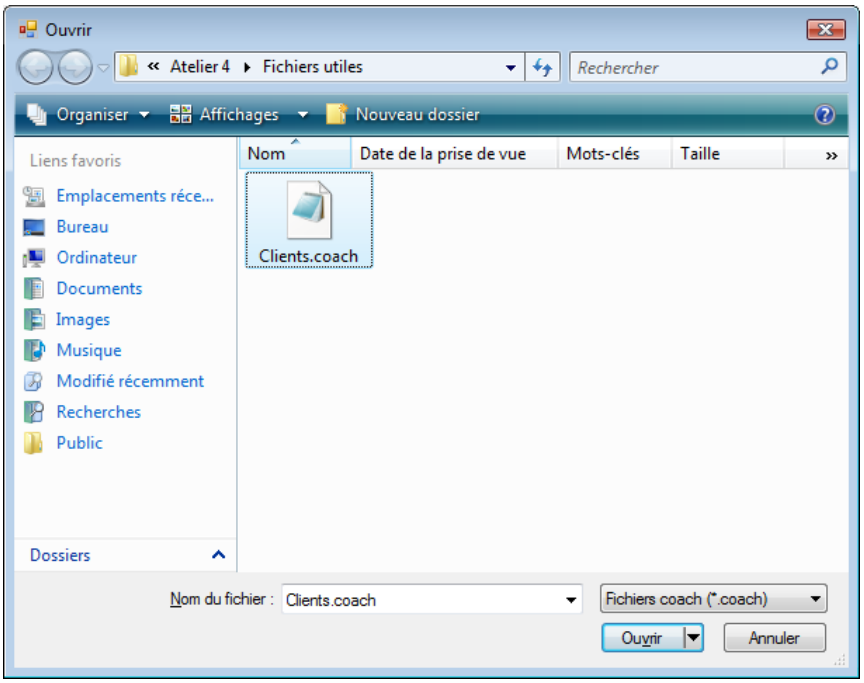
- Ajoutez le test du nom de fichier. S'il est vide, invoquer le gestionnaire d'évènement **EnregistrerSousToolStripMenuItem_Click** pour exécuter la procédure précédente :

Code VB

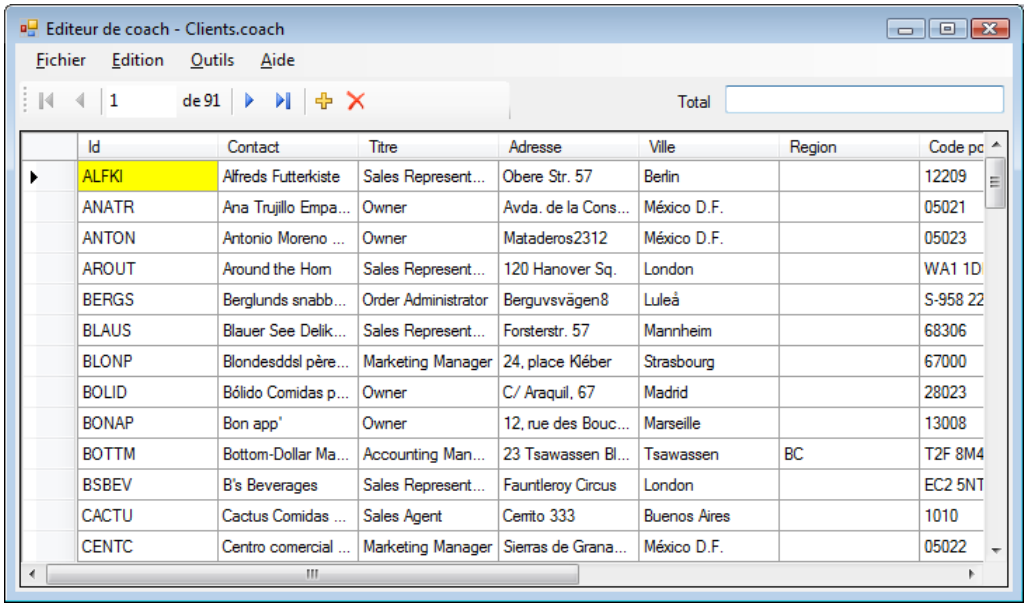
```
Private Sub EnregistrerToolStripMenuItem_Click( _
                                    ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles EnregistrerToolStripMenuItem.Click
    If String.IsNullOrEmpty(DataFilePath) Then
       'Il s'agit d'un nouveau fichier. Déclencher la proc. Enregistrer Sous
       EnregistrersousToolStripMenuItem_Click(sender, e)
    Else
       'Le nom du fichier est connu. Sauvegarder les donne
       WriteFile(DataFilePath)
    End If
End Sub
```

Retransmettez au gestionnaire d'évènement les mêmes paramètres que le gestionnaire en cours.

6. Testez le fonctionnement de l'enregistrement des données dans un fichier de données d'extension *.coach :
   - Enregistrez tous vos changements en cliquant sur dans la barre d'outils de Visual Studio.
   - Exécutez l'application en cliquant sur (ou touche F5).
   - Cliquez le menu **Fichier** > **Ouvrir**.
   - Sélectionnez le fichier **Clients.coach** fourni dans le dossier **…Atelier 4\Fichiers Utiles** de cet atelier.



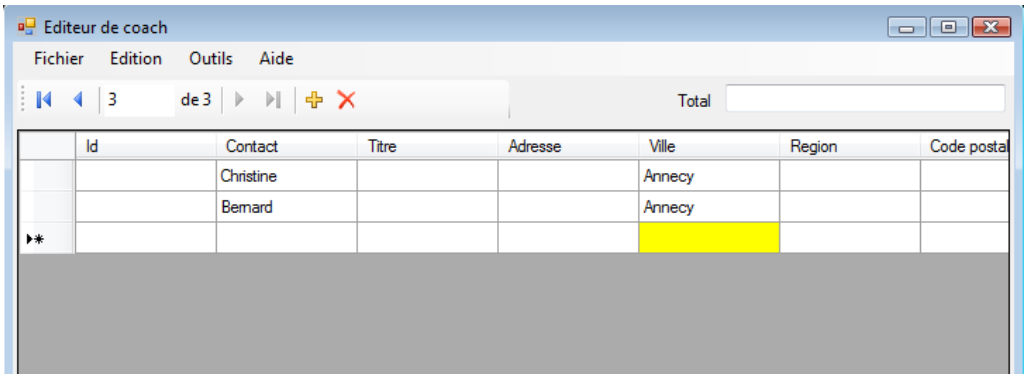
- Cliquez **Ouvrir**.



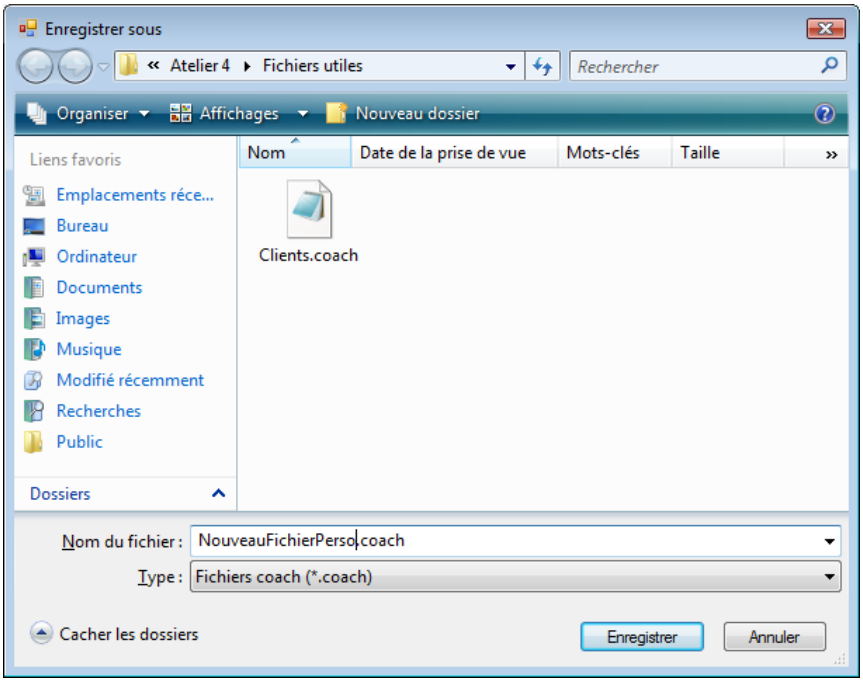
- Modifiez une ou plusieurs données dans la grille de données.
- Cliquez le menu **Fichier** > **Enregistrer**.
- Cliquez ensuite **Fichier** > **Nouveau** puis ré ouvrez le même fichier en cliquant le menu **Fichier** > **Ouvrir** comme précédemment.

Vérifiez que le fichier s'ouvre avec les valeurs enregistrées précédemment !

- Créez un nouveau fichier en cliquant **Fichier** > **Nouveau**.
- Saisissez une ou deux lignes de données.



- Cliquez **Fichier** > **Enregistrer**. Comme il s'agit d'un nouveau fichier, la boîte **Enregistrer sous** doit apparaître :



- Enregistrer le fichier avec un nouveau nom puis ré ouvrez-le pour vérifier que la sauvegarde s'est bien passée.
- Quitter l'application.

Comme pour la lecture, vous pouvez aussi utiliser les extraits de code et travailler sur la base de l'objet **My.Computer.FileSystem**.
Pour vous documenter sur l'écriture dans un fichier en Visual Basic en utilisant l'objet **My.Computer.FileSystem** :
http://msdn.microsoft.com/fr-fr/library/c520bdhb.aspx

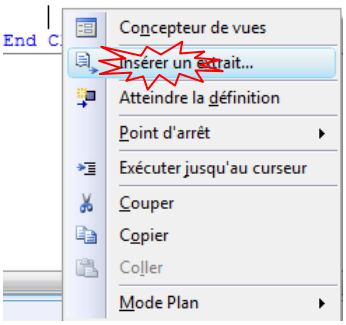
4 Pour aller plus loin…

4.1 L'objet My.Computer.FileSystem

Vous vous souvenez des objets **My.Application**, **My.Computer**, **My.Log** etc. de Visual Basic que nous avons déjà rencontrés à plusieurs reprises ?
Il se trouve que l'objet **My.Computer** comporte un objet **FileSytem** qui propose des raccourcis super simples pour manipuler des fichiers.
Plus d'histoire de flux de données et tout le tremblement… Par exemple, pour lire un fichier il suffit d'invoquer la méthode **ReadAllText** de l'objet et le tour est joué !

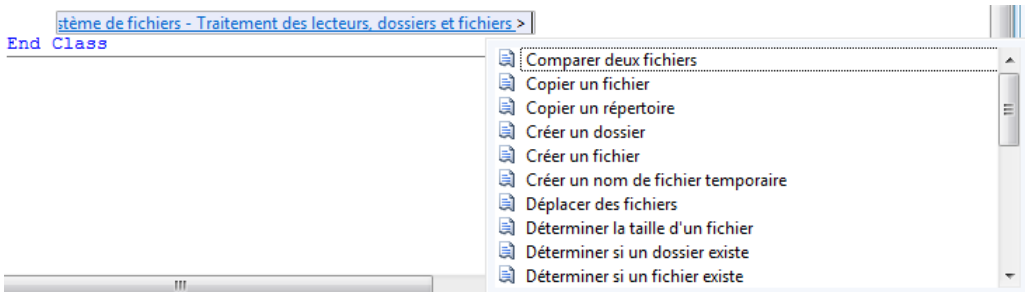
C'est d'ailleurs ce même objet **My.Computer.FileSystem** que vous retrouvez si vous utilisez les extraits de code proposés par Visual Studio :

- Faites un **clic droit** juste au dessus de **End Class** dans la classe **Main**.
- Cliquez **Insérer un extrait**.



- Double cliquez **Notions de base – Collections, types de données, système de fichiers, mathématiques** puis **Système de fichiers – Traitement des lecteurs, dossiers et fichiers**.

Vous tombez sur une batterie de codes d'exemple de manipulation des fichiers en tout genre, basé sur l'objet **My.Computer.FileSystem**.



Attention toutefois ! L'objet **My** est propre au langage Visual Basic alors que la méthode qui utilise les flux de données **Stream** du Framework.NET est applicable dans les autres langages tels que le C#.

Pour vous documenter sur la lecture à partir d'un fichier en Visual Basic en utilisant l'objet **My.Computer.FileSystem** :
http://msdn.microsoft.com/fr-fr/library/wz100x8w.aspx
Pour vous documenter sur l'écriture dans un fichier en Visual Basic en utilisant l'objet **My.Computer.FileSystem** :
http://msdn.microsoft.com/fr-fr/library/c520bdhb.aspx

4.2 Evènements de fichier

Il existe un composant nommé **FileSystemWatcher** qui permet de surveiller les modifications effectuées dans les fichiers et répertoires du système ou d'un ordinateur. Lorsque des modifications ont lieu, un ou plusieurs évènements sont déclenchés et transmis au composant. En écrivant des gestionnaires pour ces évènements, imaginez les scénarios auxquels vous pouvez répondre…

Pour consulter une introduction sur le sujet :
http://msdn.microsoft.com/fr-fr/library/ch2s8yd7.aspx
Pour tout savoir sur le sujet :
http://msdn.microsoft.com/fr-fr/library/342da5th.aspx