r

Cyril Seguenot

2016

Résumé

Description des exercices donnés aux stagiaires d’une formation sur WPF faite en présentiel. La formation s’adresse à des personnes ayant déjà suivi la formation à C#.

Formation à WPF

Exercices

Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l’auteur ou de ses ayants droit ou ayants cause est illicite selon le Code de la propriété intellectuelle (Art L 122-4) et constitue une contrefaçon réprimée par le Code pénal.

Seules sont autorisées (Art L122-5) les copies ou reproductions strictement réservées à l’usage privé du copiste et non destinées à une utilisation collective, ainsi que les analyses et courtes citations justifiées par le caractère critique, pédagogique ou d’information de l’œuvre à laquelle elles sont incorporées, sous réserve, toutefois, du respect des dispositions des articles L 122-10 à L 122-12 du même Code, relatives à la reproduction par reprographie.

© Cyril Seguenot - 2016

Exercices

[Propriétés communes 3](#_Toc482662019)

[Visuel de l’explorateur de fichiers 3](#_Toc482662020)

[Trombinoscope 7](#_Toc482662021)

[Fenêtre de paramétrage 10](#_Toc482662022)

[Relevés météo 12](#_Toc482662023)

[Saisie d’employés 16](#_Toc482662024)

[Collections de bandes dessinées 17](#_Toc482662025)

[Saisie de tâches 19](#_Toc482662026)

[Contrôle utilisateur Inscription 23](#_Toc482662027)

# Propriétés communes

**Etape 1** : Dans la fenêtre principale du projet Entrainement, créer un bouton avec les caractéristiques suivantes :

* Toujours aligné en haut et au centre de son conteneur
* Texte "WPF c'est super !" en taille 20
* Largeur adaptée au contenu, mais avec une petite marge entre le texte et les bords du bouton
* Une marge de 10 entre le bouton et le bord supérieur de son conteneur

**Etape 2** : mettre le code précédent en commentaire, et créer un autre bouton avec les caractéristiques suivantes :

* Texte "Annuler" en taille 16
* Dimensions fixes 100 x 30
* Boutons toujours positionné dans le coin inférieur droit de la fenêtre, mais avec une petite marge

**Etape 3** : ajouter un second bouton avec le texte « OK » et les mêmes dimensions que le bouton Annuler

Faire en sorte qu’il soit toujours positionné à gauche du bouton Annuler, avec un espace constant entre les deux quand on redimensionne la fenêtre.

**Etape 4**: changement du conteneur principal

* Remplacer le conteneur principal <Grid> par un <StackPanel>, et affecter sa propriété Orientation à Horizontal.
* Supprimer les marges des boutons. Que constatez-vous ?
* Enlever les propriétés d’alignement sur les boutons, et les mettre sur le StackPanel lui-même. Que constatez-vous ?
* Faire les ajustements nécessaires pour obtenir le même visuel qu’à l’étape 4

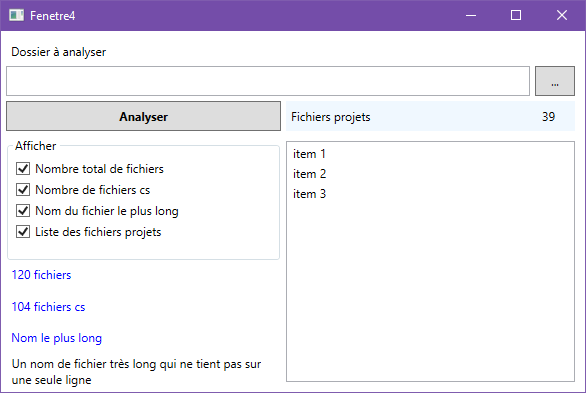
# Visuel de l’explorateur de fichiers

**Objectifs** :

* Apprendre à créer un projet WPF dans Visual Studio
* Découvrir le designer et la fenêtre de propriétés
* Mettre en œuvre une grille et des DockPanel pour faire un visuel dynamique
* Utiliser des styles

## Description du visuel

Il s’agit de faire l’interface visuelle présentée par la maquette ci-dessous :



On souhaite que l’interface soit dynamique, c’est-à-dire que lorsqu’on agrandit la fenêtre :

* Il n’apparaît pas d’espace vide sur les bords inférieur et droit de la fenêtre
* La zone de saisie du chemin du dossier s’agrandit en largeur, mais pas en hauteur
* Les contrôles de la partie gauche se répartissent en hauteur, et ne bougent pas en largeur
* La liste s’agrandit en hauteur et en largeur

**Etape 1 : Création du projet**

* Dans la solution ExercicesWPF, créer un nouveau projet de type « Application WPF » nommé ExplorateurFichiers.
* Dans MainWindow.xaml.cs, renommer la classe MainWindow en Fenetre1 et utiliser l’action rapide pour faire le renommage dans tout le projet.
* Renommer ensuite le fichier xaml lui-même en Fenetre1.xaml 🡪 le fichier .xaml.cs est également renommé automatiquement.
* Dans App.xaml , modifier la propriété StartupUri avec la valeur « Fenetre1.xaml »
* Lancer l’application pour vérifier que tout fonctionne

**Etape 2 : Création d’un visuel statique avec le Designer**

Nous allons créer l’interface ci-dessus de façon visuelle, à l’aide du designer. Pour cela :

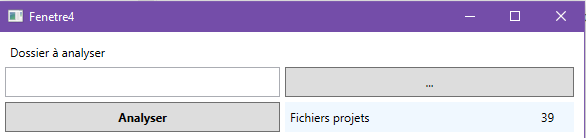
* Dans le designer, ajuster le zoom à 100%
* Auster la taille de la fenêtre à 400 x 600
* Faire glisser les contrôles depuis la boîte à outil dans le designer, modifier leurs libellés et contenus depuis la fenêtre de propriétés
* Aligner les contrôles en les sélectionnant et en choisissant l’option d’alignement souhaitée dans le menu contextuel, ou bien dans la barre d’outils Disposition (Layout)

Examiner le code xaml obtenu : tous les contrôles sont positionnés en définissant leurs marges par rapport aux bords de la grille.

Exécuter l’application, et jouer avec la taille de la fenêtre pour voir comment se comporte son contenu.

**Etape 3 : Création d’un visuel dynamique sans le designer**

Pour obtenir un rendu plus dynamique, nous allons positionner les éléments dans des lignes et colonnes de la grille. Pour simplifier, dans un premier temps, la zone de saisie du chemin de dossier et le bouton « … » s’afficheront de cette façon :



* Créer une nouvelle fenêtre nommée Fenetre2 et la définir comme fenêtre de démarrage dans l’objet Application
* Définir la même taille de fenêtre que précédemment
* Créer 8 lignes et 2 colonnes sans définir leurs tailles. Les libellés « Fichiers projets » et « 39 » sont dans la même colonne
* Ajouter les contrôles, en les positionnant dans la grille
* Exécuter l’application pour voir le rendu

**Etape 4 : Ajustement des lignes et colonnes**

* Affecter la valeur « Auto » aux hauteurs des lignes et largeurs de colonnes de la grille. Observer l’effet obtenu dans la fenêtre du designer
* Définir des hauteurs fixes de 30, 30 et 40 sur les 3 premières lignes, et des hauteurs \* sur les suivantes. Faire en sorte que la ligne contenant la groupbox soit toujours 4 fois plus haute que les suivantes
* Définir une largeur fixe de 275 pour la 1ère colonne
* Affecter la propriété ShowGridLines à true sur la grille
* Exécuter l’appli et redimensionner la fenêtre pour voir le résultat obtenu

**Etape 5 : Alignements verticaux et horizontaux**

* Faire en sorte que le compteur de fichiers projets soit aligné sur le bord droit de la grille, et aligné verticalement avec le libellé « Fichiers »
* Centrer verticalement les libellés colorés de la partie gauche à l’intérieur de leurs cellules respectives

**Etape 6 : Ajustement des marges**

Pour que les contrôles ne soient pas collés aux bords de la fenêtre, ni les uns contre les autres, mettre des marges sur la grille et la zone de saisie.

**Etape 7 : Couleurs**

Définir les couleurs des libellés de la partie gauche, et du panneau de compteur de fichiers projets

**Etape 8 : Aspect dynamique**

Faire en sorte que la zone de texte et la liste s’élargissent lorsqu’on élargit la fenêtre.

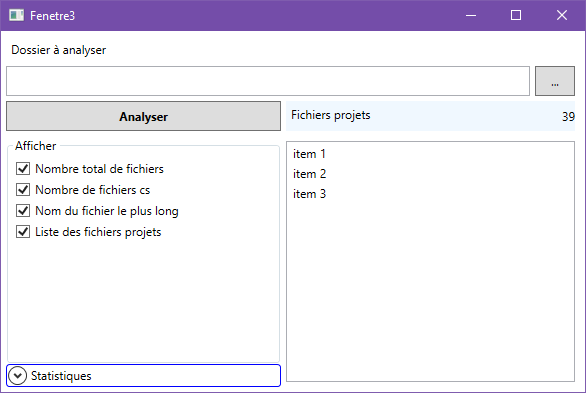
La zone de texte doit s’étendre sur la seconde colonne de la grille

**Etape 9 : Ajout de séparateurs de grille**

Ajouter un GridSplitter vertical entre les 2 colonnes, et un horizontal entre la 2ème et la 3ème ligne

**Etape 10 : Ajout d’un expander**

Remplacer les 4 dernières lignes de la grille par une seule ligne, et mettre les 4 libellés qui s’y trouvaient dans un expander. Faire en sorte que celui-ci se replie comme indiqué dans la capture ci-dessous :



**Etape 11 : Ajout d’aides visuelles**

* Ajouter une info-bulle sur le bouton pour expliquer son action à l’utilisateur
* Faire en sorte que le raccourci clavier ALT+D positionne le curseur sur la zone de saisie du dossier

**Etapes 12 : Style appliqué à un type cible**

* Dans les ressources de la fenêtre, ajouter un style ciblant les cases à cocher, permettant de les cocher par défaut et de leur affecter une marge uniforme de 3
* Supprimer les propriétés Margin et IsChecked affectées individuellement aux cases à cocher, et vérifier que le style s’applique

**Etapes 13 : styles individuels**

* Créer un style avec une clé nommée « BoutonAnalyser » qui définit une couleur de fond « Lavender », et une largeur minimale de 80, puis l’appliquer au bouton Analyser.
* Créer un style avec une clé nommée « Compteur » qui définit une couleur de police bleue, et l’appliquer aux libellés affichant les nombres de fichiers

**Etapes 14 : héritage de style**

* Créer un style avec une clé nommée « AffichageFichiersProjet », définissant une couleur de police rouge, et l’appliquer à la case à cocher « Liste des fichiers projets »
* Faire en sorte que ce style s’applique en plus du style défini à l’étape 12 pour le type CheckBox.

**Etapes 15 : Trigger**

* Ajouter quelques items en dur dans la ListBox
* Faire en sorte que lorsqu’on survole un item avec la souris, sa couleur de police devienne rouge
* Faire en sorte que lorsque l’item à le focus, cette couleur reste, et qu’en plus le texte soit en gras

# Trombinoscope

**Objectifs** :

* Créer des UserControl
* Illustrer les propriétés des ItemsControl (Menu, ListBox), sans databinding
* Récupérer et afficher des images stockées en base de données
* Utiliser différents types de ressources
* Mettre en œuvre le databinding
* Implémenter des relations maître-détail simple et hiérarchique

On souhaite créer une application avec une fenêtre principale et 2 contrôles utilisateurs qui contiendront respectivement un trombinoscope des employés de la base Northwind, et une liste des noms des employés. Ces contrôles seront chargés dans la fenêtre principale par clic sur des menus.

**Etape 1 : Création du projet et de la classe DAL**

* Créer un nouveau projet nommé Trombinoscope
* Créer une connexion et une classe de DAL pour la base Northwind

**Etape 2 : Fenêtre principale**

* Définir ses dimensions à 500 x 700
* Ajouter une barre de menus ancrée sur le bord supérieur, et 2 entrées de menus nommées Trombinoscope et Employés
* Dans la partie restante de la fenêtre, ajouter un ContentControl nommé « contentCtrl ». C’est lui qui affichera les contrôles utilisateur.

**Etape 3 : Création du 1er contrôle utilisateur**

Créer un UserControl nommé UCTrombi.

Placer à l’intérieur une ListBox, qui servira à afficher les photos des employés, les unes à côté des autres, et sur plusieurs lignes

NB/ Pour que cela fonctionne, vous devez affecter la propriété suivante sur la ListBox :

ScrollViewer.HorizontalScrollBarVisibility="Disabled"

Dans la fenêtre principale, au clic sur l’entrée de menu Trombinoscope, instancier un contrôle utilisateur UCTrombi, et définir cette instance comme contenu du contrôle contentCtrl.

Vous devez faire en sorte que si l’on clique plusieurs fois sur le menu, une seule instance du contrôle utilisateur soit créée.

**Etape 4 : récupération des photos**

Dans la DAL, créer une méthode pour récupérer les photos des employés.

Chaque photo sera récupérée sous forme d’un tableau de Byte, qui sera converti dans un format affichable grâce à la méthode suivante :

private static ImageSource ConvertBytesToImageSource(Byte[] tab)

{

   using (MemoryStream ms = new MemoryStream())

   {

      // Les images stockées dans la base Northwind ont un en-tête de 78 octets

      // qu'il faut enlever pour pouvoir les charger correctement

      ms.Write(tab, 78, tab.Length - 78);

      ImageSource image = BitmapFrame.Create(ms, BitmapCreateOptions.None,  
  BitmapCacheOption.OnLoad);

      return image;

   }

}

**Etape 5 : Affichage des photos**

Dans le constructeur du contrôle utilisateur UCTrombi :

* Récupérer la liste des photos sous forme de liste d’objet ImageSource en appelant la méthode de DAL créée précédemment.
* Pour chaque photo :
  + Instancier un contrôle Image (espace de noms System.Windows.Controls)
  + Affecter sa propriété Source avec la photo
  + Affecter sa largeur à 200 DPI
  + Ajouter l’image dans la collection des items de la ListBox

Lancer l’appli pour vérifier que les photos s’affichent bien

Vérifier que le visuel est conforme à ce qu’on attend lorsqu’on redimensionne la fenêtre

**Etape 6 : Création du 2ème contrôle utilisateur**

* Créer un contrôle utilisateur nommé UCEmployes
* Ajouter dedans une ListBox
* A côté, ajouter des TextBlock qui serviront à afficher l’Id, le nom et le prénom de l’employé sélectionné dans la liste
* Dans la fenêtre principale, au clic sur l’entrée de menu Employés, instancier un contrôle utilisateur UCEmployes, et définir cette instance comme contenu du contrôle contentCtrl.
* Vous devez faire en sorte que si l’on clique plusieurs fois sur le menu, une seule instance du contrôle utilisateur soit créée.

**Etape 7 : récupération des employés**

Dans la DAL, créer une méthode pour récupérer les Id, noms et prénoms des employés.

**Etape 8 : Affichage des employés**

Dans le constructeur du contrôle utilisateur UCEmployes :

* Récupérer la liste des employés sous forme de List<Employe>
* Affecter cette liste à la propriété ItemsSource de la ListBox
* Faire en sorte que la ListBox affiche les noms et prénoms des employés
* Lorsqu’on sélectionne un employé dans la liste, afficher ses Id, nom et prénom dans les TextBlock créés plus haut

Vérifier que tout fonctionne

**Etape 9 : Incorporation d’une image**

Ajouter l’image ci-dessous dans un dossier « Images » du projet. On fera en sorte qu’elle soit incorporée à l’exe à la compilation



Afficher cette image dans le contrôle utilisateur UCEmployes, au-dessus de la liste des employés

**Etape 10 : création de ressources resx**

Faire en sorte que les libellés des 2 entrées de menus soient stockées dans le fichier resx de l’application.

**Les étapes qui suivent concernent le databinding et s’appliquent sur le contrôle utilisateur UCEmployes**

**Etape 11 : Databinding sur des objets simples**

* Faire en sorte que l’affichage des caractéristiques de l’employé sélectionné (id, nom, prénom) se fasse désormais par databinding
* Ajouter une couleur de brosse « CouleurEmploye » dans le dictionnaire de ressources, puis appliquer cette brosse à la propriété Background du conteneur qui contient les libellés Id, Nom, Prénom, en utilisant un databinding.

**Etape 12 : Databinding sur une liste d’objets**

Faire en sorte que la liste des employés soit désormais remplie par databinding.

**Etape 13 : Maître-détail simple**

Supprimer le gestionnaire d’évènement associé à la sélection d’un employé dans la ListBox

Dans le xaml, faire en sorte que les caractéristiques de l’employé (id, nom, prénom) soient toujours rafraîchies lorsqu’on sélectionne un nouvel employé.

**Etape 14 : Mise en place d’une ListView**

Remplacer la ListBox par une ListView à 3 colonnes (Id, nom, prénom)

**Etape 15 : Maître-détail hiérarchique**

Dans la DAL, modifier la méthode qui récupère la liste des employés afin qu’elle récupère également la liste des territoires de chaque employé.

Dans le contrôle utilisateur, ajouter une seconde ListView à droite de la première, et faire en sorte qu’il affiche les territoires de l’employé sélectionné.

**Etape 16 : Compteur de territoires**

Juste au-dessus de la ListView des territoires, aligné sur la droite, afficher un compteur du nombre de territoires couverts par l’employé.

**Les étapes qui suivent concernent le DataTemplate**

**Etape 17 : Affichage des photos par binding**

Dans le constructeur du contrôle utilisateur UCTrombi, supprimer le code de remplissage de la ListBox.

Remplacer ce code par l’utilisation d’un DataTemplate dans le xaml

**Etape 18 : Affichage des noms des employés sous leurs photos**

Sous chaque photo, afficher le nom et le prénom de l’employé. Le texte doit être centré et de couleur bleue.

**Etape 19 : DataTemplate sur ListView**

Dans la contrôle utilisateur UCEmploye, dans la ListView des employés, ajouter une colonne « Manager » qui affiche le nom complet du manager (nom + prénom) à l’aide d’un DataTemplate.

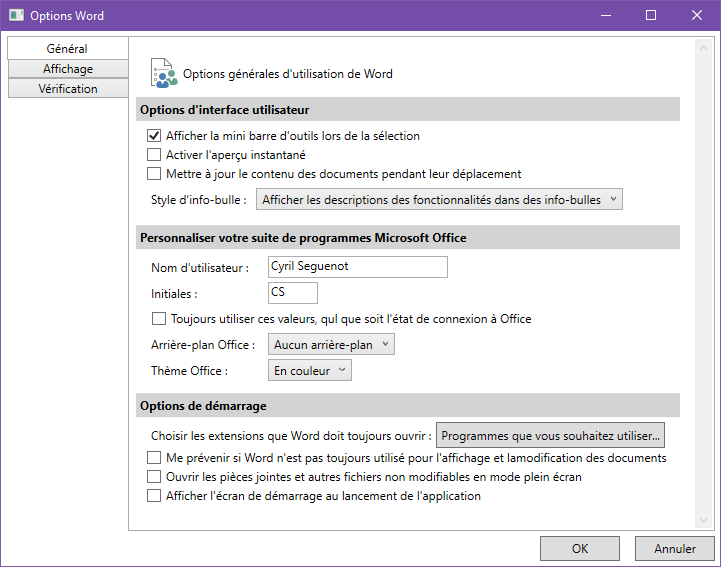
# Fenêtre de paramétrage

**Objectifs** :

* Se familiariser avec les contrôles de positionnement en réalisant une interface de complexité moyenne
* Mettre en œuvre les contrôles TabControl, ScrollViewer
* Définir des styles simples

## Description du visuel

Il s’agit de reproduire une partie de la fenêtre d’options de Word. Vous devrez faire en sorte d’obtenir un visuel le plus proche possible de celui-ci :



Vous devez respecter les contraintes suivantes :

* Le contenu de l’onglet doit défiler verticalement lorsqu’on réduit la hauteur de la fenêtre.
* L’image doit être ajoutée dans un dictionnaire de ressources spécifique DicoImages. Le fichier à utiliser est fourni ci-dessous
* Utiliser un style pour les bandeaux de sous-titres



## Indications

* Pour positionner les onglets sur la gauche, utiliser la propriété TabStripPlacement du TabControl
* Il est possible de spécifier des marges négatives pour obtenir un retrait

# Relevés météo

**Objectifs** : mettre en œuvre les notions suivantes :

* Liaison de données avec des objet simples et des collections
* DataTemplate
* Maître-détail
* Navigation, tri, regroupement
* Convertisseurs
* Evènement attachés

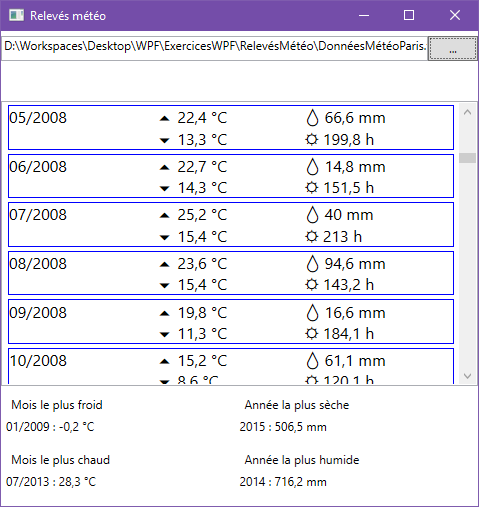
## Description de l’application

Le premier fichier ci-joint contient les relevés météo mensuels de la ville de Paris sur les 10 dernières années. Le second contient le code permettant de charger ces données dans une liste en mémoire.

|  |  |  |
| --- | --- | --- |
|  |  | Caractères Unicode pour les pictos :  🞁 = &#x23F6; 🌢 = &#x263C;  🞃 = &#x23F7; ☼= &#x1F4A7; |

Dans un premier temps L’application permettra de :

* Sélectionner le fichier de données météo à l’aide d’une boîte de dialogue
* Calculer des statistiques sur ces données
* Afficher les données et les résultats dans une ListBox avec le visuel ci-dessous



**Les premières étapes concernent les Datatemplate, et le maître-détail**

**Etape 1 : Création du projet et de la couche d’accès aux données**

* Créer un nouveau projet nommé RelevésMétéo
* Copier les 2 fichiers fournis en pièce jointe dans le répertoire du projet, et les ajouter au projet
* Dans le constructeur de la fenêtre, initialiser une variable privée de type DALMétéo

**Etape 2 : Ebauche du visuel**

Créer le visuel décrit par la capture ci-dessus, sans s’occuper de celui des items de la liste pour l’instant.

**Etape 3 : sélection du fichier et chargement des données**

Sur l’évènement Click du bouton :

* Ouvrir une boîte de dialogue « Ouvrir un fichier » en utilisant un objet Microsoft.Win32.OpenFileDialog
* Charger les données du fichier correspondant
* Si le chargement s’est bien déroulé, définir la variable privée de type DALMeteo comme contexte de la fenêtre, sinon afficher un message d’erreur

NB/ On peut initialiser le répertoire de départ de cette boîte de dialogue avec le répertoire de l’application en faisant : dlg.InitialDirectory = System.AppDomain.CurrentDomain.BaseDirectory;

**Etape 4 : création du DataTemplate**

* Dans les ressources de la fenêtre, créer le DataTemplate permettant d’afficher les items de la liste sous la forme souhaitée
* Le brancher sur la ListBox et vérifier en exécution que les données s’affichent correctement.

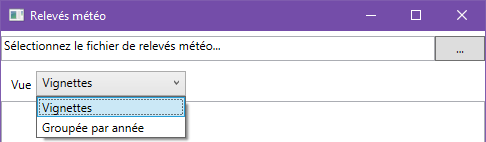
**Etape 5 : affichage des statistiques**

Sous la liste, afficher par databinding les statistiques fournies par la propriété Stats de la classe DALMétéo

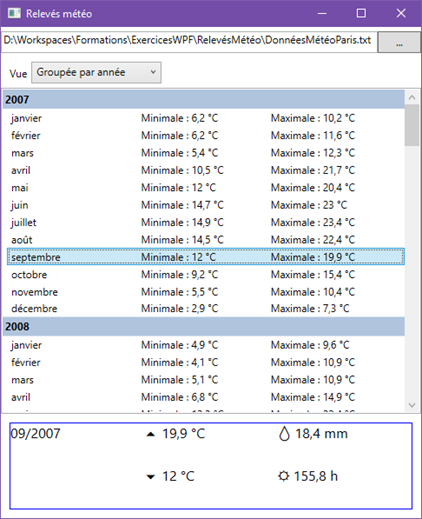
**Etape 6 : Création d’une seconde vue**

On souhaite ajouter les fonctionnalités suivantes à l’application :

* Sélection du type de vue (en vignettes ou regroupée par année)
* Affichage d’un détail différent selon la vue sélectionnée

****

Vue groupée par année + détail de la ligne courante :



On s’occupe dans un premier temps de la ListBox. Le bloc de détail sera géré à l’étape suivante

* Créer le second DataTemplate pour la ListBox correspondant à cette vue, sans créer de regroupement pour le moment (c’est-à-dire que les années sont affichées sur chaque ligne)
* Ajouter les 2 items « Vignettes » et « Groupée par année » dans la liste déroulante Vue
* Sur le changement de sélection, affecter à la ListBox le DataTemplate correspondant à la vue sélectionnée

**Les étapes qui suivent concernent les convertisseurs**

**Etape 7 : Affichage du détail**

Sous la ListBox, afficher le détail de la ligne sélectionnée au moyen d’un ContentControl, en réutilisant le template des items de la vue Vignette.

Lorsque la vue « Vignettes » est sélectionnée, on doit afficher les statistiques, et lorsque la vue « Groupée par année » est sélectionnée, on doit afficher le détail de la ligne courante. Pour cela, on utilisera un binding entre la propriété Visibility du ContentControl, et la propriété IsSelected du ComboBoxItem « Groupée par année ».

**Etape 8 : Mise en évidence des mois les plus secs**

Dans la vue avec vignettes, à l’aide de bindings et convertisseurs, mettre en évidence par une couleur de fond jaune clair les mois dont les précipitations sont inférieures à 10 mm.

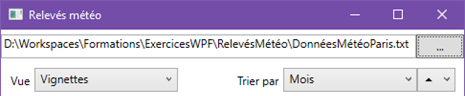
**Les étapes qui suivent concernent le regroupement et le tri**

**Etape 9 : Regroupement**

* Ajouter une propriété « Année » en lecture seule sur la classe DonnéesMois
* Créer le DataTemplate pour les en-têtes de groupes
* Dans le code C#, appliquer ou enlever le regroupement par année selon la vue sélectionnée

**Etape 10 : Ajout des listes déroulantes de tri**

* Ajouter les listes déroulantes suivantes pour les critères de tri :



* La liste déroulante « trier par » contient les choix suivants : Mois, T° min, T° max, Précipitations, Durée ensoleillement
* La liste déroulante située à droite contient deux choix symbolisés par une flèche montante et une flèche descendante, et permettant de choisir le sens de tri.

**Etape 11 : Implémentation des critères de tri**

* Sur le changement de sélection, trier les relevés météo selon le critère sélectionné, par ordre croissant pour l’instant.  
  Si la vue « groupée par année » est sélectionnée, le tri doit toujours être fait en premier sur l’année. Idem si l’on tri selon le mois.
* Tester. Que se passe-t-il si on tri selon la T° min, puis qu’on passe à la vue groupée ?
* Le fait de gérer le tri et le regroupement sur 2 gestionnaires d’évènements séparés est compliqué 🡪 créer une méthode privée pour gérer les 2 en même temps, et l’appeler dans les 2 gestionnaires d’évènement.

**Etape 12 : Implémentation de l’ordre de tri**

* Pendre en compte l’ordre de tri dans la fonction créée précédemment.

**Etape 13 : Amélioration de la gestion des évènements**

* Gérer un évènement routé commun pour les trois listes déroulantes

# Saisie d’employés

Repartir de la solution de l’exercice Trombinoscope. Le but de cet exercice est d’implémenter l’ajout et la suppression d’employés dans la liste des employés.

**Etape 1 : Définition du contexte**

Ajouter une classe ContexteEmploye qui servira désormais de DataContexte au contrôle utilisateur UCEmploye

Dans le constructeur de UCEmploye, définir une instance de ContexteEmploye comme DataContexte, et supprimer le code d’appel de la méthode de DAL.

**Etape 2 : création des propriétés du contexte**

Dans ContexteEmploye, créer :

* Une propriété en lecture seule Employes, de type de ObervableCollection<Employe>
* Une propriété en lecture + écriture privée EmployeCourant, qui représentera l’employé en cours de création

Dans le constructeur, initialiser ces deux propriétés, et charger la collection en appelant la méthode de DAL adéquate.

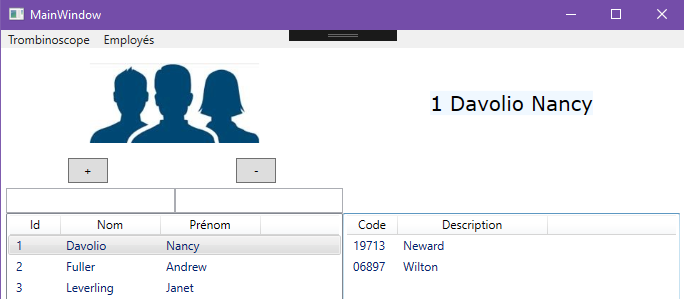
**Etape 3 : correction des bindings**

Dans le code xaml, corriger les bindings pour que les données s’affichent correctement. Exemple de syntaxe à utiliser pour le binding de la liste des territoires :

ItemsSource="{Binding Path=Employes/Territoires}"

**Etape 4 : création du visuel pour la saisie**

Ajouter des boutons + et -, ainsi que deux zones de texte pour la saisie des noms et prénom d’un nouvel employé.



Brancher par Databinding les zones de texte sur les propriétés Nom et Prénom de la propriété EmployeCourant du contexte. Ceci permettra de récupérer automatiquement les valeurs saisies.

**Etape 5 : création de la commande d’ajout**

* Créer une classe RelayCommand en reprenant le code fourni dans le cours.
* Dans ContexteEmploye, créer une commande relai CmdAjouter qui appelle une méthode privée AjouterEmploye.
* Dans le xaml, brancher le bouton + sur la commande

**Etape 6 : implémentation de la méthode d’ajout**

* Dans la méthode AjouterEmploye, si le nom et le prénom de l’employé courant sont bien renseignés, ajouter cet employé à la collection Employes.
* Puis affecter une nouvelle instance d’employé à la propriété EmployeCourant, afin de préparer la prochaine saisie.
* Vérifier en exécution que l’ajout fonctionne, et que l’employé apparaît dans la ListView

**Etape 7 : Vidage des zones de saisies après ajout**

On souhaite que les zone de saisie du nom et du prénom soient automatiquement vidées après clic sur le bouton +. Pour cela, il suffit de notifier les changements de la propriété EmployeCourant. En effet, comme l’étape précédente réinitialise l’employé courant, il suffit de notifier ce changement.

**Etape 8 : création de la commande de suppression**

* Dans ContexteEmploye, créer une commande relai CmdSupprimer qui appelle une méthode privée SupprimerEmploye.
* Dans le xaml, brancher le bouton - sur la commande
* Implémenter la méthode SupprimerEmploye. L’employé sélectionné dans la liste peut être obtenu au moyen du code suivant :  
  var e = (Employe)CollectionViewSource.GetDefaultView(Employes).CurrentItem;
* Vérifier en exécution que tout fonctionne

**Etape 9 pour les plus courageux : enregistrement en base**

* Créer les méthodes de DAL permettant d’ajouter et supprimer un employé ;
* Les appeler dans les méthodes AjouterEmploye et SupprimerEmploye (en gérant les erreurs éventuelles à la suppression)

# Collections de bandes dessinées

**Objectifs** : mettre en œuvre les notions suivantes :

* Databinding
* Navigation
* Ressources

## Description de l’application

Il s’agit de présenter des collections de bandes dessinées. Voici le visuel attendu :



La barre de boutons permet de naviguer parmi les collections.

Une collection est décrite par un nom (affiché en bleu), et contient :

* Une liste de genres
* Une liste d’auteurs
* Une liste d’albums, décrits par un N°, une année de parution et un titre

A chaque collection, on associe une image de même nom. Les images sont intégrées dans l’exécutable.

Le fichier zip ci-dessous fournit :

* Les données au format xml
* Le code de désérialisation
* Les images associées aux collections



## Indications

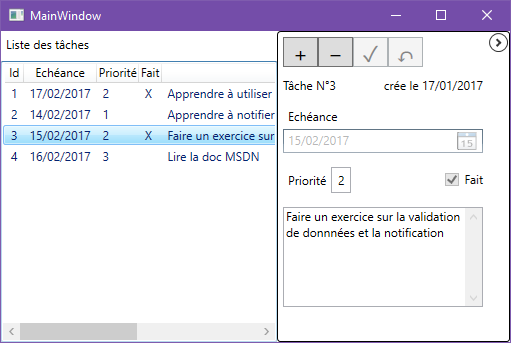
* Comme chaque image a le même nom que la collection de BD qu’elle représente, il est possible de construire son URI Pack dynamiquement à partir du nom de la collection : « pack://application:,,,/Images/nom collection.jpg »
* Caractères Unicode pour les flèches du navigateur : &#x25C4; &#x25BA;
* Pour créer une alternance de couleurs sur les lignes de la grille, regarder [cette page](https://msdn.microsoft.com/en-us/library/system.windows.controls.itemscontrol.alternationcount%28v=vs.110%29.aspx).

# Saisie de tâches

**Objectifs** : mettre en œuvre les notions suivantes :

* Commandes
* Notification des changements
* Validation de la saisie
* Modèle MVVM

Nous souhaitons réaliser une petite application de saisie de tâches, ressemblant à ceci :



La partie gauche affiche la liste des tâches, en lecture seule.

La partie droite (appelée fiche) affiche la tâche sélectionnée.

La barre de boutons permet de :

* Ajouter une nouvelle tâche
* Supprimer la tâche courante
* Enregistrer ou annuler la saisie

L’enregistrement se fait tâche par tâche, et il n’est pas possible de naviguer vers une autre tâche si une saisie est en cours. Pour simplifier, on n’autorise pas la modification d’une tâche déjà enregistrée.

Les données sont enregistrées dans un fichier **Taches.xml**, dont voici un exemple de contenu :

<?xml version="1.0" encoding="utf-8"?>

<Taches xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Tache Id="1" Creation="2017-01-15T00:00:00" Term="2017-02-17T00:00:00" Prio="2" Fait="true">Apprendre à utiliser les règles de validation en WPF</Tache>

<Tache Id="2" Creation="2017-01-16T00:00:00" Term="2017-02-14T00:00:00" Prio="1" Fait="false">Apprendre à notifier le visuel des changements de la source de données</Tache>

<Tache Id="3" Creation="2017-01-17T00:00:00" Term="2017-02-15T00:00:00" Prio="2" Fait="true">Faire un exercice sur la validation de donnnées et la notification</Tache>

<Tache Id="4" Creation="2017-01-18T00:00:00" Term="2017-02-16T00:00:00" Prio="3" Fait="false">Lire la doc MSDN</Tache>

</Taches>

**Etape 1 : Création du visuel**

Dans un nouveau projet nommé SaisieTaches, créer le visuel décrit par la maquette, sans s’occuper des bindings et commandes pour le moment.

La fiche est dans un Expander et son conteneur principal est un StackPanel

Les visuels des boutons de suppression, de validation et d’annulation peuvent être obtenus avec les caractères Unicode suivants : "&#x2212;" "&#x2713;" "&#x21b6;"

**Etape 2 : création du contexte**

* Créer une classe Tache avec le contenu suivant :

public class Tache

{

[XmlAttribute("Id")]

public int Id { get; set; }

[XmlAttribute("Creation")]

public DateTime Creation { get; set; }

[XmlAttribute("Term")]

public DateTime Term { get; set; }

[XmlAttribute("Prio")]

public int Prio { get; set; }

[XmlAttribute("Fait")]

public bool Fait { get; set; }

[XmlText]

public string Description { get; set; }

}

* Créer une classe Contexte contenant une liste de tâches nommée Taches, accessible en lecture depuis l’extérieur, et en écriture à l’intérieur de la classe.
* Définir un objet Contexte comme DataContext de la fenêtre

**Etape 3 : chargement des données xml**

* Créer le fichier Taches.xml dans le répertoire de la solution, avec le contenu donné plus haut.
* Dans VS, ajouter le fichier à la solution et spécifier qu’il doit être copié dans le répertoire de sortie à la compilation, ce qui permettra d’y faire référence très facilement.
* Créer une classe statique AccesDonnees qui sera dédiée à l’accès au fichier xml
* Ajouter une méthode ChargerTaches, qui charge les données du fichier dans une liste de taches, et la retourne comme résultat

Dans le constructeur de la classe Context, appeler la méthode ChargerTaches

**Etape 4 : création des commandes**

* Créer une classe RelayCommand en reprenant le code fourni dans le cours.
* Dans le contexte, créer 4 commandes nommées CmdAjouter, CmdSupprimer, CmdEnregistrer et CmdAnnuler, qui appellent 4 méthodes AjouterTache, SupprimerTache…etc. (vides pour l’instant).
* Dans le xaml, brancher les boutons sur les commandes

**Etape 5 : bindings de la liste**

Ajouter les bindings nécessaires sur la ListView pour afficher correctement les tâches. Lancer l’application pour vérifier que les données s’affichent bien.

**Etape 6 : liaison de la fiche sur la tâche courante**

Ajouter les bindings sur les contrôles de la fiche, en utilisant la syntaxe suivante pour accéder aux propriétés de la tâche courante :

Text="{Binding Path=Taches/Prio}"

Faire en sorte que la tâche courante soit bien celle qui est sélectionnée dans la ListView

Vérifier que la navigation dans la liste de tâches fonctionne.

**Etape 7 : mode d’édition**

La fenêtre peut être dans l’un des 2 modes suivants :

* En mode consultation, seuls les boutons « + » et « – » sont activés. Les autres boutons, les zones de saisie, et la liste sont désactivés ou en lecture seule
* En mode édition, c’est l’inverse

Le mode par défaut doit être la consultation. On passe en édition lorsqu’on ajoute une nouvelle tâche, et on revient en mode consultation lorsqu’on enregistre. Rappel : on n’autorise pas la modification d’une tâche déjà enregistrée.

Pour gérer le mode d’édition, nous allons ajouter une propriété ModeEdit sur le contexte afin de pouvoir faire des bindings dessus. Pour cela :

* Créer un énuméré ModesEdition avec les 2 valeurs Consultation et Edition
* Dans la classe Contexte, ajouter une propriété ModeEdit du type énuméré, et faire en sorte que ses changements de valeur soient bien notifiés aux éléments visuels qui seront liés à elle
* Dans les méthodes associées aux commandes, affecter la bonne valeur à la propriété ModeEdit

**Etape 8 : Activation des commandes et contrôles selon le mode d’édition**

Gérer l’activation des commandes en fonction du mode d’édition, puis vérifier en exécution que l’état des boutons est correct dans chaque mode.

Pour gérer l’activation des zones de saisie et de la liste, nous allons lier une de leurs propriétés à la propriété ModeEdit du contexte. La TextBox a une propriété IsReadOnly, mais le DatePicker et la CheckBox n’en ont pas. Nous utiliserons donc leur propriété IsEnabled. Pour cela :

* Créer un nouveau fichier Convertisseurs.cs contenant 2 classes de convertisseurs :
  + ConvModeEditActivation : pour transformer le type ModesEdition en un booléen utilisable sur IsEnabled
  + ConvModeEditLectureSeule : pour transformer le type ModesEdition en un booléen utilisable sur IsReadOnly
* Créer ces convertisseurs en tant que ressources de l’application pour qu’ils soient utilisables partout dans l’application
* Dans le code xaml, ajouter les bindings des propriétés IsEnabled et IsReadonly vers la propriété ModeEdit, en appliquant les convertisseurs

Vérifier que l’état d’activation des éléments visuels est correct dans chaque mode.

**Etape 9 : Ajout d’une tâche**

* Dans la classe Contexte, implémenter la méthode AjouterTache. Celle-ci doit ajouter une nouvelle tâche à la collection Taches, avec les valeurs par défaut suivante :
  + L'id sera initialisé avec la plus grande valeur trouvée dans la liste +1.
  + Les dates de création et d’échéance seront initialisées à la date du jour
  + La priorité sera initialisée à 1
* Faire en sorte que la nouvelle tâche apparaisse bien dans la ListView dès qu’elle est ajoutée à la collection Taches
* Faire en sorte que la nouvelle tâche devienne la tâche courante, c’est à dire qu’elle soit automatiquement sélectionnée dès qu’on l‘a créée.

**Etape 10 : Enregistrement**

Dans la classe AccesDonnees, ajouter une méthode EnregistrerTaches qui enregistre une liste de tâches dans le fichier xml. Pour simplifier, on ne gère pas de différentiel ; la liste passée en paramètre écrase les tâches du fichier.

Appeler cette méthode dans la méthode EnregistrerTache du contexte.

**Etape 11 : Suppression de la tâche courante**

Dans la classe Contexte, implémenter la méthode SupprimerTache, qui :

* Supprime la tâche courante de la collection
* Appelle la méthode AccesDonnees.EnregistrerTaches

NB/ Notez que le visuel est automatiquement mis à jour.

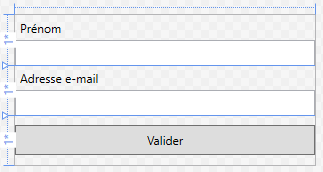
**Etape 12 : Annulation de la saisie**

Dans la classe Contexte, implémenter la méthode Annuler, qui consiste à supprimer la nouvelle tâche ajoutée, tant qu’elle n’a pas encore été enregistrée.

# Contrôle utilisateur Inscription

**Objectifs** : savoir créer un contrôle utilisateur ayant des propriétés de dépendance. Savoir le mettre en œuvre dans une fenêtre de test.

**Etape 1** : Dans un projet nommé Inscription, ajouter un nouveau contrôle utilisateur nommé UCInscription, et créer son visuel pour qu’il ressemble à ceci :



**Etape 2** : Dans le code-behind du contrôle, créer les 3 propriétés de dépendance suivantes au moyen du snippet propdp :

* Prenom : de type string avec une chaîne vide comme valeur par défaut
* Email : de type string
* Command : de type ICommand

**Etape 3** : Dans le xaml, relier les propriétés Text des TextBox et Command du bouton aux propriétés de dépendance, à l’aide de bindings sur le UserControl

**Etape 4** : Pour pouvoir tester le contrôle utilisateur, commencer par créer une classe Contexte contenant :

* Des propriétés Prenom et Email
* Une propriété CmdValider de type ICommand utilisant en interne une RelayCommand (récupérer le code de cette classe dans le cours)
* Une fonction Valider qui sera appelée par la commande, et qu’on laissera vide

**Etape 5** : Dans la fenêtre principale :

* Initialiser le DataContext de la fenêtre avec une instance de la classe Contexte, dont les valeurs des propriétés Prenom et Email seront définies en dur.
* Ajouter un contrôle utilisateur UCInscription
* Relier ses propriétés Prenom, Email et Command aux propriétés du contexte par binding

**Etape 6 :** mettre un point d’arrêt dans la méthode Valider et lancer l’application. Vérifier que :

* Les zones Prénom et e-mail sont bien remplies
* Lorsqu’on clique sur le bouton Valider, l’application s’arrête bien sur le point d’arrêt