



Unité	Développer des composants d'interface				U03
Etape	Programmer des formulaires et des états				E03
Compétence C2		Séance	S02	Activité	A-004

Cette activité vous permettra de travailler sur la mise en place de contrôles de gestion de listes d'éléments.

## Sommaire :

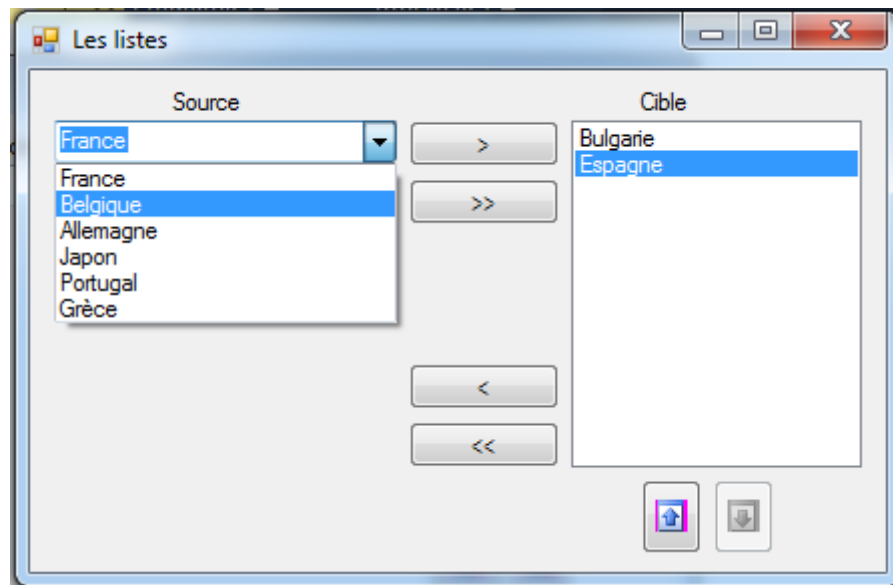
1	Gestion de listes .....	2
1.1	But du traitement .....	2
1.2	Les traitements.....	2
1.3	Etape 1 : design graphique.....	3
1.4	Etape 2 : peuplement de la ComboBox par programme .....	3
1.5	Etape 3 : transfert d'un ou de tous les éléments par les boutons .....	3
1.6	Etape 4 : transfert d'un élément.....	3
1.7	Etape 5 : gérer l'état des boutons de transfert. ....	4
1.8	Etape 6 : saisie d'un nouvel élément en liste source .....	4
1.9	Etape 7 : gérer le repositionnement dans la ComboBox .....	5
1.10	Etape 8 : gestion des boutons Haut et Bas .....	5

# 1 Gestion de listes

*Pour réaliser cet exercice, étudiez la documentation mise à votre disposition sur métis (les contrôles de liste) et la documentation de référence en ligne sur les contrôles graphiques **ListBox** et **ComboBox***







## 1.1 But du traitement

Gérer le contenu d'une liste, le passage d'informations entre listes (ici ComboBox et ListBox), élément par élément, ou l'ensemble des éléments en une seule opération, reclasser les éléments sélectionnés.



## 1.2 Les traitements

Les informations sont chargées dans la Combobox Source au démarrage de l'application. Tout élément saisi dans la partie texte de la Source peut être ajouté en cours de traitement sur l'évènement **DropDown**, s'il n'existe pas déjà ni dans la liste Cible, ni dans la ComboBox Source.

- Un click sur le bouton  ajoute l'élément sélectionné depuis la ComboBox Source dans la ListBox Cible ; l'élément est supprimé de la Source. Ce bouton est désactivé si aucun élément de la Source n'est sélectionné.
- Un click sur le bouton  ajoute tous les éléments présents de la Source dans la Cible ; tous les éléments sont supprimés de la Source. Ce bouton est désactivé si la Source est vide.
- Un click sur le bouton  ajoute l'élément sélectionné de la Cible dans la Source ; l'élément est supprimé de la Cible. Ce bouton est désactivé si aucun élément de la liste Cible n'est sélectionné.
- Un click sur le bouton  ajoute tous les éléments de la Cible dans la Source ; tous les éléments sont supprimés de la Cible. Ce bouton est désactivé si la Cible est vide.
- Un click sur le bouton  déplace l'élément sélectionné de la Cible d'une ligne vers le haut. Ce bouton est désactivé si aucun élément de la liste Cible n'est sélectionné, ou si le premier élément est sélectionné.
- Un click sur le bouton  déplace l'élément sélectionné de la Cible d'une ligne vers le bas. Ce bouton est désactivé si aucun élément de la liste Cible n'est sélectionné, ou si le dernier élément est sélectionné.

## 1.3 Etape 1 : design graphique

Placer en partie gauche un contrôle ComboBox nommé `cbxSource` (à ce stade, il n'affiche qu'une ligne).

Placer en partie droite une ListBox nommée `lstCible` et l'ajuster à la dimension voulue.

Placer les 4 boutons de transfert, nommés `btnAjouter`, `btnAjouterTout`, `btnSupprimer` et `btnSupprimerTout`.

Placer les 2 boutons de classement et définir leur propriété Image pour afficher une flèche graphique (au choix, par exemple dans les ressources fournies avec Visual Studio ou trouvées en ligne sur Internet). Nommer ces boutons `btnHaut` et `btnBas`.

## 1.4 Etape 2 : peuplement de la ComboBox par programme

En .Net, les éléments d'un contrôle de liste sont des objets dont les références sont mémorisées dans une « collection » gérée par la propriété `Items` du contrôle. Le designer graphique de Visual Studio permet de saisir des valeurs initiales directement dans la fenêtre de propriétés. Ceci génère du code c#. Vous pouvez vous en inspirer pour créer une méthode d'initialisation qui permettra de produire des valeurs dans la comboBox en utilisant la méthode `Add()` ou `AddRange()` des collections !

Vous pouvez ainsi définir une méthode d'initialisation `Init()` permettant d'initialiser cette liste et appelée par le « constructeur » du formulaire.

## 1.5 Etape 3 : transfert d'un ou de tous les éléments par les boutons

En mode Création, double-cliquer sur le bouton `btnAjoute` de manière à générer le prototype de méthode événementielle correspondant à l'événement par défaut `Click`.

Il s'agit ici *d'ajouter* (méthode `Add()`) *l'élément sélectionné* dans la ComboBox (propriété `SelectedItem`), puis de *le retirer* de la collection source (méthode `Remove()`).

Pour ajouter tous les éléments de la collection, utiliser l'instruction (bien pratique) de parcours de liste `foreach() {}` de manière à répéter ce traitement élémentaire autant de fois que nécessaire. Vous pouvez alors utiliser la méthode `Clear()` pour supprimer tous les éléments de la collection.

## 1.6 Etape 4 : transfert d'un élément

La logique reste la même que ci-dessus et les contrôles ListBox et ComboBox disposent des mêmes propriétés et méthodes ; à vous de transposer le code.

## 1.7 Etape 5 : gérer l'état des boutons de transfert.

Il faut maintenant sécuriser l'application en activant et désactivant les boutons de transfert selon le contexte.

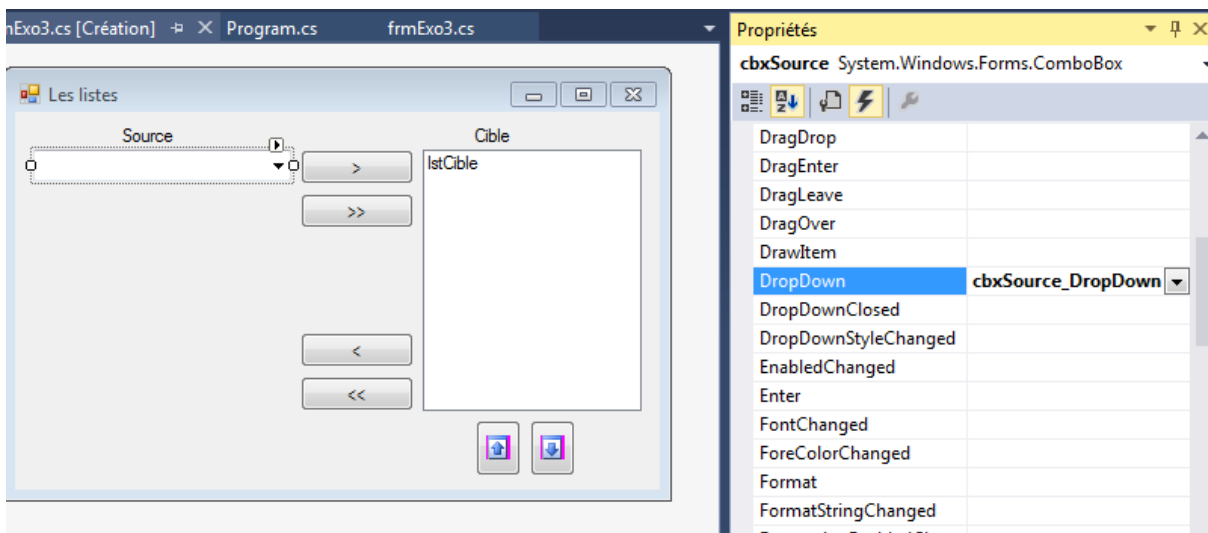
Analysons les cas :

- Le bouton `bntAjouteTout` reste disponible tant qu'il y a des éléments à ajouter (tant que la liste n'est pas vide) ; il est désactivé après usage ;
- Si un élément est sélectionné dans la liste source, alors les boutons d'ajout sont activés ;
- Le bouton `bntAjoute` n'est valide que si un élément est sélectionné dans la liste source ;
- De même pour les boutons de suppression.

On peut traiter ce genre de problème de différentes manières, par exemple en se posant la question d'activer ou désactiver les boutons à chaque action ou tout centraliser au sein d'une méthode à laquelle on passe un contexte (Ajout, AjoutTout, Initialisation, ...)

## 1.8 Etape 6 : saisie d'un nouvel élément en liste source

Un contrôle `ComboBox` est une combinaison d'un `TextBox` et d'une `ListBox` ; il permet donc à l'utilisateur de *choisir* ou de *saisir*. On souhaite valider la saisie dès le click sur la flèche de déroulement du contrôle (événement `DropDown`). Dans Visual Studio, en mode Design, activer l'affichage des événements dans la fenêtre de \*Propriétés et double-cliquer sur l'événement `DropDown` du contrôle `cbxSource` pour générer le prototype de méthode événementielle correspondante :



**A noter :** il vous reste à gérer le cas où la zone de texte est vide quand l'utilisateur déroule la liste du `ComboBox` (essayez donc pour voir le problème !).

## 1.9 Etape 7 : gérer le repositionnement dans la ComboBox

Quand l'utilisateur sélectionne un élément de la ComboBox, celui-ci est automatiquement recopié dans la zone de saisie ; il reste à activer le bouton `btnAjouter`.

## 1.10 Etape 8 : gestion des boutons Haut et Bas.

Il s'agit ici de permuter un élément dans la ListBox avec son successeur ou son prédécesseur. On peut réaliser cela simplement en passant par une variable intermédiaire et en utilisant toujours les propriétés des objets de liste. Vous les chercherez mais la solution réside certainement dans la combinaison des propriétés `SelectedItem` et `SelectedIndex`. Vous devez bien sur prendre en compte les exceptions.

Tout comme pour gérer la ComboBox, il sera nécessaire de placer du code sur l'événement `SelectedIndexChanged` de la ListBox afin de gérer l'état des boutons `btnSupprime`, `btnHaut` et `btnBas`.

*Toute cette gestion de l'état des boutons devient lourde et parfois compliquée à gérer mais elle est nécessaire pour éviter les erreurs (**préventivement**) plutôt que de les détecter et de les gérer après coup (**curativement**) et informer correctement l'utilisateur sur les actions possibles dans le contexte courant.*

*Toute bonne application Windows doit affiner ainsi l'état des boutons et menus en fonction du contexte ; cela représente un effort de développement non-négligeable : à bien y regarder, on écrit ici plus de lignes de code pour gérer/éviter les erreurs que pour effectuer les traitements proprement dits !*

### **Amélioration :**

Plutôt que de gérer l'état des boutons et des autres contrôles dans les procédures événementielles de chaque gestionnaire d'événement, il peut avantageusement nous pouvons déporter cette gestion dans une autre procédure (par exemple `gererBoutons()`) appelée par chacune des procédures événementielles.

La logique de cette nouvelle procédure pourrait être d'activer (ou désactiver) les boutons suite à l'analyse des différents cas de figure (est-ce qu'on pointe le premier ? est-ce que la liste est vide ?...) Cette construction, tout aussi opérationnelle, peut paraître plus « brutale », moins « optimisée », mais elle présente l'avantage de faciliter la maintenance ultérieure du programme.

A vous !