IPEFA Sup Seraing - Verviers

Projet de développement SGDB

Notes de cours

WPF

Georgette Collard

2025-2026

Table des matières

Chapitre 13 : WPF (*Windows Presentation Foundation*)

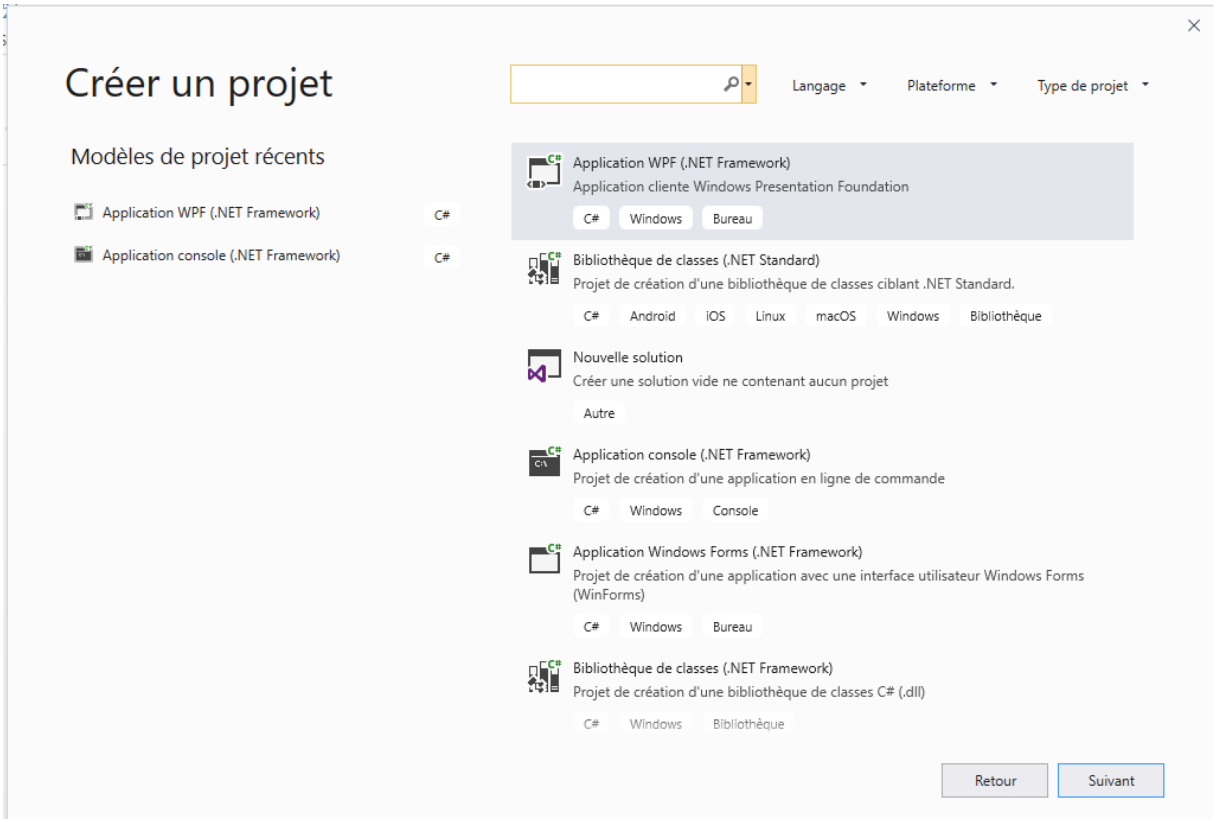
WPF est le successeur de la technologie *Windows Forms* et permet de créer des applications fenêtrées de type « client lourd ». Les applications WPF sont des applications de type « client lourd ». Elles s'exécutent directement depuis le système d'exploitation.

WPF est une bibliothèque permettant de réaliser des applications graphiques. Ces applications sont dites événementielles car elles réagissent à des événements (clic sur un bouton, redimensionnement de la fenêtre, saisie de texte, etc.)

WPF se base sur un paradigme **MVVM** (*Modèle-vue-vue-modèle*) ce qui permet une grande flexibilité grâce à la séparation entre données, traitements et présentation. Il permet à un designer de travailler spécifiquement sur une présentation sans se préoccuper des données à afficher qui est de la responsabilité du développeur. La liaison entre la présentation et les données se fait via un procédé nommé ***Binding*** ou ***DataBinding***.

Application WPF

Création d'une application WPF dans Visual Studio :



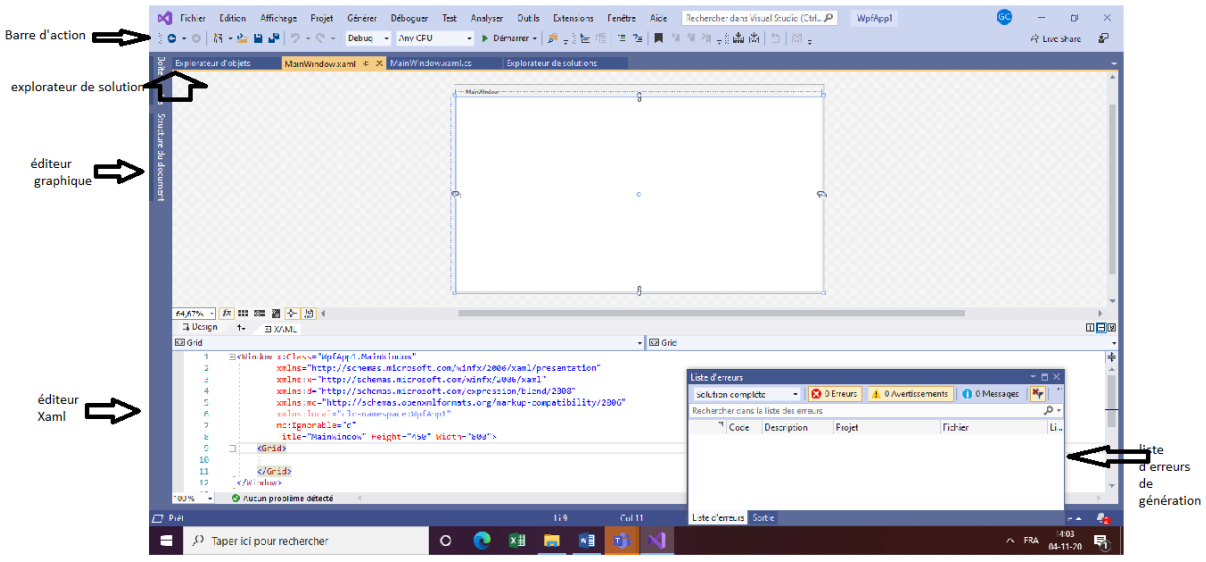
Composition d'un projet WPF

Une application WPF est construite grâce à deux langages. Un langage de présentation qui va permettre de décrire le contenu de notre fenêtre : le **XAML** (prononcez xamelle) et du C# qui va permettre de faire tout le code métier.

Une application WPF est décomposée en fenêtres qui sont affichées à partir de l'application principale. Chaque fenêtre est composée d'un fichier *.xaml* qui contient la description de la fenêtre (l'emplacement des boutons, des images, …) et d'un fichier de code *.xaml.cs* qui contient le code associé à cette fenêtre. On appelle ce fichier le « ***code behind*** ».

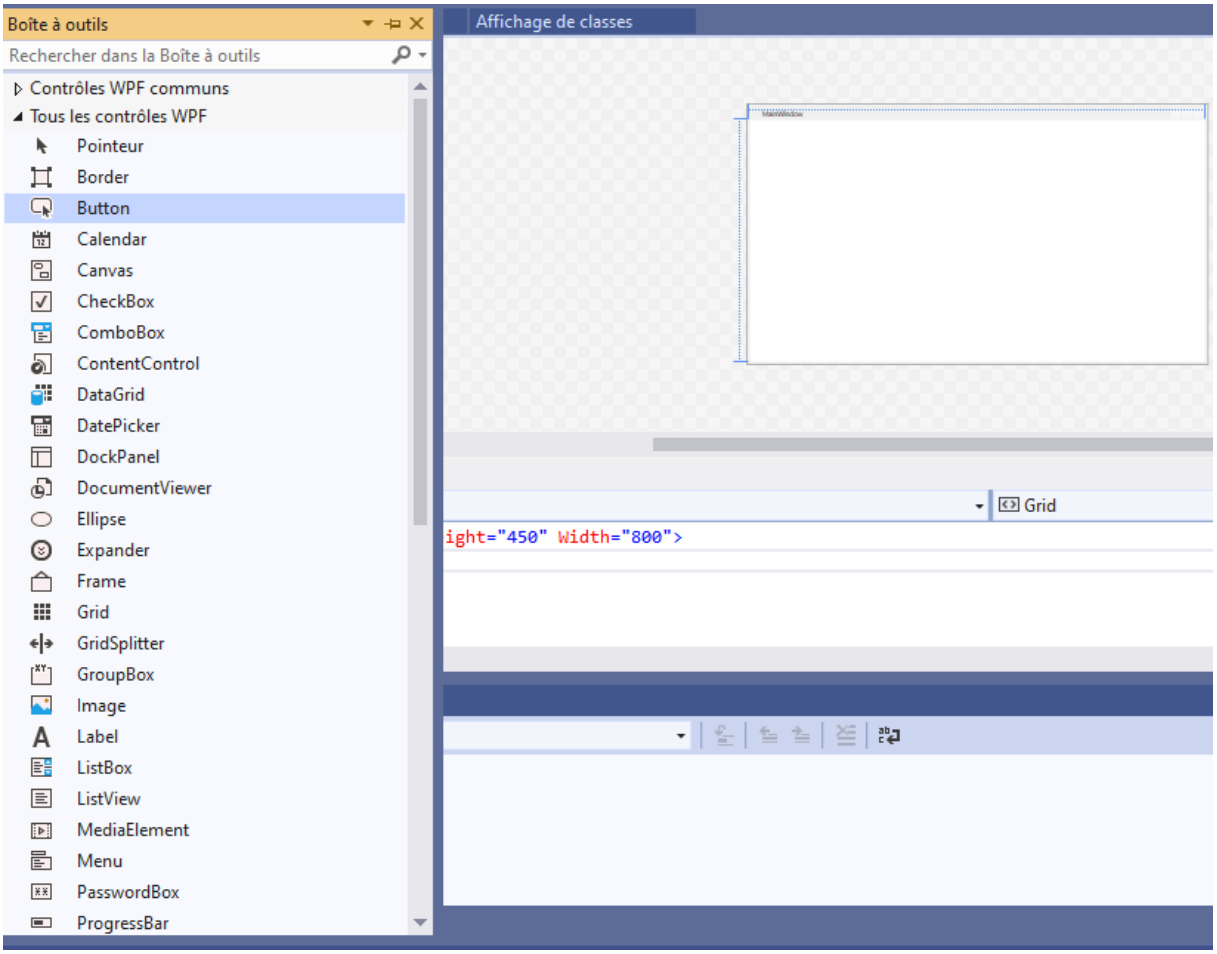
Lorsqu'on crée un projet, Visual C# Express créé automatiquement une fenêtre par défaut qui s'appelle *MainWindow*. Ce fichier est d'ailleurs ouvert automatiquement. Si nous déplions ce fichier dans l'explorateur de solutions, nous pouvons voir un autre fichier dessous qui possède l'extension *.xaml.cs*, le fichier de **code behind**.

Le XAML est un fichier XML qui va nous permettre de décrire le contenu de notre fenêtre. Cette fenêtre est composée de « contrôles », qui sont des éléments graphiques unitaires comme un bouton, une case à cocher, etc. Nous pouvons soit remplir ce fichier XAML à la main si nous connaissons la syntaxe, soit utiliser le designer pour y glisser les contrôles disponibles dans la boite à outils.

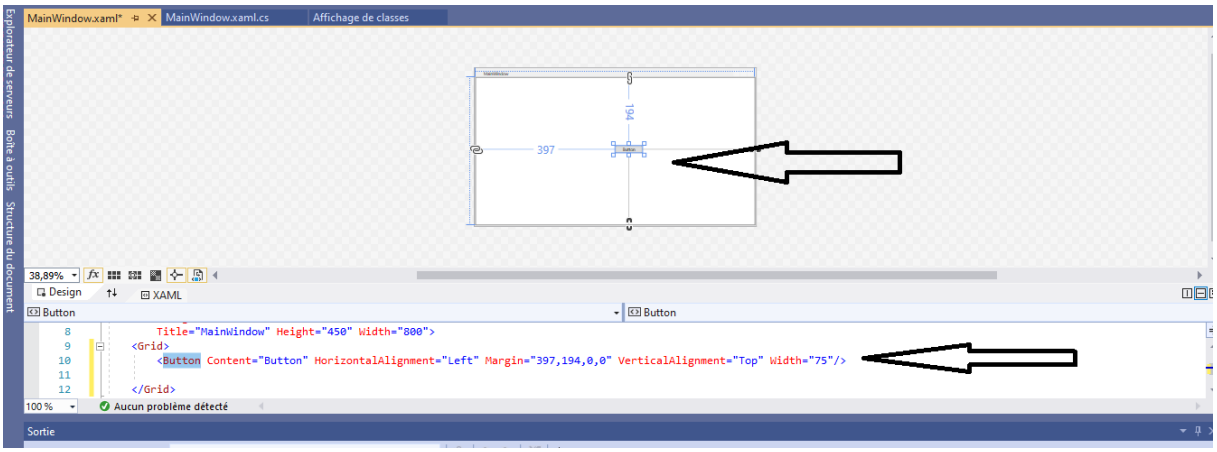


1) le rendu de notre fenêtre dans la partie haute du fichier *MainWindows.xaml*. C'est ça qui sera affiché lors du lancement de notre application.
2) le descriptif de cette fenêtre dans le code XAML, c'est-à-dire le langage permettant de décrire la fenêtre. Ce XAML est modifié si nous ajoutons des contrôles avec le designer. Inversement, si nous modifions le code XAML, le designer se met à jour avec les modifications.
3) Nous pouvons voir et modifier les propriétés du contrôle dans la fenêtre de propriétés.

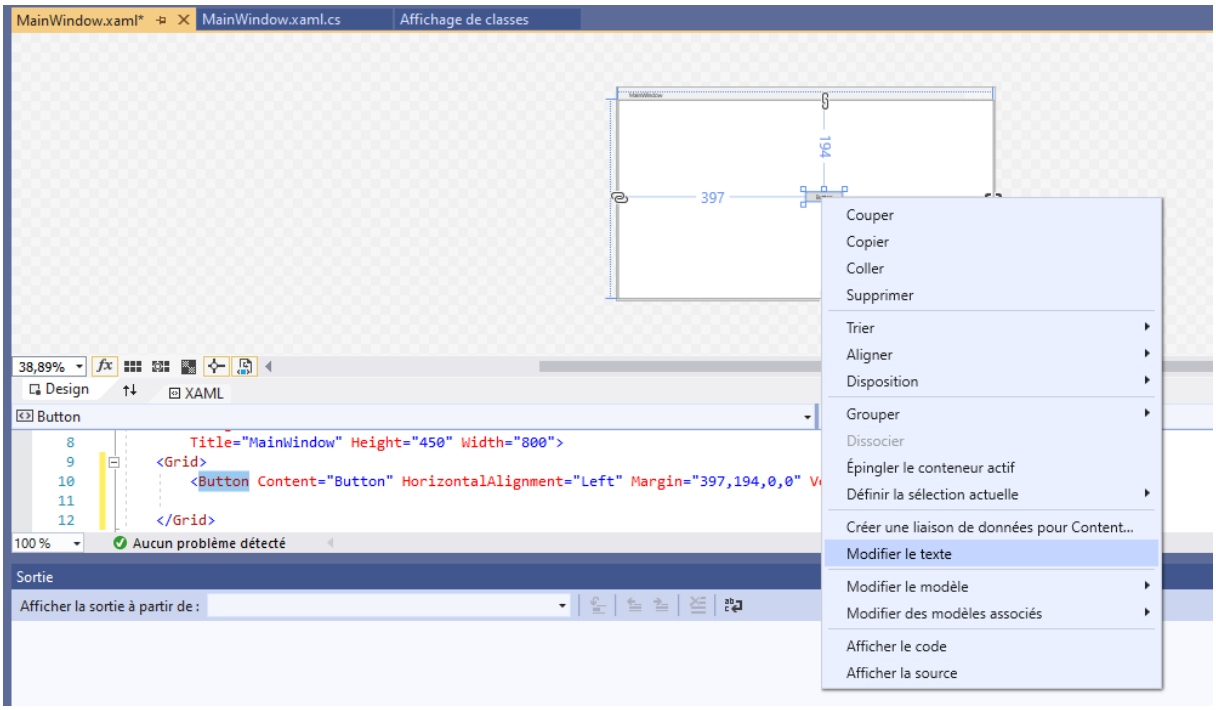
Exemple



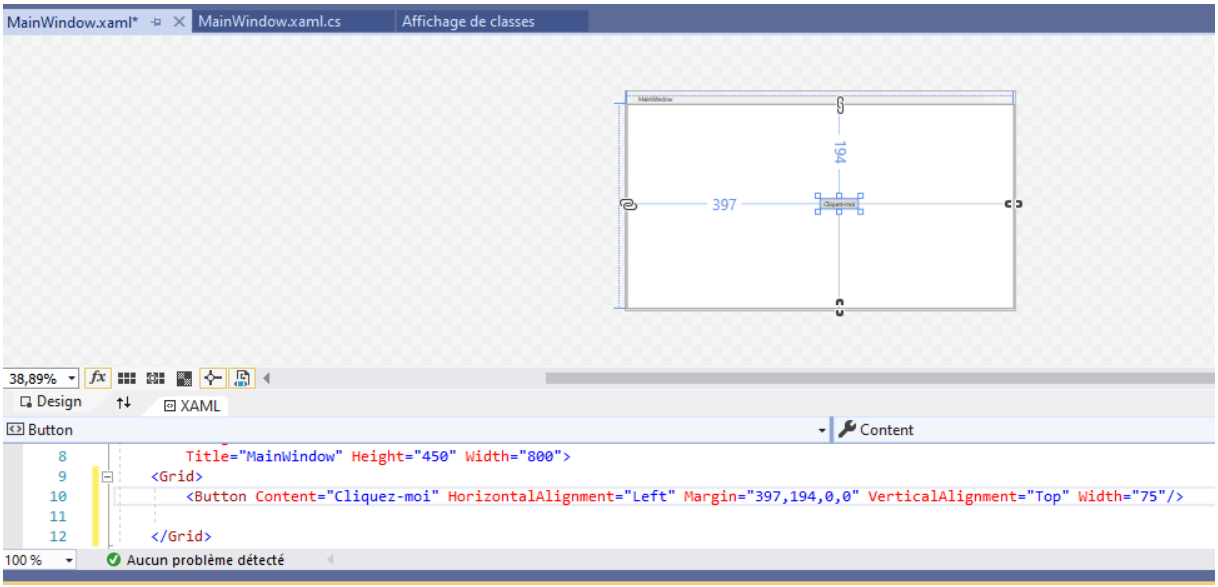
- Faire glisser le Button de la boîte à outils vers la fenêtre.



- Cliquer sur le bouton, cliquer droit et choisir Modifier le texte.



- Encoder Cliquez-moi



APP.xaml

App.xaml est le point de départ des déclarations de l'application. Il est créé automatiquement par Visual Studio, de même que le fichier code-behind ***App.xaml.cs***. Cette classe démarre les instructions et ensuite démarre la fenêtre ou la page désirée. C'est dans cette classe, qu'on souscrit à des événements d'application, tels que le démarrage de l'application, les exceptions gérées, … C'est dans cette classe, qu'on définit des ressources globales pouvant être accessibles depuis l'ensemble de l'application par exemple des styles globaux.

Quand nous créons une application, le fichier App.xaml est automatiquement généré et a cette structure comme ci-dessous :

```
<Application x:Class="WPF_App1.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WPF_App1"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

La propriété ***StartupUri*** définit la fenêtre ou la page à démarrer lorsque l'application est lancée. Dans ce cas-ci, *MainWindow.xaml* sera lancé.

MVVM

La technologie WPF est directement basée sur les préceptes d'une architecture nommée **Modèle-Vue-Vue** (MVVM – *Model View ViewModel*). Le principe est la séparation des données (le modèle) de l'interface utilisateur (la vue ou la présentation) et dont les interactions sont gérées par une couche intermédiaire, la vue-modèle.

MVVM implique la présence de 3 grandes entités :

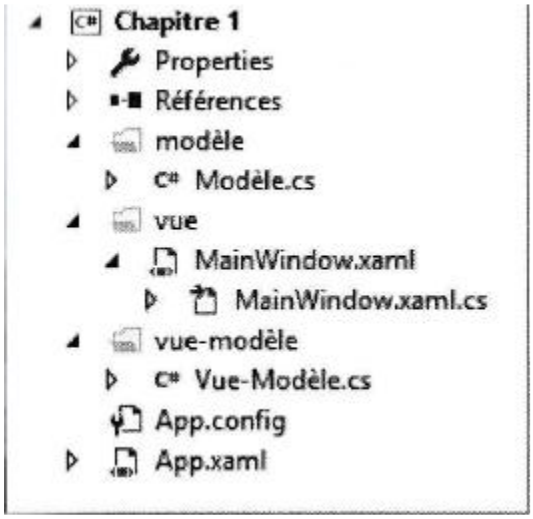
- L'interface graphique (la vue)
- L'accès aux données (modèle)
- Une couche interactive (le modèle-vue)

Exemple

Soit une interface utilisateur permettant de mettre à jour une valeur numérique.

Le projet doit comprendre 3 couches : vue, vue-modèle et modèle. Ces trois couches doivent respectivement correspondre à la présentation (vue), à la gestion des données (vue-modèle) et l'accès aux données (modèle). Idéalement, la valeur numérique doit être synchronisée entre la vue et la vue-modèle, voire avec le modèle.

Arborescence du projet :



- Accès aux données : c'est une simple classe *modèle.cs*
- Couche vue-modèle : c'est une classe *Vue-Modèle.cs* qui implémente l'interface **INotifyPropertyChanged**. Cette interface est le dispositif qui permet de notifier à la vue-modèle un changement de valeur. Cette interface est au cœur du mécanisme de **Binding**.
- Couche vue : *MainWindow.xaml*. Cette fenêtre reçoit comme contexte d'exécution la vue-modèle.

La présentation contient les contrôles suivants :

- Une *TextBox* contenant la valeur synchronisée avec la vue-modèle.
- Une *CheckBox* qui si elle est cochée, indique que non seulement la valeur numérique est synchronisée avec la vue-modèle, mais également avec le modèle.

Modèle.cs

```
namespace Chapitre_1
{
    class Modèle
    {
        public int Valeur { get; set; }

        public Modèle()
        {
            this.Valeur = 42;
        }
    }
}
```

Vue-Modèle.cs

```
using System.ComponentModel;

namespace Chapitre_1
{
    class Vue_Modèle : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        Modèle monModèle;

        public Vue_Modèle()
        {
            monModèle = new Modèle();
            MaValeur = monModèle.Valeur;
            ModificationModèle = false;
        }

        private int maValeur;
        public int MaValeur
        {
            get { return this.maValeur;  }
            set
            {
                this.maValeur = value;
                OnPropertyChanged("MaValeur");

                if (ModificationModèle == true)
                {
                    monModèle.Valeur = MaValeur;
                }
            }
        }

        private bool modificationModèle;
        public bool ModificationModèle
        {
            get { return this.modificationModèle; }
            set
            {
                this.modificationModèle = value;
                OnPropertyChanged("ModificationModèle");
            }
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="Chapitre_1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Chapitre_1"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Chapitre 1" Height="200" Width="250">
    <Grid>
        <StackPanel>
            <TextBox Text="{Binding MaValeur}" />
            <CheckBox IsChecked="{Binding ModificationModèle}" />
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;

namespace Chapitre_1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new Vue_Modèle();
        }
    }
}
```

Explication relative à MVVM

L'alimentation de la ***TextBox*** se fait uniquement par l'utilisation de

```
Text="{Binding MaValeur}
```

C'est-à-dire que l'alimentation de la *Textbox* depuis la vue-modèle et l'envoi de la valeur de la *TextBox* vers la vue-modèle se réalisent uniquement avec ce morceau de code. Cette approche permet de laisser à un graphiste la responsabilité du design XML de la vue tandis que le développeur se chargera du fonctionnel. L'utilisation de l'interface **INotifyPropertyChanged** permet de gérer cette séparation entre présentation et gestion des données.

- La vue-modèle « connaît » le modèle : en effet, c'est la vue-modèle qui l'instancie.
- La vue « connaît » la vue-modèle : en effet, c'est la vue qui l'instancie et qui l'envisage comme son contexte de données.

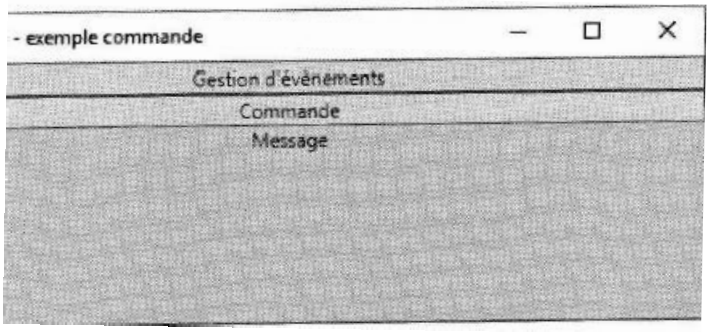
Chaque couche ne connaît que ce dont elle a besoin.

- Le modèle ne connaît pas son utilisation.
- La vue-modèle ne connaît pas son contexte d'utilisation, c'est-à-dire qu'elle est indépendante des vues qui l'utiliseront.

De même, WPF offre un mécanisme basé sur les commandes permettant de gérer les événements tout en respectant la séparation entre couches MVVM.

MVVM et les événements

Soit une fenêtre comprenant deux boutons : *Bouton_1* et *Bouton_2* ainsi qu'une zone de texte non éditable, une TextBox nommée *Resultat*. *Bouton_1* déclenche l'affichage d'un message dans la zone de texte et utilise une gestion d'événements classique. *Bouton_2* utilise une commande et affiche également un message dans la zone de texte.



MainWindow.xaml

```
<Window x:Class="CH1_SOURCE_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH1_SOURCE_2"
        mc:Ignorable="d"
        Title="Chapitre 1 - exemple commande" Height="200" Width="525"
        Background="Lightgray">
    <Grid>
        <StackPanel Orientation="Vertical">
            <Button Name="Bouton_1" Content="Gestion d'évènements" Click="Bouton_1_Click" />
            <Button Name="Bouton_2" Content="Commande" Command="{Binding Commande}" />
            <TextBlock Name="Resultat" Text="{Binding Message, Mode=TwoWay}" HorizontalAlignment="Center" />
        </StackPanel>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;

namespace CH1_SOURCE_2
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new Vue_Modele();
        }

        private void Bouton_1_Click(object sender, RoutedEventArgs e)
        {
            Resultat.Text = "Le bouton 1 a été cliqué (gestion classique des évènements)";
        }
    }
}
```

bouton_1 a un événement **Click** défini auquel est associé la méthode *Bouton_1_Click* dont le travail est de modifier le contenu de la zone de texte *Resultat.*

Le *Bouton_2* modifie le contenu de la zone de text *Resultat*, mais d'une manière différente, propre à WPF et respectant les préceptes de MVVM qui suggère la séparation stricte des fonctions, y compris quand il s'agit de gestion d'événements.

Vue_modèle.cs

```
namespace CH1_SOURCE_2
{
    class Vue_Modele : INotifyPropertyChanged
    {
        private ICommand commande;
        public ICommand Commande
        {
            get
            {
                return commande;
            }
            set
            {
                commande = value;
            }
        }

        string message;
        public string Message
        {
            get
            {
                return message;
            }
            set
            {
                message = value;
                OnPropertyChanged("Message");
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        public Vue_Modele()
        {
            commande = new RelayCommand(Execute_BoutonClick, CanExecute_BoutonClick);
            message = "Message";
        }
```

```
        public void Execute_BoutonClick(object parameter)
        {
            Message = "Le bouton 2 a été cliqué (commande)";
        }

        public bool CanExecute_BoutonClick(object parameter)
        {
            return true;
        }
    }
```

RelaisCommande.cs

```
using System;
using System.Windows.Input;

namespace CH1_SOURCE_2
{
    public class RelayCommand : ICommand
    {
        public delegate void ICommandOnExecute(object parameter);
        public delegate bool ICommandOnCanExecute(object parameter);

        private ICommandOnExecute _execute;
        private ICommandOnCanExecute _canExecute;

        public RelayCommand(ICommandOnExecute onExecuteMethod, ICommandOnCanExecute onCanExecuteMethod)
        {
            _execute = onExecuteMethod;
            _canExecute = onCanExecuteMethod;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute.Invoke(parameter);
        }

        public void Execute(object parameter)
        {
            _execute.Invoke(parameter);
        }
    }
}
```

La zone de texte *Resultat* est « bindée » avec l'attribut message de la vue-modèle qui a été définie comme le **DataContext** de la vue grâce à cette ligne :

```
this.DataContext = new Vue_Modele();
```

Le bouton *Bouton_2* voit l'action Click « bindée » avec une commande grâce à cet attribut dans le code XAML :

```
Command="{Binding Commande}"
```

XAML (*eXentensible Application Markup Language*)

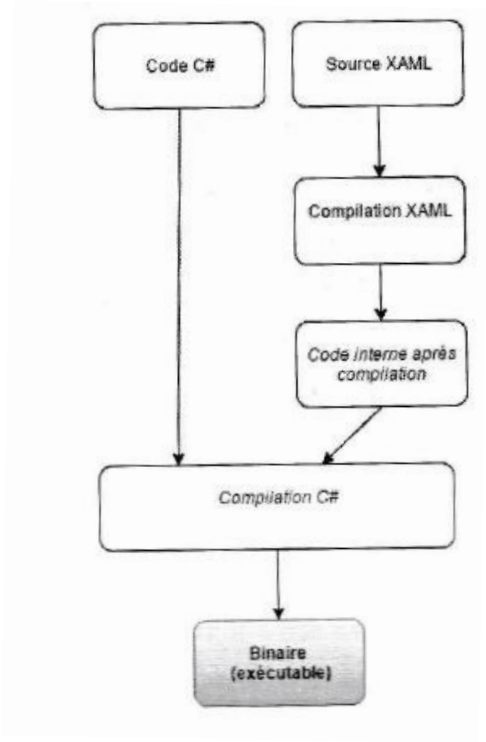
Dans un projet WPF minimal, il y a au moins deux fichiers d'extension .xaml : celui de l'application elle-même et celui de la description et de définition de la fenêtre principale.

Le code XAML est basé sur le langage de balisage XML. Chaque élément d'une interface, une vue, un bouton, un tableau ou tout autre contrôle graphique usuel, peut se définir en XAML.

Compilateur XAML

On retrouve du XAML pour le développement mobile phone, pour Silverlight, pour le développement d'application accessibles dans le Windows Store voire à du développement du type Xamarin.

Le compilateur XAML permet de transformer du code XAML en une sorte de code interne qu'un compilateur C# saura interpréter.



Syntaxe de base en XAML

Le langage XAML permet d'instancier des objets .Net. En effet, le XAML est un langage balisé dont la syntaxe est qualifiée d' « **élément-propriété** ». En effet, à une balise (un élément), on associe une collection de propriétés.

Exemple : L'élément **Button** possède les propriétés **Foreground, Background, Width, Height, Margin** et **Content**.

```
<Button Name="btn"
        Foreground="Red"
        Background="White"
        Width="75"
        Height="25"
        Margin="5"
        Content="Cliquez !"
/>
```

Le bouton a son fond coloré en blanc, son écriture en rouge, ses dimensions et une marge sont définies et contient le texte « Cliquez ! »

Le code suivant correspond au précédent. La propriété **Content** est écrite dans une balise dédiée et non comme attribut de la balise de l'élément.

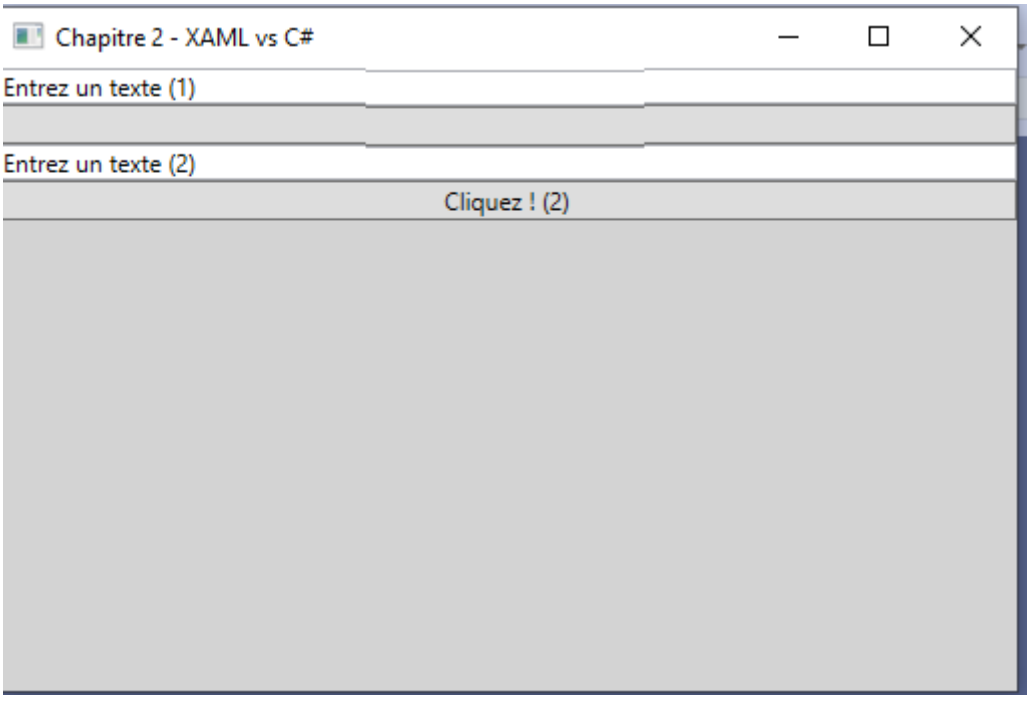
```
<Button Name="btn"
        Foreground="Red"
        Background="White"
        Width="75"
        Height="25"
        Margin="5">
    <Button.Content>"Cliquez !" </Button.Content>
</Button>
```

Chaque élément peut avoir une propriété par défaut en XAML. La propriété par défaut de l'élément **Button** est **Content**. Le code suivant est identique aux précédentes. La propriété **Content** n'est pas précisée, elle est implicite.

```
<Button Name="btn"
        Foreground="Red"
        Background="White"
        Width="75"
        Height="25"
        Margin="5">"Cliquez !"</Button>
```

Code XAML vs C#

L'exemple suivant ajoute un contrôle **StackPanel** comprenant un contrôle **TextBox** et un contrôle **Button**.



Définition de cet exemple en XAML

```
<Window x:Class="CH2_SOURCE_1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH2_SOURCE_1"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Chapitre 2 - XAML vs C#" Height="350" Width="525">
    <Grid Name="grid">
        <StackPanel Name="stack1">
            <TextBox Text="Entrez un texte (1)" />
            <Button Content="Cliquez ! (1)" />
        </StackPanel>
    </Grid>
</Window>
```

Définition de cet exemple en C#

```
using System.Windows;
using System.Windows.Controls;

namespace CH2_SOURCE_1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            StackPanel stack2 = new StackPanel();

            TextBox txtbox = new TextBox();
            txtbox.Text = "Entrez un texte (2)";
            stack2.Children.Add(txtbox);

            Button bouton = new Button();
            bouton.Content = "Cliquez ! (2)";
            stack2.Children.Add(bouton);

            stack1.Children.Add(stack2);
        }
    }
}
```

Classe de référence

Dans le code XAML, qui correspond à une fenêtre (*Window*), il est nécessaire de préciser la classe correspondance. Par défaut, lors de la génération d'un projet WPF, cette classe est automatiquement créée et renseignée. L'élément de syntaxe XML utilisé est **x :class** en attribut de la balise **window.**

```
<Window x:Class="MaClasse"
```

Espaces de noms

En développement C#, il est fréquent d'utiliser le mot-clé **using** pour référencer des espaces de noms comprenant des composants ou des éléments logiciels que le code utilise. En XAML, on fait de même.

```
xmlns:PREFIXE="clr-namespace:NOM_ESPACE_DE_NOMS"
```

- `PREFIXE` nomme localement l'espace de nom
- `NOM_ESPACE_DE_NOMS` référence l'espace de noms lui-même.

Par défaut, lors de la création d'un nouveau projet, ces espaces de noms sont créés :

```
<Window x:Class="MonEspaceDeNom.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr- MonEspaceDeNom"
```

XAML et contrôles utilisateur

Les contrôles possèdent un certain nombre de propriétés communes comme par exemple la largeur (*Width*), la hauteur (*Height*), le nom (*Name*), la couleur de fond (*Background*). Car, ils héritent de la classe *Control*.

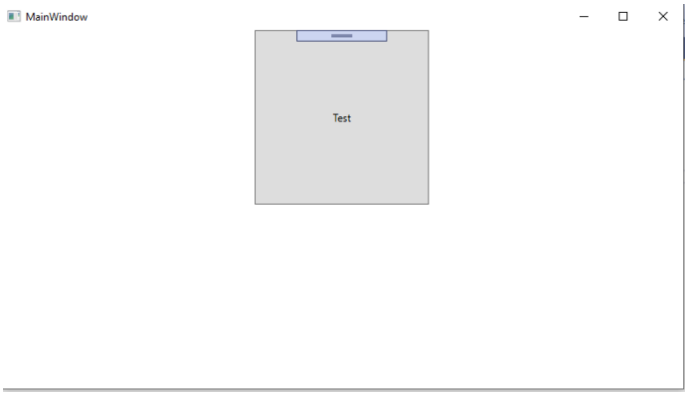
Concernant les dimensions, **Height** et **Width** peuvent prendre les valeurs suivantes :

- Une valeur numérique exprimée par défaut en pixels
- Le mot-clé **Auto** qui signifie que le contrôle courant va occuper toute la place possible dans les limites du contrôle qui le contient.

Par exemple, le code XAML :

```
<Grid>
    <StackPanel Width="200">
        <Button Content="Test"
                Width="Auto"
                Height="200"/>
    </StackPanel>
</Grid>
```

permet l'affichage de cette fenêtre :



- Le conteneur du bouton est un **StackPanel** de largeur 200 pixels (`Width="200"`) qui par défaut est centré.
- La hauteur du bouton est de 200 pixels car explicitement indiquée comme telle (`Height="200"`)
- La largeur du bouton prend la valeur **Auto**, c'est-à-dire que le bouton occupe en largeur tout l'espace qui lui et potentiellement octroyé par son conteneur, en l'occurrence, les 200 pixels du **StackPanel**.

Propriétés de positionnement des contrôles

Alignement

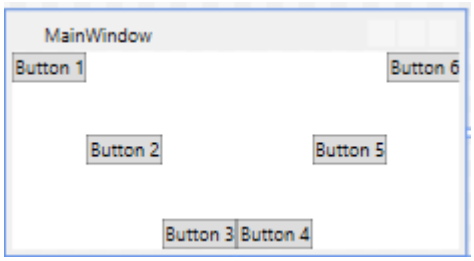
HorizontalAlignement peut prendre les valeurs **Left, Right, Center, Strech**.

VerticalAlignement peut prendre les valeurs **Top, Bottom, Center** ou **Strech**.

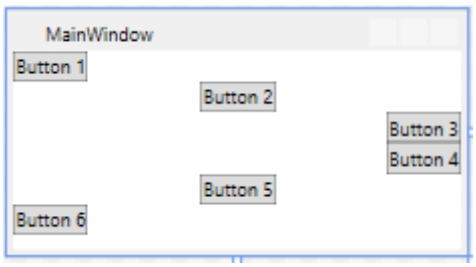
Les propriétés **Left**, **Right**, **Top**, **Bottom** permettent de « coller » le contrôle sur l'un des 4 bords tandis que **Center** permet un alignement centré verticalement et/ou horizontalement.

La propriété **Stretch** indique au contrôle d'occuper le maximum d'espace possible au sein de son conteneur dans la direction indiquée (verticale ou horizontal).

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="160" Width="300">
    <StackPanel Orientation="Horizontal">
        <Button VerticalAlignment="Top">Button 1</Button>
        <Button VerticalAlignment="Center">Button 2</Button>
        <Button VerticalAlignment="Bottom">Button 3</Button>
        <Button VerticalAlignment="Bottom">Button 4</Button>
        <Button VerticalAlignment="Center">Button 5</Button>
        <Button VerticalAlignment="Top">Button 6</Button>
    </StackPanel>
</Window>
```




```
<StackPanel Orientation="Vertical">
                <Button HorizontalAlignment="Left">Button 1</Button>
                <Button HorizontalAlignment="Center">Button 2</Button>
                <Button HorizontalAlignment="Right">Button 3</Button>
                <Button HorizontalAlignment="Right">Button 4</Button>
                <Button HorizontalAlignment="Center">Button 5</Button>
                <Button HorizontalAlignment="Left">Button 6</Button>
      </StackPanel>
```



Marges

Les marges permettent de spécifier une mesure sur chacune des 4 directions (gauche, haut, droite, bas) entre le bord du conteneur et le bord du contrôle courant.

On utilise la propriété **Margin**.

Ce code place une marge de 5 à gauche, 10 en haut, 15 à droite et 20 en bas. (L'ordre est important, rotation dans le sens des aiguilles d'une montre à partir de la gauche)

```
Margin="5,10,15,20"
```

Ce code est équivalent à `Margin="8,8,8,8"`

```
Margin="8"
```

Propriété Padding

Cette propriété permet de définir une zone à l'intérieur du contrôle dans laquelle le contenu ne pourra pas s'afficher.

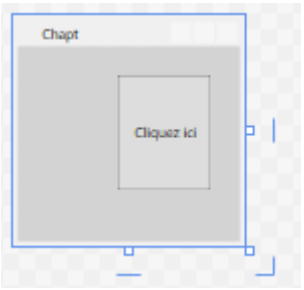
En quelque sorte, le **Padding** est l'équivalent de **Margin** pour l'intérieur du contrôle là où les marges gèrent l'espace à l'extérieur du contrôle.

Syntaxiquement, elle reprend le même principe que **Margin**.

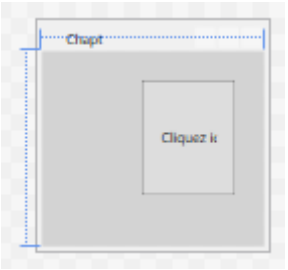
Exemple

```
<Window x:Class="testchapitre2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:testchapitre2"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Chapt" Height="200" Width="200">
    <Grid>
        <StackPanel Orientation="Vertical"

                    HorizontalAlignment="Stretch" VerticalAlignment="Stretch">

            <Button Content="Cliquez ici"
                    Width="80"
                    Height="100"
                    VerticalAlignment="Top"
                    HorizontalAlignment="Right"
                    Margin="25"
                    Padding="5"/>
        </StackPanel>
    </Grid>

</Window>
```

Ce code correspond à



Si la valeur du **Padding** est mise à 15, on a le résultat suivant : « *Cliquez-ici* » est tronqué.



Fenêtre et contrôles de disposition

Les contrôles de type conteneurs ou contrôles de disposition héritent de la classe Control qui possède une propriété **Content**. L'existence éventuelle de cette propriété **Content** permet d'envisager le contrôle en question comme un conteneur, c'est-à-dire un contrôle pouvant contenir d'autres contrôles.

Contrôle de grille

Le contrôle **Grid** consiste en une grille de lignes et de colonnes. Par défaut, si aucune propriété n'est indiquée, la grille ne comporte qu'une seule cellule. Mais son utilisation permet un quadrillage de l'espace.

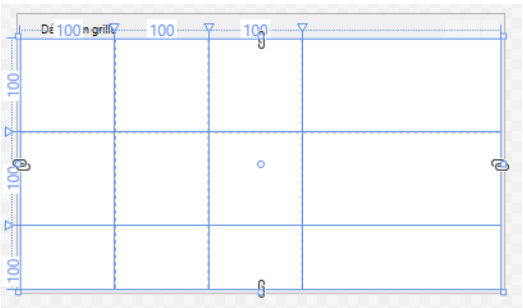
Pour ce faire, le contrôle **Grid** fournit deux propriétés de type collection :

- **RowDefinitions**
- **ColumnDefinitions**

Ces deux collections contiennent respectivement des objets de type **RowDefinitions** et **ColumnDefinitions.** L'idée est de construire une matrice définie par une collection de lignes et une collection de colonnes.

Exemple : ce code définit la grille :

```
<Window x:Class="CH3_SOURCE1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_SOURCE1"
        mc:Ignorable="d"
        Title="Définition grille" Height="300" Width="525">
    <Grid ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="100" />
            <RowDefinition Height="100" />
            <RowDefinition Height="100" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
    </Grid>
</Window>
```



À ce stade, seule la définition de la grille est réalisée. Il reste encore à l'utiliser et à insérer des contrôles et des informations dans les cellules de la grille.

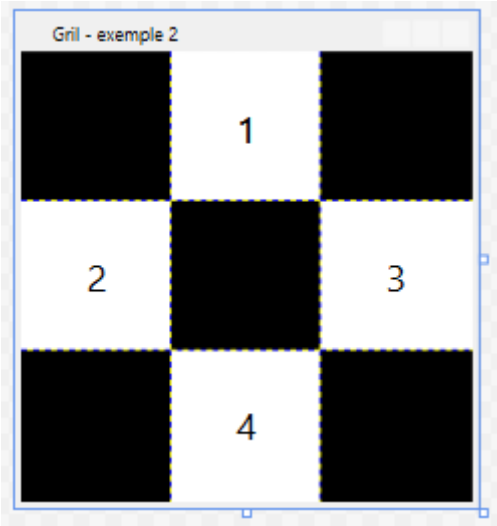
Pour cela, on utilise les propriétés **Grid.row** et **Grid.Column** qui permettent de référencer une cellule grâce à un système de coordonnées :

Grid.Row="0" et *Grid.Column="4"* correspond à la cellule située à la première ligne et à la cinquième colonne.

Exemple : Damier de trois cases sur trois cases colorées en noir et blanc alternativement. Les cases blanches sont numérotées.

```
<Window x:Class="DAMIERGRIL.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_SOURCE1"
        mc:Ignorable="d"
        Title="Gril - exemple 2" Height="333" Width="310">
    <Grid ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="100" />
            <RowDefinition Height="100" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Grid Grid.Row="0" Grid.Column="0" Background="Black" />
        <Grid Grid.Row="0" Grid.Column="2" Background="Black" />
        <Grid Grid.Row="1" Grid.Column="1" Background="Black" />
        <Grid Grid.Row="2" Grid.Column="0" Background="Black" />
        <Grid Grid.Row="2" Grid.Column="2" Background="Black" />

        <StackPanel Grid.Row="0"
                    Grid.Column="1"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <TextBlock Text="1" FontSize="25"/>
        </StackPanel>

        <StackPanel Grid.Row="0"
                    Grid.Column="1"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <TextBlock Text="1" FontSize="25"/>
        </StackPanel>

        <StackPanel Grid.Row="1"
                    Grid.Column="0"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <TextBlock Text="2" FontSize="25"/>
        </StackPanel>

        <StackPanel Grid.Row="1"
                    Grid.Column="2"
```

```
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
                <TextBlock Text="3" FontSize="25"/>
            </StackPanel>

            <StackPanel Grid.Row="2"
                    Grid.Column="1"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
                <TextBlock Text="4" FontSize="25"/>
            </StackPanel>
        </Grid>
</Window>
```



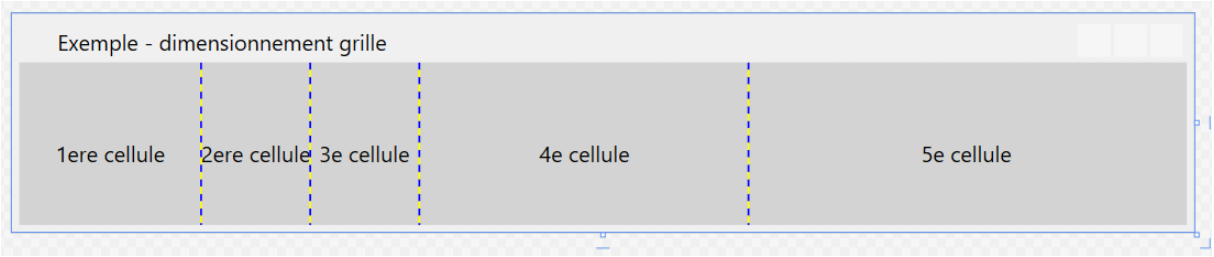
La grille peut rester visible à l'exécution. Pour cela, il suffit d'ajouter la propriété **ShowGridLines** à la valeur **True**.

Grille et dimension des cellules

Width ou **Height** peuvent prendre les valeurs suivantes :

- Une valeur numérique `Height="100"`. La hauteur sera de 100 pixels.
- `"Auto"` Cette valeur signifie que la taille de la colonne ou de la ligne concernée s'adapte au contenu de ladite colonne ou ligne. S'il n'y a pas de contenu, la largeur ou la longueur concernée sera nulle.
- * : Cette valeur permet de spécifier une proportion de l'espace restante. Ainsi, si l'espace restante (après prise en compte des valeurs numériques et des valeurs `"Auto"`) voit trois colonnes avec ces largeurs `"*", "3*", "4*"`, la première colonne occupera 1/8 de l'espace restante, la seconde 3/8 et la dernière, la moitié.

```xaml
<Window x:Class="CH3_SOURCE2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_SOURCE2"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Exemple - dimensionnement grille" Height="120" Width="650">
    <Grid ShowGridLines="True">
        <Grid.RowDefinitions>
            <RowDefinition Height="100" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100" />
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="3*" />
            <ColumnDefinition Width="4*" />
        </Grid.ColumnDefinitions>

        <Grid Grid.Row="0" Grid.Column="0" VerticalAlignment="Center" HorizontalAlignment="Center">
            <TextBlock Text="1ere cellule" />
        </Grid>

        <Grid Grid.Row="0" Grid.Column="1" VerticalAlignment="Center" HorizontalAlignment="Center">
            <TextBlock Text="2ere cellule" />
        </Grid>

        <Grid Grid.Row="0" Grid.Column="2" VerticalAlignment="Center" HorizontalAlignment="Center">
            <TextBlock Text="3e cellule" />
        </Grid>

        <Grid Grid.Row="0" Grid.Column="3" VerticalAlignment="Center" HorizontalAlignment="Center">
            <TextBlock Text="4e cellule" />
        </Grid>

        <Grid Grid.Row="0" Grid.Column="4" VerticalAlignment="Center" HorizontalAlignment="Center">
            <TextBlock Text="5e cellule" />
        </Grid>

    </Grid>
</Window>
```



Par défaut, chaque contrôle occupe une case ou cellule. Pour qu'un contrôle occupe plus d'une ligne ou d'une colonne, on utilise respectivement la propriété ***RowSpan*** ou ***ColumnSpan***.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button>Button 1</Button>
    <Button Grid.Column="1">Button 2</Button>
    <Button Grid.Row="1" Grid.ColumnSpan="2">Button 3</Button>
</Grid>
```

Contrôle de type Panel

Outre, la grille détaillée précédemment, les différents contrôles de type Panel sont les principaux contrôles qui participent à l'organisation d'une fenêtre WPF. En particulier, ils permettent de structurer l'espace de la fenêtre et de dédier un emplacement de la fenêtre à une fonctionnalité donnée. Ainsi, le développeur peut par exemple structurer sa fenêtre en consacrant une partie de l'espace à un formulaire de recherche et aux critères de recherche, et une autre partie de l'espace à l'affichage des résultats. En WPF, on distingue trois types de Panel : le **StackPanel**, le **DockPanel**, le **WrapPanel**.

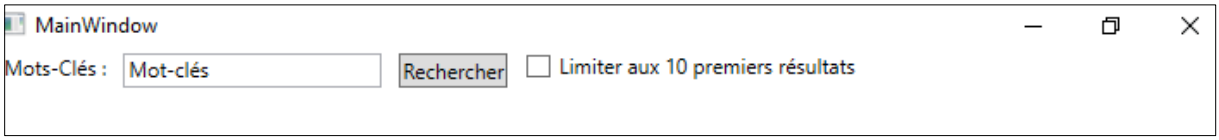
StackPanel

Ce contrôle permet de disposer de façon horizontale ou verticale plusieurs composants visuels. Sa principale propriété est donc **Orientation** qui prend comme valeur soit **Horizontal** ou **Vertical** (valeur par défaut).

Exemple : formulaire de recherche défini à l'aide d'un **StackPanel** orienté horizontalement.

```
<StackPanel Orientation="Horizontal"
            HorizontalAlignment="Left"
            VerticalAlignment="Top">
    <TextBlock Text="Mots-Clés :"
               Margin="5"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"/>
    <Button Content="Rechercher"
            Margin="5" />
    <CheckBox Content="Limiter aux 10 premiers résultats"
              Margin="5" />

</StackPanel>
```

Résultat :



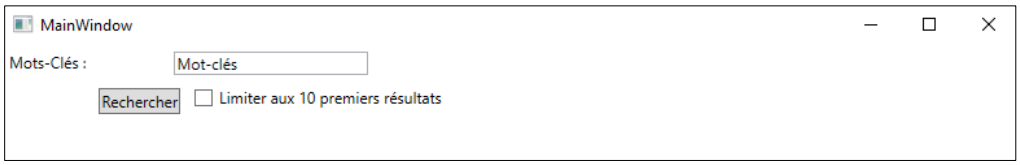
DockPanel

Le **DockPanel** se distingue par la gestion des contrôles selon un système d'ancres. L'idée est de fixer un composant inclus dans le **DockPanel** sur son bord haut, bas, gauche ou droit via la propriété **Dock**.

Exemple :

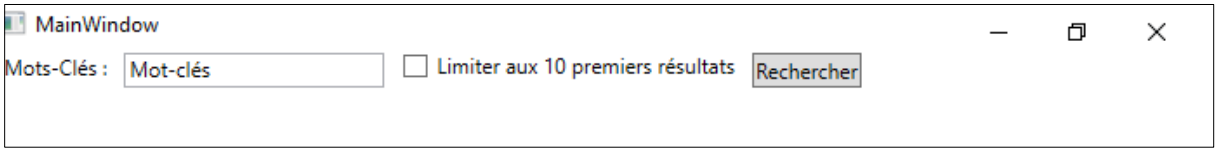
```
<DockPanel HorizontalAlignment="Left"
           VerticalAlignment="Top">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Left"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Top"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Right"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Bottom"
            Margin="5" />
</DockPanel>
```

Résultat :

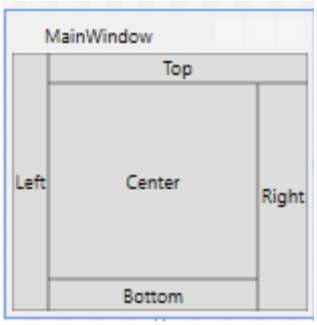



```
<DockPanel HorizontalAlignment="Left"
           VerticalAlignment="Top">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Left"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Left"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Left"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Left"
            Margin="5" />
</DockPanel>
```

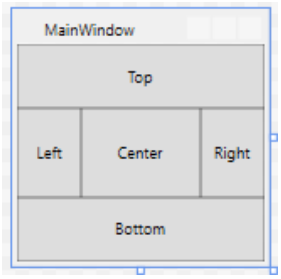
Résultat :



```xaml
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="200">
    <DockPanel>
        <Button DockPanel.Dock="Left">Left</Button>
        <Button DockPanel.Dock="Top">Top</Button>
        <Button DockPanel.Dock="Right">Right</Button>
        <Button DockPanel.Dock="Bottom">Bottom</Button>
        <Button>Center</Button>
    </DockPanel>
</Window>
```




```xaml
<DockPanel>
                <Button DockPanel.Dock="Top" Height="50">Top</Button>
                <Button DockPanel.Dock="Bottom" Height="50">Bottom</Button>
                <Button DockPanel.Dock="Left" Width="50">Left</Button>
                <Button DockPanel.Dock="Right" Width="50">Right</Button>
                <Button>Center</Button>
      </DockPanel>
```



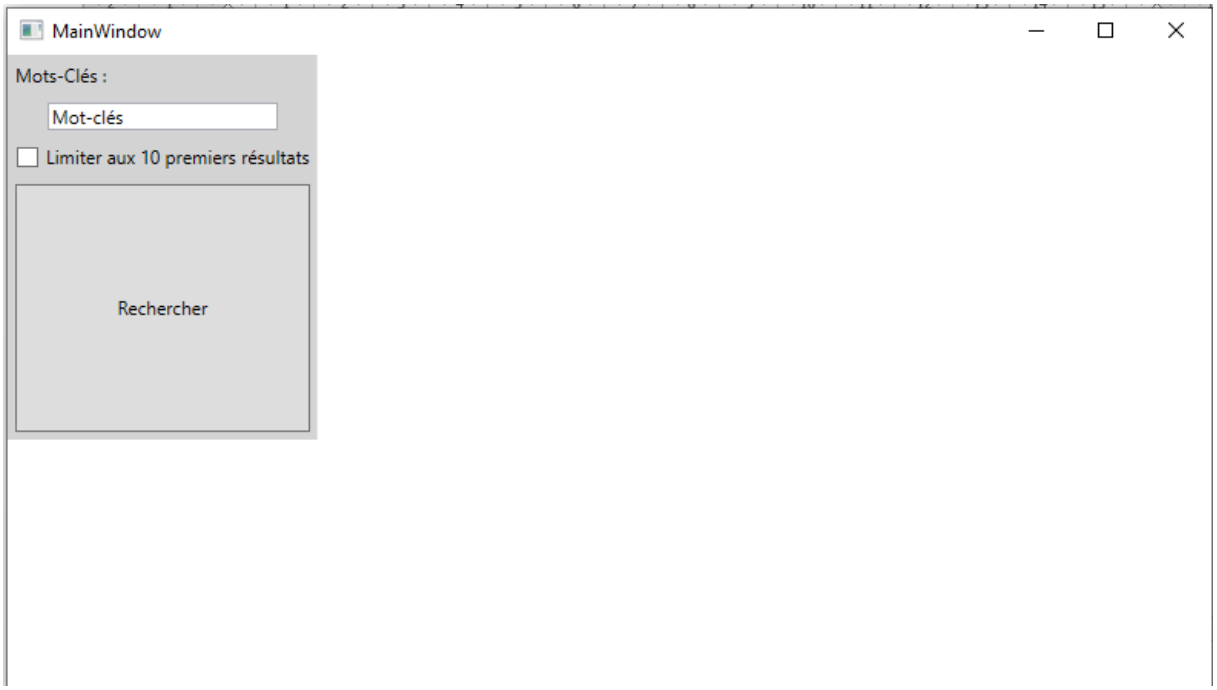
Propriété LastChildFill

La propriété **LastFill** permet d'indiquer si le dernier contrôle instancié va occuper (valeur **True**) ou non (valeur **False**), l'espace restant au sein du **DockPanel**.

Exemple : les composants sur le bord haut et on donne une hauteur déterminé au control **DockPanel**.

```
<DockPanel HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Background="LightGray"
           Height="250"
           LastChildFill="True">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Top"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Top"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Top"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Top"
            Margin="5" />
</DockPanel>
```

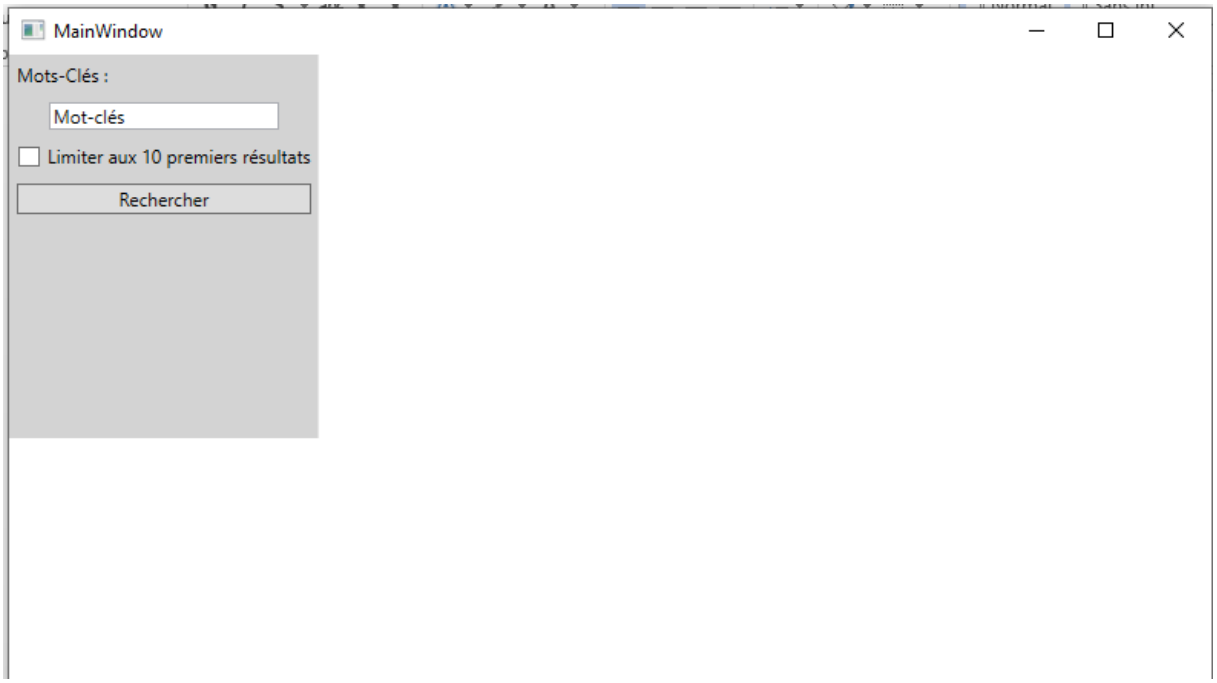
Résultat



Exemple : **LastChildFill** mis à **False**

```
<DockPanel HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Background="LightGray"
           Height="250"
           LastChildFill="False">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Top"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Top"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Top"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Top"
            Margin="5" />
</DockPanel>
```

Résultat



WrapPanel

Le **WrapPanel** permet dans le cas d'une orientation horizontale de gérer la largeur de contrôle avec une approche par ligne. Si l'espace n'est pas suffisant sur la ligne courante, le **WrapPanel** affecte le composant à instancier à la ligne suivante. De même, si l'orientation est verticale, le **WrapPanel** évalue l'espace disponible dans la colonne courante.

Exemple : orientation horizontal

```
<WrapPanel HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Background="LightGray"
           Height="100"
           Width="300"
           Orientation="Horizontal">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Top"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Top"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Top"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Top"
            Margin="5" />
</WrapPanel>
```

Résultat :

Le **CheckBox** et le **Button** sont donc instancié sur la seconde ligne.

Exemple : orientation verticale

```
<WrapPanel HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Background="LightGray"
           Height="100"
           Width="300"
           Orientation="Vertical">
    <TextBlock Text="Mots-Clés :"
               Margin="5"
               DockPanel.Dock="Top"/>
    <TextBox Text="Mot-clés"
             Margin="5"
             Width="150"
             DockPanel.Dock="Top"/>
    <CheckBox Content="Limiter aux 10 premiers résultats"
              DockPanel.Dock="Top"
              Margin="5" />
    <Button Content="Rechercher"
            DockPanel.Dock="Top"
            Margin="5" />
</WrapPanel>
```

Résultat :

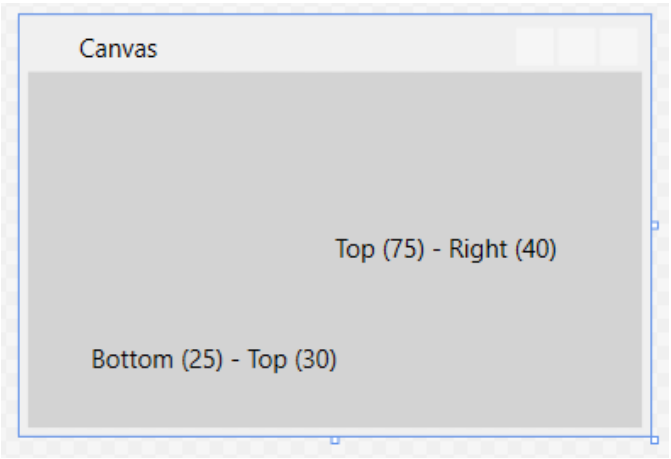
Lorsqu'il s'agit d'instancier le bouton *Recherche*, la colonne courante n'a plus assez d'espace disponible. Le bouton est alors instancié sur la seconde colonne.

Autres contrôles de dispositions.

Contrôle Canvas

Le contrôle **Canvas** permet de définir un contenu de manière absolu. On spécifie les positionnements absolus des éléments contenus. On précise, par exemple, que tel contrôle est situé à *x* pixels du bord gauche et à *y* pixels du bord haut.

```
<Window x:Class="CH3_SOURCE3.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_SOURCE3"
        mc:Ignorable="d"
        Background="LightGray"
        Title=" Canvas" Height="200" Width="300">
    <Grid>
        <Canvas>

            <StackPanel Canvas.Bottom="25" Canvas.Left="30">
                <TextBlock Text="Bottom (25) - Top (30)" />
            </StackPanel>

            <StackPanel Canvas.Top="75" Canvas.Right="40">
                <TextBlock Text="Top (75) - Right (40)" />
            </StackPanel>

        </Canvas>
    </Grid>
</Window>
```



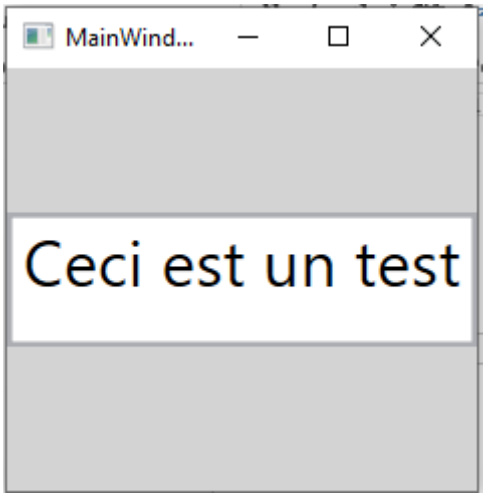
Contrôle ViewBox

Le contrôle **ViewBox** permet l'étirement maximal des éléments contenus s'adaptant ainsi à l'espace disponible (transformation homothétique).

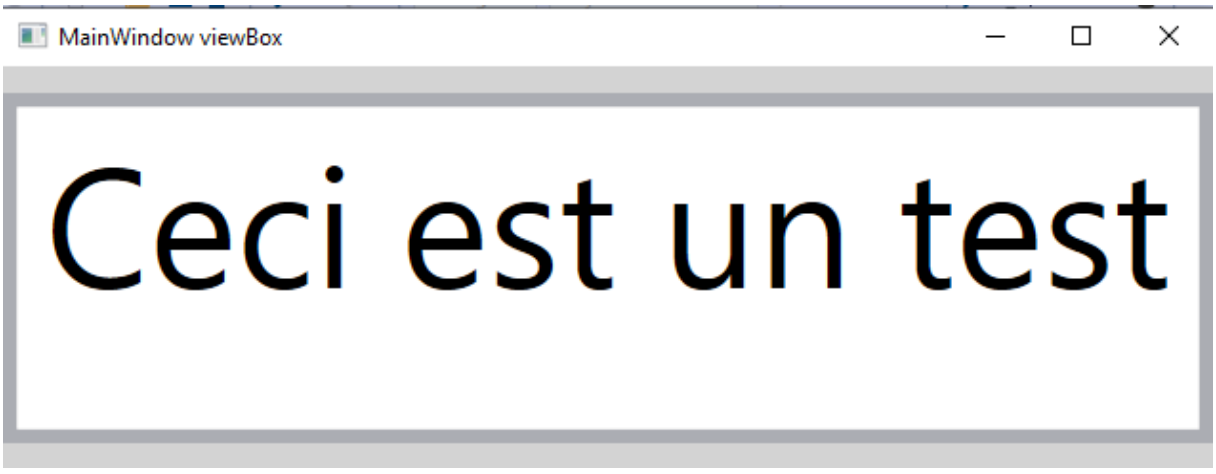
Exemple :

```
<Window x:Class="testch2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:testch2"
        mc:Ignorable="d"
        Background="LightGray"
        Title="MainWindow viewBox" Height="250" Width="250">
    <Grid>
        <Viewbox>
            <TextBox Height="25" Text="Ceci est un test" />
        </Viewbox>
    </Grid>
</Window>
```

Résultat :



En cas d'agrandissement de la fenêtre (`Width="650"`):



Plusieurs modalités d'étirement graphique existent.

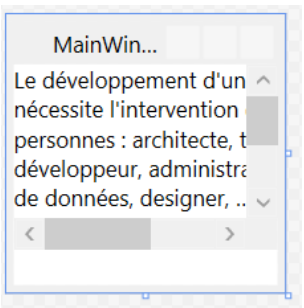
- **Fill** : le contenu est étiré en fonction de la fenêtre et non en fonction des proportions de départ.
- **None** : pas de transformation.
- **Uniform** : (valeur par défaut) le contenu est étiré en fonction des dimensions de la fenêtre mais conserve les proportions de départ).
- **UniformToFill** : le contenu s'ajuste au mieux à la destination sans distorsion (il conserve ses proportions de départ). Par contre, le contenu peut être dérouté pour s'adapter au mieux à la destination.

Contrôle ScrollViewer

Un contrôle de type **ScrollViewer** permet d'associer aux éléments contenus dans la barre de défilement. Certains contrôles possèdent leurs propres propriétés à même d'afficher des barres de défilement mais ce n'est pas toujours le cas. Ce contrôle permet de « scroller » n'importe quel contenu.

```
<Window x:Class="CH3_SCROLL.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_SCROLL"
        mc:Ignorable="d"
        Title="MainWindow" Height="150" Width="150">
    <Grid>
        <ScrollViewer Height="100"
                      VerticalAlignment="Top"
                      ScrollViewer.HorizontalScrollBarVisibility="Auto"
                      ScrollViewer.VerticalScrollBarVisibility="Visible">
                <TextBlock Width="200" Height="150" TextWrapping="Wrap"
                           Text="Le développement d'un logiciel nécessite
l'intervention de différentes personnes : architecte, testeur, développeur,
administrateur de base de données, designer, …  La méthodologie MVVM apporte une
séparation entre le code métier et sa représentation graphique, la production d'un
code qui est testable, une organisation qui facilite la communication, et qui optimise
le travail entre développeurs et designers, une maintenance et une réutilisation
aisée." />
        </ScrollViewer>
    </Grid>
</Window>
```

Résultat



Les propriétés `ScrollViewer.HorizontalScrollBarVisibility` et `ScrollViewer.VerticalScrollBarVisibility` peuvent prendre les valeurs suivantes :

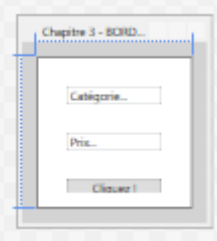
- **Auto** : affichage quand le scrolling est nécessaire.
- **Disabled** : permet de voir les barres de défilement qui ne seront jamais actives
- **Hidden** : permet de masquer les barres de défilement.
- **Visible** : rend toujours les barres de défilement visibles, quel que soit le besoin du scrolling.

Border

Un contrôle de type **Border** permet de munir les éléments contenus de bordures qui les entourent. En effet, la bordure d'un contrôle n'est pas une propriété du contrôle lui-même. Il s'agit donc d'inclure le ou les contrôle(s) dans un contrôle de type bordure.

Exemple :

```
<Window x:Class="CH3_BORDER.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_BORDER"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Chapitre 3 - BORDER" Height="220" Width="200">
    <Grid Margin="15">
        <Border Background="White" BorderBrush="Gray" BorderThickness="2">
            <StackPanel Margin="15">
                <TextBox  Margin="15" Text="Categorie..." />
                <TextBox  Margin="15" Text="Prix..." />
                <Button  Margin="15" Content="Cliquez !"/>
            </StackPanel>
        </Border>
    </Grid>
</Window>
```



Contrôle ItemControl

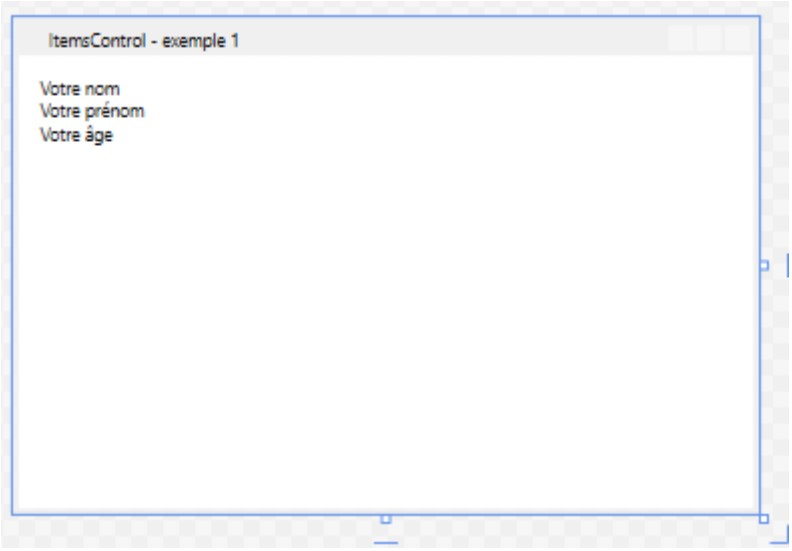
Le contrôle **ItemsControl** permet d'afficher une collection d'entités selon un conteneur donné. Par défaut, le conteneur implicite est un **StackPanel** d'orientation verticale.

Exemple :

```
<Window x:Class="CH3_ItemsControl.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH3_ItemsControl"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        mc:Ignorable="d"
        Title="Chapitre 3 - ItemsControl - exemple 1" Height="350" Width="525">
    <Grid>
        <ItemsControl Margin="15">
            <system:String>Votre nom</system:String>
            <system:String>Votre prénom</system:String>
            <system:String>Votre âge</system:String>
        </ItemsControl>
    </Grid>
</Window>
```

On peut remarquer que la balise *system:String* est directement utilisée dans ce code, ce qui est rendu possible par l'ajout de cet espace de noms

```
xmlns:system="clr-namespace:System;assembly=mscorlib"
```

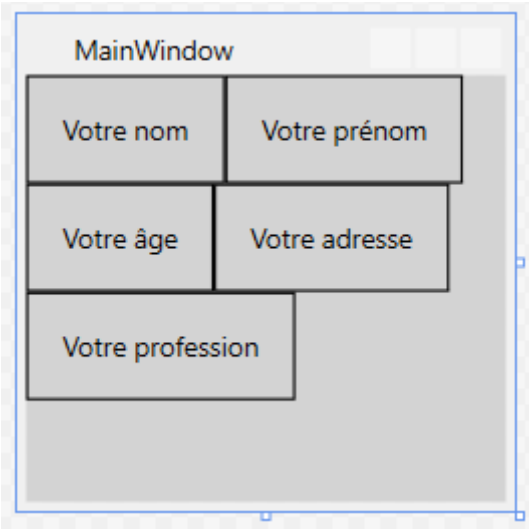


Les trois chaînes de caractères sont alignées verticalement car le conteneur est un **StackPanel** orienté verticalement.

Pour chaque élément de collection, on peut définir les contrôles qui seront utilisés. Pour ce faire, on utilise **ItemsControl.ItemTemplate** et **DataTemplate**.

Exemple : **template** d'élément de collection est un **TextBox** entouré d'une bordure noire.

```
<Window x:Class="CH3_ITEMSCONTROL_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:system="clr-namespace:System;assembly=mscorlib"
        xmlns:local="clr-namespace:CH3_ITEMSCONTROL_2"
        mc:Ignorable="d"
        Background="LightGray"
        Title="MainWindow" Height="220" Width="220">
    <Grid>
        <ItemsControl>
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
            <ItemsControl.ItemTemplate>
                <DataTemplate>
                    <Border BorderThickness="1" BorderBrush="Black">
                        <TextBlock Text="{Binding}" Margin="15" />
                    </Border>
                </DataTemplate>
            </ItemsControl.ItemTemplate>
            <system:String>Votre nom</system:String>
            <system:String>Votre prénom</system:String>
            <system:String>Votre âge</system:String>
            <system:String>Votre adresse</system:String>
            <system:String>Votre profession</system:String>
        </ItemsControl>
    </Grid>
</Window>
```



Principaux contrôles

Contrôles d'affichages

TextBox

Ce contrôle permet l'affichage d'un texte.

```
<TextBox Text="Ceci est un texte" />
```

La propriété **Text** étant la propriété par défaut, on peut écrire la ligne précédente de cette manière :

```
<TextBox>Ceci est un texte</TextBox>
```

TextTrimming

Cette propriété permet de mettre en page le contenu du contrôle. Il permet de tronquer le texte en cas de manque de place en indiquant l'existence de texte complémentaire en affichant trois petits points (…).

TextTrimming peut prendre les valeurs suivantes :

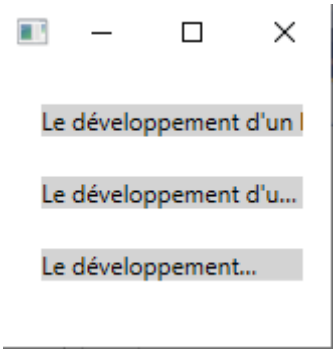
- **WordEllipsis** : tronque au niveau du dernier mot
- **CharacterEllipsis** : tronque au niveau du dernier caractère
- **None**

Exemple :

```
<Window x:Class="CH4_TEXTBLOCK.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_TEXTBLOCK"
        mc:Ignorable="d"
        Title="MainWindow" Height="180" Width="180">
    <Grid Margin="10">

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="150" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="0"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextTrimming="None" />
```

```
        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="1"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextTrimming="CharacterEllipsis" />

        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="2"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextTrimming="WordEllipsis" />
    </Grid>
</Window>
```

Résultat :



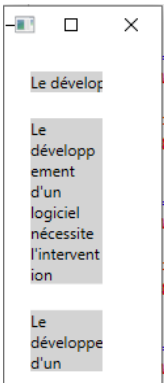
TextWrapping

Cette propriété permet d'optimiser l'affichage du texte en optimisant l'espace disponible. Il peut prendre les valeurs suivantes :

- **Wrap**
- **WrapWithOverflow**
- **NoWrap**

```
<Window x:Class="CH4_TEXTBLOCK_2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_TEXTBLOCK_2"
        mc:Ignorable="d"
        Title="MainWindow" Height="300" Width="100">
    <Grid Margin="10">

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="75" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="0"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextWrapping="NoWrap" />


        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="1"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextWrapping="Wrap" />

        <TextBlock Margin="10"
                   Background="LightGray"
                   Grid.Row="2"
                   Grid.Column="0"
                   Text="Le développement d'un logiciel nécessite l'intervention"
                   TextWrapping="WrapWithOverflow" />
    </Grid>
</Window>
```

Résultat



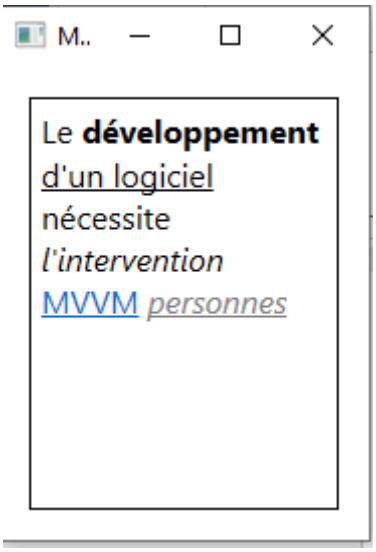
Autres aspect de la mise en forme d'un TextBlock

Voici les principales balises de mises en forme :

- **LineBreak** : permet un passage à la ligne
- **Bold** : permet de mettre en gras une partie du texte
- **Italic** : permet de mettre en italique une partie du texte
- **Hyperlink** : permet d'insérer un lien hypertexte.

```
    Title="MainWindow" Height="275" Width="200">
<Grid Margin="15">
    <Border BorderThickness="1" BorderBrush="Black">
        <TextBlock
                TextWrapping="Wrap"
                FontSize="16"
                Margin="5">
                    Le
                    <Bold>développement </Bold>

                    <LineBreak />

                    <Underline>d'un logiciel </Underline>
                     nécessite
                    <Italic>l'intervention </Italic>

                    <Hyperlink NavigateUri="http://www.editions-eni.fr">MVVM</Hyperlink>

                    <Span TextDecorations="Underline"
                            FontStyle="Italic"
                            Foreground="Gray">personnes </Span>

        </TextBlock>
    </Border>
</Grid>
```

Résultat :



Label

Le label possède un contenu (propriété **Content**), un label peut inclure des contrôles, tel un contrôle de disposition.

Image

L'affichage dans les principaux formats *(.png, .jpg, ..)* utilisés se fait à l'aide de la balise **Image**. La propriété spécifiant l'image à afficher est **Source**. La valeur de cette dernière peut être un chemin physique sur la machine cliente ou une URI.

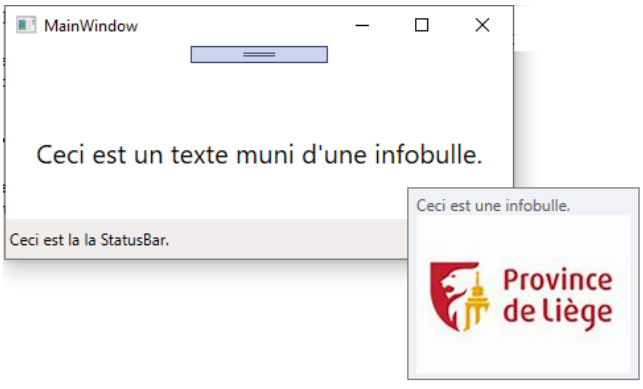
StatusBar et ToolTip

StatusBar permet d'afficher une barre de statut située en bas de la fenêtre.

ToolTip permet l'affichage détaillé de données via des infobulles.

```
<Window x:Class="CH4_STATUSBAR_TOOLTIP.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_STATUSBAR_TOOLTIP"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="400">
    <Grid>

        <TextBlock FontSize="20"
                   VerticalAlignment="Center"
                   HorizontalAlignment="Center"
                   Text="Ceci est un texte muni d'une infobulle.">
            <TextBlock.ToolTip>
                <StackPanel Orientation="Vertical">
                    <TextBlock Text="Ceci est une infobulle." />
                    <Image Source = "logoProvincedeLiège.jpg" />
                </StackPanel>
            </TextBlock.ToolTip>
        </TextBlock>

        <StatusBar Height="30" VerticalAlignment="Bottom">
            <StatusBarItem  >
                Ceci est la la StatusBar.
            </StatusBarItem>
        </StatusBar>
    </Grid>
</Window>
```

Résultat



Contrôle d'édition

TextBox

Dans une fenêtre de type formulaire, il est fréquent d'avoir une **TextBox** (*zone de texte*) permettant d'entrer des données.

PasswordBox

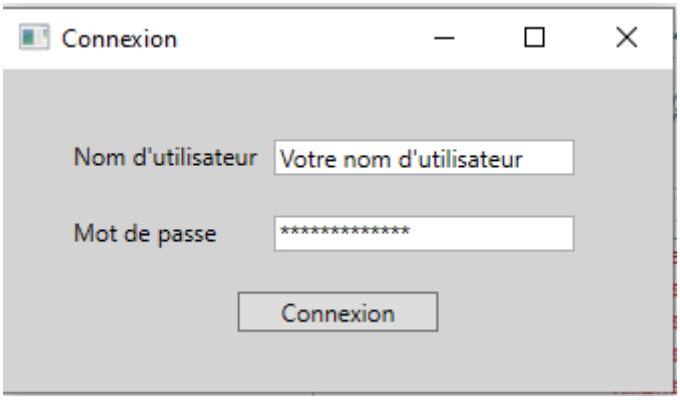
est une zone de texte dédiée à un mot de passe.

La propriété **PasswordChar** : caractère utilisé pour masquer le mot de passe.

Exemple : Affichage d'un formulaire de connexion

```
<Window x:Class="CH4_PASSWORDBOX.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_PASSWORDBOX"
        mc:Ignorable="d"
        Background="LightGray"
        Title="Connexion" Height="200" Width="350">
    <StackPanel Orientation="Vertical" Margin="25">

        <StackPanel Orientation="Horizontal" Margin="10">
            <TextBlock Width="100" Text="Nom d'utilisateur" />
            <TextBox Width="150" Text="Votre nom d'utilisateur" />
        </StackPanel>

        <StackPanel Orientation="Horizontal" Margin="10">
            <TextBlock Width="100" Text="Mot de passe" />
            <PasswordBox Width="150" PasswordChar="*" Password="MonMotDePasse" />
        </StackPanel>

        <Button Width="100"
                Margin="10"
                Content="Connexion"
                HorizontalAlignment="Center" />

    </StackPanel>
</Window>
```

Résultat :



RichTextBox

Ce contrôle permet d'éditer du texte au format RTF

Exemple :

MainWindow.xaml

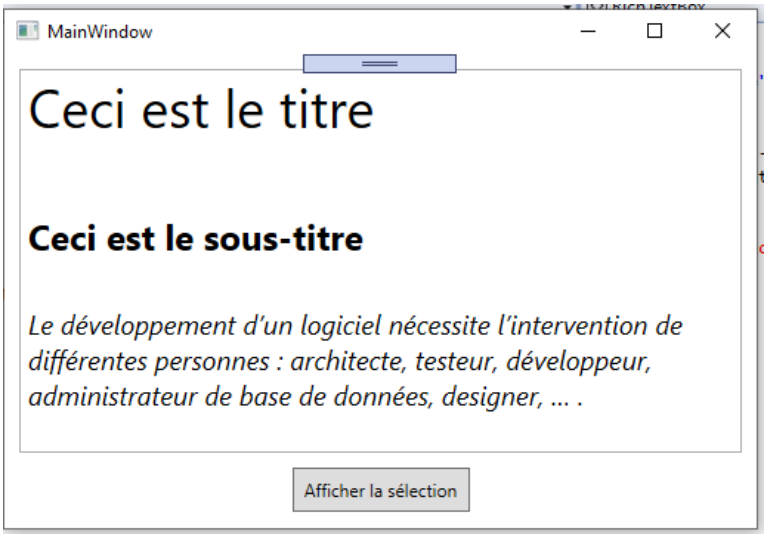
```
<Window x:Class="CH4_RICHTEXTBOX.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_RICHTEXTBOX"
        mc:Ignorable="d"
        Title="MainWindow" Height="360" Width="525">
    <StackPanel Orientation="Vertical">
        <RichTextBox Margin="10" Height="260" VerticalAlignment="Top" Name="MaRichTextBox">
            <FlowDocument>
                <Paragraph FontSize="36">Ceci est le titre</Paragraph>
                <Paragraph FontSize="24" FontWeight="Bold">Ceci est le sous-titre</Paragraph>
                <Paragraph FontStyle="Italic" FontSize="18">Le développement d'un logiciel nécessite l'intervention de 
            </FlowDocument>
        </RichTextBox>
        <Button Height="30" Width="120" Content="Afficher la sélection" Click="Button_Click" />
    </StackPanel>
</Window>
```

MainWindow.xaml.cs

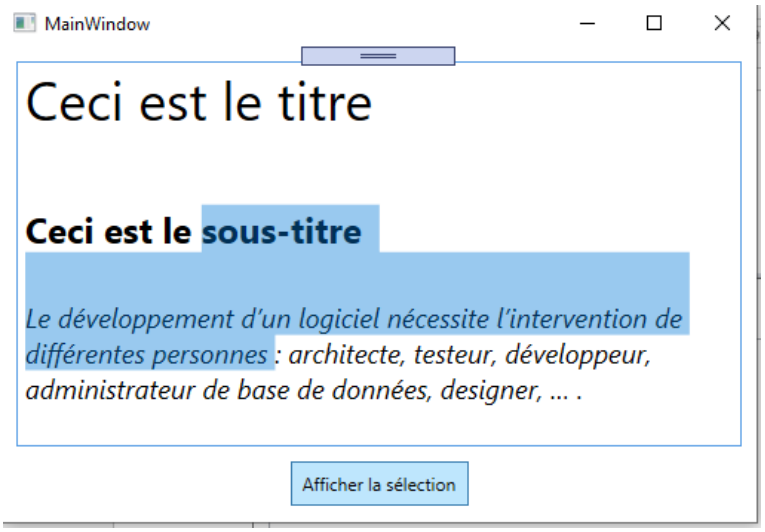
MainWindow.xaml.cs

```
namespace CH4_RICHTEXTBOX
{
    2 références
    public partial class MainWindow : Window
    {
        0 références
        public MainWindow()
        {
            InitializeComponent();
        }

        1 référence
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show( MaRichTextBox.Selection.Text );
        }
    }
}
```

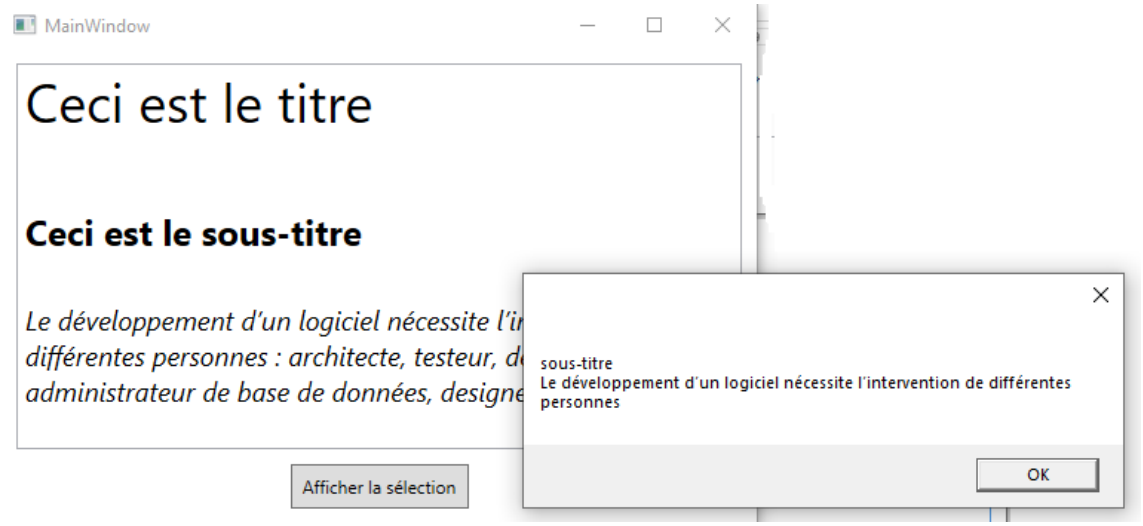
Au démarrage :



- Sélectionner puis cliquer sur *afficher la sélection*



Résultat



Contrôle de sélection de données

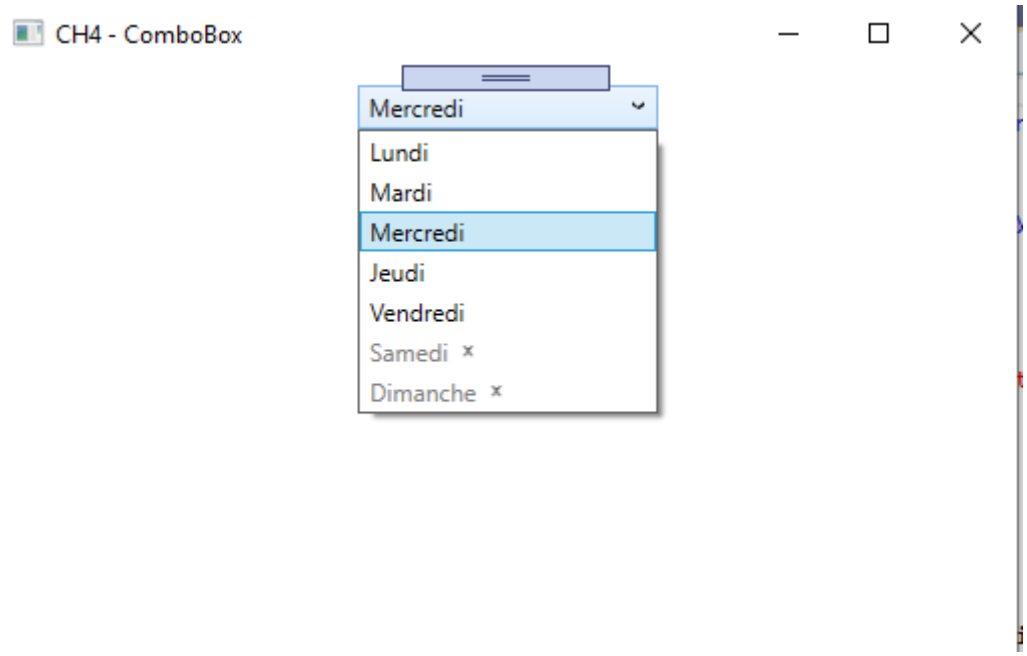
ComboBox

Ce contrôle permet la sélection d'une seule valeur dans une liste de données.

ComboBox dérive du contrôle **ItemsControl**. **ComboBoxItem** dérive de **ITemsControlItem.**

Exemple

```
<Window x:Class="CH4_COMBOBOX.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_COMBOBOX"
        mc:Ignorable="d"
        Title="CH4 - ComboBox" Height="350" Width="525">
    <StackPanel Margin="10">
        <ComboBox Width="150" HorizontalContentAlignment="Left" HorizontalAlignment="Center">
            <ComboBoxItem>Lundi</ComboBoxItem>
            <ComboBoxItem>Mardi</ComboBoxItem>
            <ComboBoxItem IsSelected="True">Mercredi</ComboBoxItem>
            <ComboBoxItem>Jeudi</ComboBoxItem>
            <ComboBoxItem>Vendredi</ComboBoxItem>
            <ComboBoxItem>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Foreground="Gray" Margin="0,0,5,0">Samedi</TextBlock>
                    <Image Source="/Croix.png" Width="10" />
                </StackPanel>
            </ComboBoxItem>
            <ComboBoxItem>
                <StackPanel Orientation="Horizontal">
                    <TextBlock Foreground="Gray" Margin="0,0,5,0">Dimanche</TextBlock>
                    <Image Source="/Croix.png" Width="10" />
                </StackPanel>
            </ComboBoxItem>
        </ComboBox>
    </StackPanel>
</Window>
```



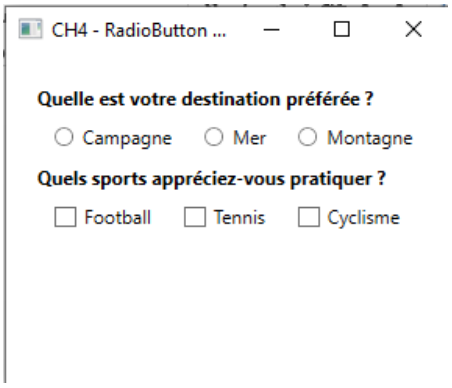
CheckBox et RadioButton

Ces contrôles permettent de sélectionner ou non une valeur. La grande différence entre les deux réside dans le fait que **RadioButton** aoute une exclusivité entre différents choix.

Exemple

```
<Window x:Class="CH4_RADIO_CHECK.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_RADIO_CHECK"
        mc:Ignorable="d"
        Title="CH4 - RadioButton - CheckBox" Height="250" Width="300">
    <StackPanel Orientation="Vertical" Margin="20">
        <TextBlock Text="Quelle est votre destination préférée ?" FontWeight="Bold" />
        <StackPanel Orientation="Horizontal" >
            <RadioButton GroupName="Choix" Content="Campagne" Margin="10 "/>
            <RadioButton GroupName="Choix" Content="Mer" Margin="10 "/>
            <RadioButton GroupName="Choix" Content="Montagne" Margin="10 "/>
        </StackPanel>

        <TextBlock Text="Quels sports appréciez-vous pratiquer ?" FontWeight="Bold" />
        <StackPanel Orientation="Horizontal" >
            <CheckBox Content="Football" Margin="10 "/>
            <CheckBox Content="Tennis" Margin="10 "/>
            <CheckBox Content="Cyclisme" Margin="10 "/>
        </StackPanel>

    </StackPanel>
</Window>
```

Résultat:



Sélection dans des objets complexes

ListBox

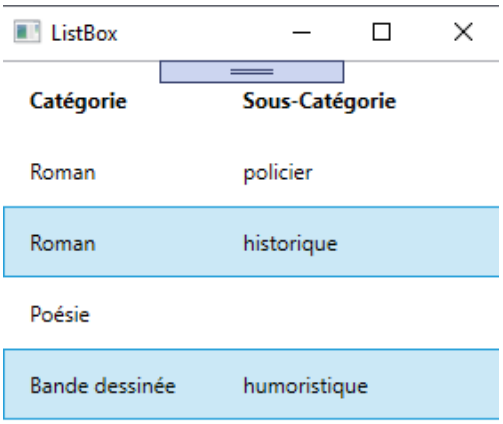
Au niveau de la sélection, elle peut être unique ou multiple selon la valeur de la propriété **SelectionMode** qui peut prendre les valeurs suivantes :

- **Single** : permet une sélection unique. L'élément sélectionné est accessible dans **SelectedItem**
- **Multiple** : permet la solution multiple sans recours au bouton *[shift]*. Les éléments séléctionnées sont accessibles dans **SelectedItems**.
- **Extended** : permet la solution multiple avec recours au bouton *[shift]*. Les éléments séléctionnées sont accessibles dans **SelectedItems**.

Exemple :

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="ListBox" Height="250" Width="300">
    <ListBox Height="250" Width="300" SelectionMode="Multiple">
        <ListBoxItem>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="100" Margin="10"
                           FontWeight="Bold">Catégorie</TextBlock>
                <TextBlock Width="100" Margin="10"
                           FontWeight="Bold">Sous-Catégorie</TextBlock>
            </StackPanel>
        </ListBoxItem>
        <ListBoxItem>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="100" Margin="10">Roman</TextBlock>
                <TextBlock Width="100" Margin="10">policier</TextBlock>
            </StackPanel>
        </ListBoxItem>
        <ListBoxItem>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="100" Margin="10">Roman</TextBlock>
                <TextBlock Width="100" Margin="10">historique</TextBlock>
            </StackPanel>
        </ListBoxItem>
        <ListBoxItem>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="100" Margin="10">Poésie</TextBlock>
                <TextBlock Width="100" Margin="10"></TextBlock>
            </StackPanel>
        </ListBoxItem>
        <ListBoxItem>
            <StackPanel Orientation="Horizontal">
                <TextBlock Width="100" Margin="10">Bande dessinée</TextBlock>
                <TextBlock Width="100" Margin="10">humoristique</TextBlock>
            </StackPanel>
        </ListBoxItem>
    </ListBox>
</Window>
```

Résultat :



ListView et GridView

Le contrôle **ListView** dérive du contrôle **ItemsControl**. **ListView** possède une propriété **View** qui correspond aux données affichées. En général, **View** prend comme valeur **GridView**.

Exemple : **ListView** dont **View** est une **GridView**

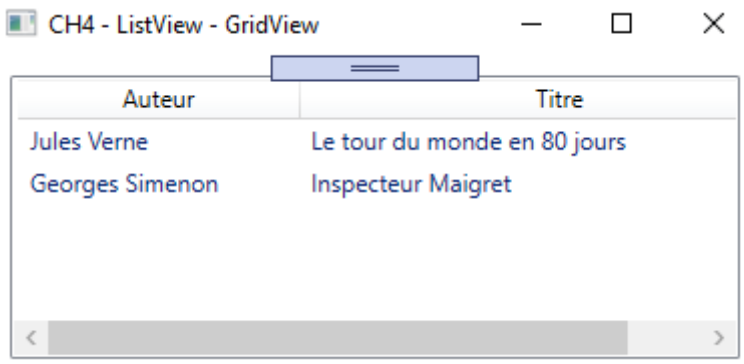
Fichier.xaml

```
<Window x:Class="CH4_GRIDVIEW.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_GRIDVIEW"
        mc:Ignorable="d"
        Title="CH4 - ListView - GridView" Height="200" Width="400">
    <ListView Margin="10" Name="ListViewLivre">
        <ListView.View>
            <GridView>
                <GridViewColumn Header="Auteur" Width="140"
DisplayMemberBinding="{Binding Auteur}" />
                <GridViewColumn Header="Titre" Width="260"
DisplayMemberBinding="{Binding Titre}" />

            </GridView>
        </ListView.View>
    </ListView>
</Window>
```

Fichier.xaml.cs

```
using System.Collections.Generic;
using System.Windows;

namespace CH4_GRIDVIEW
{
    public class Livre
    {
        public string Auteur { get; set;}
        public string Titre { get; set; }

    };

    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            List<Livre> livres = new List<Livre>();
            livres.Add(new Livre() { Auteur = "Jules Verne",
                                     Titre = "Le tour du monde en 80 jours"});
            livres.Add(new Livre() { Auteur = "Georges Simenon",
                                     Titre = "Inspecteur Maigret" });

            ListViewLivre.ItemsSource = livres;
        }
    }
}
```

Résultat :



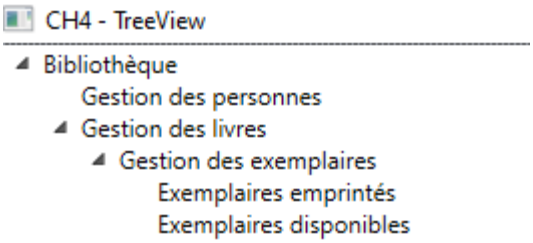
TreeView

Ce contrôle permet la sélection au sein d'un arbre de données. La propriété **TreeViewItem** permet de définir les entités de l'arbre.

Exemple :

```
<Window x:Class="CH4_TREEVIEW.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_TREEVIEW"
        mc:Ignorable="d"
        Title="CH4 - TreeView" Height="250" Width="300">
    <TreeView>
        <TreeViewItem Header="Bibliothèque" IsExpanded="True">
            <TreeViewItem Header="Gestion des personnes"  />
            <TreeViewItem Header="Gestion des livres" IsExpanded="True">
                <TreeViewItem Header="Gestion des exemplaires" IsExpanded="True">
                    <TreeViewItem Header="Exemplaires emprintés" />
                    <TreeViewItem Header="Exemplaires disponibles" />
                </TreeViewItem>
            </TreeViewItem>

        </TreeViewItem>
    </TreeView>
</Window>
```

Résultat :



Introduction aux Data Binding

La liaison des données est une technique générale qui lie les deux sources de données/informations et assure la synchronisation des données entre elles.

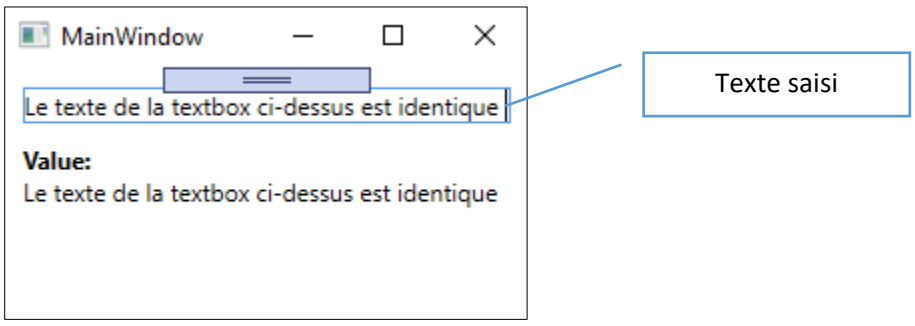
- Créez un nouveau projet WFP et nommer ce projet WPF_App1

Exemple 1 : Syntaxe d'un Binding

- Copiez ceci dans le fichier MainWindow.xaml

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="110" Width="280">
    <StackPanel Margin="10">
        <TextBox Name="txtValue" />
        <WrapPanel Margin="0,10">
            <TextBlock Text="Value: " FontWeight="Bold" />
            <TextBlock Text="{Binding Path=Text, ElementName=txtValue}" />
        </WrapPanel>
    </StackPanel>

</Window>
```

- Exécutez
- Tapez du texte dans le textBox. Que remarquez-vous ?
  Le texte de la TextBlock est synchronisé avec le texte tapé dans le premier TextBlock




```
<TextBlock Text="{Binding Path=Text, ElementName=txtValue}" />
```

Les accolades servent à encapsuler une extension de balisage en XAML. Pour lier (*bind*) des données, on utilise l'extension ***Binding*** qui permet de décrire un Binding pour la propriété *Text.*

Le Path indique vers quelle propriété on effectue le binding. Comme il s'agit de la propriété par défaut, on n'a pas besoin de le préciser, dans ce cas-ci on aurait pu écrire :

```
<TextBlock Text="{Binding Text, ElementName=txtValue}" />
```

Un Binding possède d'autres propriétés, dans l'exemple ci-dessus, on utilise la propriété *ElementName*. Cette propriété permet de se connecter directement à un autre élément de l'interface utilisateur(UI). Les propriétés assignées dans le Binding sont séparées par une virgule.

Exemple 2 : Utilisation du contexte de données (DataContexte)

- Remplacez le fichier MainWindows.xaml par ceci

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="210" Width="280">

    <StackPanel Margin="15">
        <WrapPanel>
            <TextBlock Text="Window title:  " />
            <TextBox Text="{Binding Title, UpdateSourceTrigger=PropertyChanged}"
Width="150" />
        </WrapPanel>
        <WrapPanel Margin="0,10,0,0">
            <TextBlock Text="Window dimensions: " />
            <TextBox Text="{Binding Width}" Width="50" />
            <TextBlock Text=" x " />
            <TextBox Text="{Binding Height}" Width="50" />
        </WrapPanel>
    </StackPanel>

</Window>
```

- Modifiez le fichier MainWindows.xaml.cs pour obtenir ceci :

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = this;
        }
    }
}
```

- Exécutez
- Entrez un autre titre dans le *TextBox Window Title*. Que remarquez-vous ?
  Le titre de la fenêtre est mise à jour

- Modifiez la première dimension ? Que remarquez-vous ?
  La fenêtre est redimensionnée

- Modifiez la deuxième dimension ? Que remarquez-vous ?
  La fenêtre est redimensionnée

Au démarrage, la propriété *DataContext* est égale à null. Un *DataContext* est passé par héritage à tous les contrôles enfants.

L'instruction *this.DataContext = this ;* indique à la classe *Window* qu'il soit lui-même son propre contexte de données. Ici, nous avons utilisé plusieurs propriétés de la classe Window, tels que *Title, Width et Height*.

Cette utilisation du DataContxt permet de définir un contexte global à la fenêtre WPF. En aucun cas cela n'oblige à faire se référer chaque composant à ce contexte. Un composant peut très bien avoir son propre contexte ou temporairement ne pas utiliser le contexte global.

Un autre aspect fondamental relatif au contexte de données est l'héritage. En effet, si le contexte de données est associé à une fenêtre par exemple (*Window*), ce dernier sera partagé par tous les composants et sous-composants compris dans la fenêtre. Le contexte de données est donc associé à un élément et à toute son arborescence-fille.

Exemple 3

Remplacez le fichier MainWindow.xaml par

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="210" Width="280">

    <StackPanel Margin="10">
        <TextBox Name="txtValue" />
        <WrapPanel Margin="0,10">
            <TextBlock Text="Value: " FontWeight="Bold" />
            <TextBlock Name="lblValue" />
        </WrapPanel>
    </StackPanel>
</Window>
```

- Modifiez le fichier MainWindows.xaml.cs pour obtenir ceci :

```
namespace WPF_App1
{
  /// <summary>
  /// Logique d'interaction pour MainWindow.xaml
  /// </summary>
  public partial class MainWindow : Window
  {
    public MainWindow()
    {
      InitializeComponent();

      Binding binding = new Binding("Text");
      binding.Source = txtValue;
      lblValue.SetBinding(TextBlock.TextProperty, binding);

    })
  }}
```

- Exécutez
- Tapez du texte dans le TextBox. Que remarquez-vous ?

Dans cet exemple, nous créons une instance de liaison. Nous spécifions le chemin voulu directement dans le constructeur (*Binding binding = new Binding("Text");*) , en l'occurrence *"Text"*, puisque nous voulons le lier à la propriété Text. Nous spécifions ensuite une Source, dans cet exemple, la source est le contrôle TextBox (*binding.Source = txtValue;*). Maintenant, WPF sait qu'il doit utiliser la TextBox comme source de contrôle, et que nous recherchons spécifiquement la valeur contenue dans sa propriété Text. Nous utilisons la méthode ***SetBinding*** pour combiner notre objet de liaison avec le contrôle de destination/cible : *TextBlock (lblvalue)*. La méthode ***SetBinding()*** prend deux paramètres, le premier indiquant la propriété de dépendance à laquelle on veut faire la liaison et le second contenant l'objet de liaison (*lblValue.SetBinding(TextBlock.TextProperty, binding*);).

Exemple 4

- Remplacez le fichier MainWindow.xaml par

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="130" Width="310">
    <StackPanel Margin="15">
        <WrapPanel>
            <TextBlock Text="Window title:  " />
            <TextBox Name="txtWindowTitle" Text="{Binding Title,
UpdateSourceTrigger=Explicit}" Width="150" />
            <Button Name="btnUpdateSource" Click="btnUpdateSource_Click" Margin="5,0"
Padding="5,0">*</Button>
        </WrapPanel>
        <WrapPanel Margin="0,10,0,0">
            <TextBlock Text="Window dimensions: " />
            <TextBox Text="{Binding Width, UpdateSourceTrigger=LostFocus}" Width="50"
/>
            <TextBlock Text=" x " />
            <TextBox Text="{Binding Height, UpdateSourceTrigger=PropertyChanged}"
Width="50" />
        </WrapPanel>
    </StackPanel>

</Window>
```

- Remplacez le fichier MainWindow.xaml.cs par

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = this;
        }
        private void btnUpdateSource_Click(object sender, RoutedEventArgs e)
        {
            BindingExpression binding =
txtWindowTitle.GetBindingExpression(TextBox.TextProperty);
            binding.UpdateSource();
        }
    }
}
```

- Exécutez
- Changez le titre dans la TextBox. Le titre a-t-il changé ? Non

- Cliquez sur le bouton  Que remarquez-vous ? Le titre de la fenêtre a changé

- Changez la première valeur de la dimension de la fenêtre. Que remarquez-vous ?
  La fenêtre est redimensionnée dès qu'on quitte ce textBox

- Changez la deuxième valeur de la dimension de la fenêtre. Que remarquez-vous ?
  La fenêtre est redimensionnée dès qu'on change de valeur.

La valeur par défaut de ***UpdateSourceTrigger*** est « *Default* ». Les autres options sont *PropertyChanged, LostFocus et Explicit*.

La première est définie comme ***Explicit***, ce qui veut dire que la source ne sera pas mise à jour à moins qu'on le fasse manuellement. Dans cet exemple, on a ajouté un bouton qui va mettre à jour la source sur demande (dès qu'on clique dessus). Dans le Code-behing, on a le handler d l'événement *Click* qui récupère le contrôle de destination pour ensuite appeler la méthode *UpdateSource()* dessus.

La seconde TextBox utilise la valeur ***LostFocus***, qui est en fait la valeur par défaut pour un binding d'une propriété *Text*. Cela signifie que la valeur source sera mise à jour à chaque fois que le contrôle destination perd le focus.

Le troisième TextBox utilise la valeur *PropertyChanged*, qui signifie que la valeur source sera mise à jour chaque fois que la propriété liée change, ce qui se produit dès que le texte change.

Exemple 5 :

- Remplacez le fichier MainWindow.xaml par

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="150" Width="300">
    <DockPanel Margin="10">
        <StackPanel DockPanel.Dock="Right" Margin="10,0,0,0">
            <Button Name="btnAjoutAuteur" Click="btnAjoutAuteur_Click">Ajouter
auteur</Button>
            <Button Name="btnModificationAuteur" Click="btnModificationAuteur_Click"
Margin="0,5">
                Modifier auteur</Button>
            <Button Name="btnSuppressionAuteur" Click="btnSuppressionAuteur_Click">
                Supprimer auteur</Button>
        </StackPanel>
        <ListBox Name="listeAuteurs" DisplayMemberPath="Nom"></ListBox>
    </DockPanel>


</Window>
```

- Changer le fichier MainWindows.xaml.cs pour obtenir ceci :

```
public partial class MainWindow : Window
    {
        private List<Auteur> auteurs = new List<Auteur>();
        public MainWindow()
        {
            InitializeComponent();

            auteurs.Add(new Auteur() { Nom = "Jules Verne" });
            auteurs.Add(new Auteur() { Nom = "Georges Simenon" });

            listeAuteurs.ItemsSource = auteurs;


        }
        private void btnAjoutAuteur_Click(object sender, RoutedEventArgs e)
        {
            auteurs.Add(new Auteur() { Nom = "Nouveau auteur" });
        }

        private void btnModificationAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                (listeAuteurs.SelectedItem as Auteur).Nom = "Random Name";
        }

        private void btnSuppressionAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                auteurs.Remove(listeAuteurs.SelectedItem as Auteur);
        }
    }

    public class Auteur
    {
        public string Nom { get; set; }
    }

}
```

- Exécutez
- Savez-vous ajouter un auteur, modifier un auteur, supprimer un auteur ? Non

La première étape est de faire en sorte que l'interface utilisateur réponde aux changements de source à l'intérieur de la liste (*ItemsSource*), comme quand on ajoute ou supprime un élément dans la liste. WPF fournit un type de liste qui prévient n'importe quelle destination des changements de son contenu. Elle s'appelle ***ObservableCollection*** et elle s'utilise de façon similaire à une *List<T>.*

Dans l'exemple ci-dessous, on a simplement remplacé *List<Auteur>* par *ObservableCollection<Auteur>*

Exemple 6 :

- Changer le fichier MainWindows.xaml.cs pour obtenir ceci :

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Auteur> auteurs = new
ObservableCollection<Auteur>();
        public MainWindow()
        {
            InitializeComponent();

            auteurs.Add(new Auteur() { Nom = "Jules Verne" });
            auteurs.Add(new Auteur() { Nom = "Georges Simenon" });

            listeAuteurs.ItemsSource = auteurs;


        }
        private void btnAjoutAuteur_Click(object sender, RoutedEventArgs e)
        {
            auteurs.Add(new Auteur() { Nom = "Nouveau auteur" });
        }

        private void btnModificationAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                (listeAuteurs.SelectedItem as Auteur).Nom = "Random Name";
        }

        private void btnSuppressionAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                auteurs.Remove(listeAuteurs.SelectedItem as Auteur);
        }
    }

    public class Auteur
    {
        public string Nom { get; set; }
    }


}
```

- Exécutez
- Savez-vous ajouter un auteur, modifier un auteur, supprimer un auteur ?
  ajouter, supprimer : Oui
  modifier : non

Exemple 7 :

- Changer le fichier MainWindows.xaml.cs pour obtenir ceci :

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private ObservableCollection<Auteur> auteurs = new
ObservableCollection<Auteur>();
        public MainWindow()
        {
            InitializeComponent();

            auteurs.Add(new Auteur() { Nom = "Jules Verne" });
            auteurs.Add(new Auteur() { Nom = "Georges Simenon" });

            listeAuteurs.ItemsSource = auteurs;
        }
        private void btnAjoutAuteur_Click(object sender, RoutedEventArgs e)
        {
            auteurs.Add(new Auteur() { Nom = "Nouveau auteur" });
        }

        private void btnModificationAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                (listeAuteurs.SelectedItem as Auteur).Nom = "Random Name";
        }

        private void btnSuppressionAuteur_Click(object sender, RoutedEventArgs e)
        {
            if (listeAuteurs.SelectedItem != null)
                auteurs.Remove(listeAuteurs.SelectedItem as Auteur);
        }
    }

    public class Auteur : INotifyPropertyChanged
    {
        private string nom;

        public string Nom {
                get { return this.nom; }
                set

            {
                    if (this.nom != value)
                    {
                        this.nom = value;
                        this.NotifyPropertyChanged("Nom");
                    }
                }
            }
        public event PropertyChangedEventHandler PropertyChanged;

        public void NotifyPropertyChanged(string propName)
        {
            if (this.PropertyChanged != null)
                this.PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }  }}
```

- Exécutez
- Savez-vous ajouter un auteur, modifier un auteur, supprimer un auteur ? Oui

L'événement ***PropertyChanged*** de type *PropertyChangedEventHandler* se déclenche relativement au nom de la propriété qui change alors de sa valeur.

Lien de commandes

Les commandes ne font en fait rien par elles-mêmes. A la base, elles consistent en une interface *ICommand*, qui définit seulement un événement et deux méthodes : ***Execute()*** et ***CanExecute***. La première réalise effectivement l'action quand la seconde détermine si cette action est disponible. Pour exécuter l'action associée à la commande vous avez besoin d'un lien entre cette commande et votre code et c'est là que le *CommandBinding* entre en jeu.

Un *CommandBinding* est en général défini dans une *Window* ou un *UserControl* et contient d'une part la référence à la commande dont il dépend ainsi que les eventHandlers associés aux événements *Execute()* et *CanExecute()* de la commande.

Les commandes pré-définies

On peut évidemment implémenter ses propres commandes, mais WPF a défini plus de 100 commandes, les plus communément employées, qu'on peut utiliser directement. Elles sont réparties en 5 catégories nommé *ApplicationCommands, NavigationCommands, MediaCommands, EditingCommands et ComponentCommands. ApplicationCommands* en particulier contient les commandes pour un grand nombre d'actions fréquemment utilisées telles que *Nouveau, Ouvrir, Sauvegarder, Couper, Copier et Coller.*

Exemple 8 : utiliser les commandes WPF

- Changer le fichier MainWindows.xaml.cs pour obtenir ceci :

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="100" Width="200">
    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.New"
Executed="NewCommand_Executed" CanExecute="NewCommand_CanExecute" />
    </Window.CommandBindings>

    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button Command="ApplicationCommands.New">New</Button>
    </StackPanel>

</Window>
```

- Changer le fichier MainWindows.xaml.cs pour obtenir ceci :

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
         public MainWindow()
        {
            InitializeComponent();
        }

        private void NewCommand_CanExecute(object sender, CanExecuteRoutedEventArgs e)
        {
            e.CanExecute = true;
        }

        private void NewCommand_Executed(object sender, ExecutedRoutedEventArgs e)
        {
            MessageBox.Show("The New command was invoked");
        }

    }
}
```

- Exécutez
- Tester Que remarquez-vous ?

Dans cet exemple, nous avons défini un CommandBinding dans la Window en l'ajoutant dans sa collection ***CommandBindings*** *(<Window.CommandBindings>).* Nous avons précisé la commande à utiliser (*New* dans *ApplicationCommands*) et ses deux *eventHandler*s (<CommandBinding Command="ApplicationCommands.New" Executed="NewCommand_Executed" CanExecute="NewCommand_CanExecute" />).
L'interface visuelle se compose d'un unique bouton auquel, nous associons la commande à l'aide de la propriété Command (<Button Command="ApplicationCommands.New">New</Button>).

Dans le Code-behind, nous associons les deux événements à leur handler. Le handler **CanExecute**, que WPF appellera quand l'application est libre pour regarder si la commande est disponible, est très simple dans cet exemple car nous souhaitons que cette commande soit toujours disponible. Cela est précisé en affectant la valeur True à la propriété **CanExecute** de l'argument de l'événement.

Le handler **Executed** se contente d'afficher un texte dans une MessageBox quand la commande est appelée. Lorsque nous exécutons cet exemple et cliquons sur le bouton nous voyons ce message.

Exemple 9 : Utiliser la méthode CanExecute

Dans l'exemple 8, nous avons implémenté un bouton qui est toujours accessible (*CanExecute* retourne True). Dans de nombreux cas, on souhaite que le bouton soit activé ou désactivé suivant un certain statut défini dans notre application. L'exemple est la bascule de boutons pour utiliser le presse-papier de Window. Les boutons *Couper et Copier* sont activés uniquement lorsqu'un texte est sélectionné et le bouton *Coller* est activé lorsqu'un texte est présent dans le presse-papier.

```
<Window x:Class="WPF_App1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_App1"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="250">
    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.Cut"
CanExecute="CutCommand_CanExecute" Executed="CutCommand_Executed" />
        <CommandBinding Command="ApplicationCommands.Paste"
CanExecute="PasteCommand_CanExecute" Executed="PasteCommand_Executed" />
    </Window.CommandBindings>
    <DockPanel>
        <WrapPanel DockPanel.Dock="Top" Margin="3">
            <Button Command="ApplicationCommands.Cut" Width="60">_Cut</Button>
            <Button Command="ApplicationCommands.Paste" Width="60"
Margin="3,0">_Paste</Button>
        </WrapPanel>
        <TextBox AcceptsReturn="True" Name="txtEditor" />
    </DockPanel>

</Window>
```

```
namespace WPF_App1
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void CutCommand_CanExecute(object sender, CanExecuteRoutedEventArgs e)
        {
            e.CanExecute = (txtEditor != null) && (txtEditor.SelectionLength > 0);
        }
        private void CutCommand_Executed(object sender, ExecutedRoutedEventArgs e)
        {
            txtEditor.Cut();
        }
      private void PasteCommand_CanExecute(object sender, CanExecuteRoutedEventArgs e)
        {
            e.CanExecute = Clipboard.ContainsText();
        }
        private void PasteCommand_Executed(object sender, ExecutedRoutedEventArgs e)
        {
            txtEditor.Paste();
        } } }
```

Sélection de dates

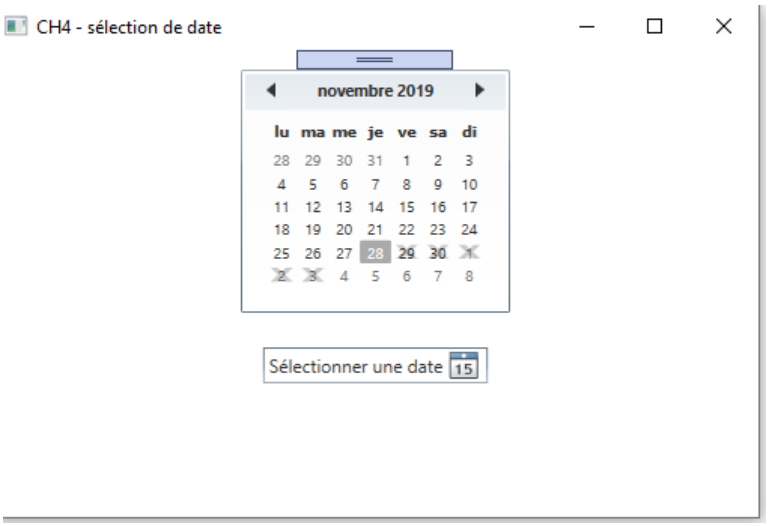
Fichier.xaml

```
<Window x:Class="CH4_DATE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_DATE"
        mc:Ignorable="d"
        Title="CH4 - sélection de date" Height="350" Width="525">
    <StackPanel Orientation="Vertical">

        <Calendar DisplayDate="2019-11-28"
                  DisplayMode="Month"
                  SelectionMode="MultipleRange"
                  Margin="10">
            <Calendar.BlackoutDates>
                <CalendarDateRange Start="2019-11-29" End="2019-12-03" />
            </Calendar.BlackoutDates>
        </Calendar>

        <DatePicker HorizontalAlignment="Center"
                    Margin="10"
                    Name="DatePicker"
                    VerticalAlignment="Top"
                    SelectedDateChanged="DatePicker_SelectedDateChanged"/>
    </StackPanel>
</Window>
```

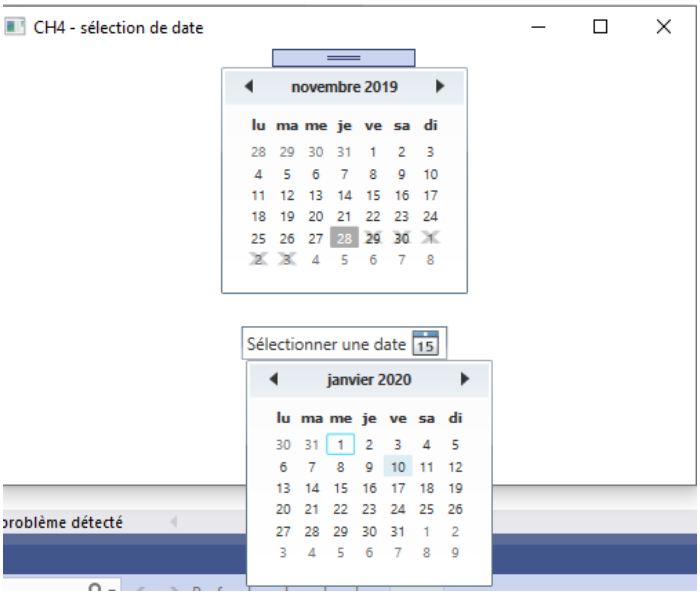
Fichier.xaml.cs

```
using System.Windows;
using System.Windows.Controls;

namespace CH4_DATE
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void DatePicker_SelectedDateChanged(object sender, SelectionChangedEventArgs e)
        {
            MessageBox.Show(DatePicker.DisplayDate.ToString());
        }
    }
}
```

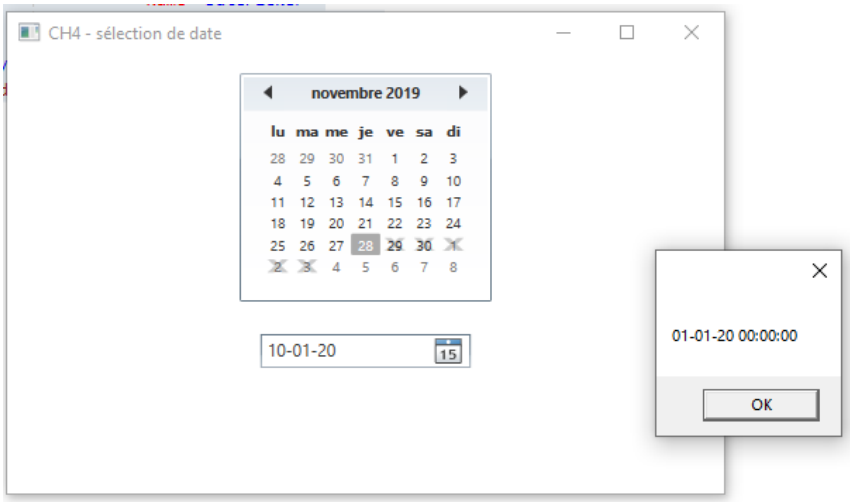
Au démarrage



- Selection d'une date



Après sélection d'une date (10 janvier 2020)



Contrôles d'action utilisateur

Les principaux contrôles d'action sont

- Le menu classique
- Un bouton ouvrant une boîte de message sur li clic gauche
- Un menu contextuel sur le clic droit

Le menu ou le **ContextMenu** se base sur des propriétés **MenuItem** pour décrire les menus.

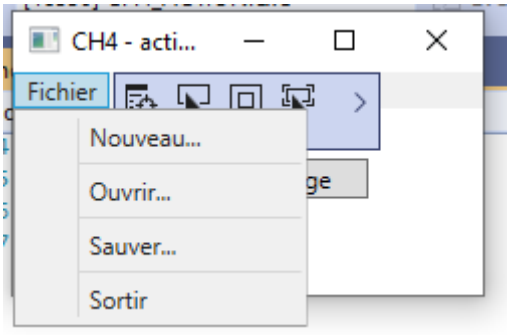
Fichier.xaml

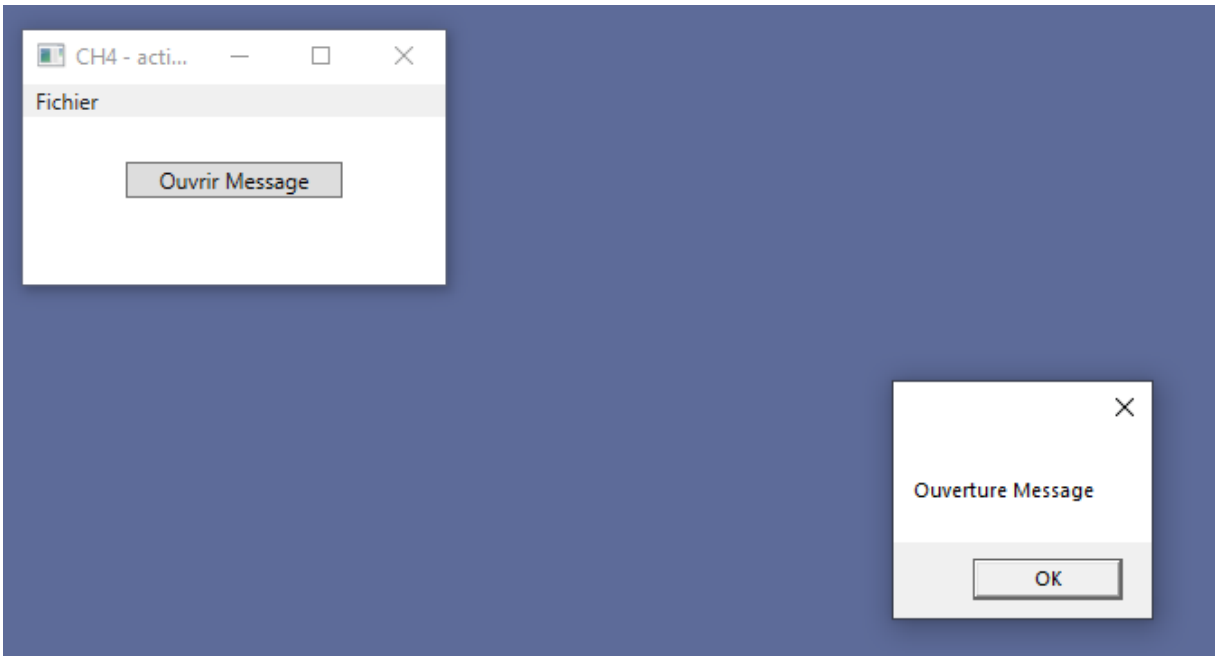
```
<Window x:Class="CH4_ACTION.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_ACTION"
        mc:Ignorable="d"
        Title="CH4 - actions" Height="150" Width="250">
    <StackPanel Orientation="Vertical">

        <Menu>
            <MenuItem Header="Fichier">
                <MenuItem Header="Nouveau..." />
                <Separator />
                <MenuItem Header="Ouvrir..." />
                <Separator />
                <MenuItem Header="Sauver..." />
                <Separator />
                <MenuItem Header="Sortir" />
            </MenuItem>
        </Menu>

        <Button Margin="25" Width="120" HorizontalAlignment="Center"
                Click="Button_Click"
                Content="Ouvrir Message">

            <Button.ContextMenu>
                <ContextMenu>
                    <MenuItem Header="Compilation" />
                </ContextMenu>
            </Button.ContextMenu>

        </Button>

    </StackPanel>
</Window>
```

Fichier.xaml.cs

```
namespace CH4_ACTION
{
    /// <summary>
    /// Logique d'interaction pour MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Ouverture Message");
        }
    }
}
```

- Clique sur ouvrir message la deuxième fenêtre apparaît





Fenêtrage

Window

Propriétés

- **Title**: permet de nommer la fenêtre. Elle s'affiche en haut de la fenêtre dans la barre de titre.
- **WindowState** : trois valeurs : ***Maximized, Minimized, Normal***
- **ResizeMode**: définir les modes de redimensionnement de la fenêtre, les valeurs possibles sont ***CanMinimize, CanResize, CanResizeWithGrip, NoResize***.
- **ShowInTaskbar** : valeur ***True*** ou ***false*** : indicateur si la fenêtre fait l'objet dans la barre des tâches Wndows.
- **WindowsStartupLocation** : emplacement d'affichage de la fenêtre. Les valeurs possibles sont ***CenterScreen, Manual, CenterOwner***.

Exemple

```
<Window x:Class="CH4_WINDOW.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH4_WINDOW"
        mc:Ignorable="d"
        Title="CH4 - Window" Height="200" Width="200"
        WindowState="Maximized"
        ShowInTaskbar="False"
        WindowStartupLocation="CenterScreen"
        ResizeMode="NoResize" />
```

Lancement depuis l'application WPF

La fenêtre Window est en général lancée depuis l'application, les modalités du lancement étant définies, notamment le nom de la fenêtre à lancer, dans le fichier *App.xaml*. Dans ce dernier, une balise **Application** possède une propriété **StartupUri** qui contient le nom de la fenêtre à lancer :

```
<Application x:Class="CH4_WINDOW.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:CH4_WINDOW"
             StartupUri="MainWindow.xaml" />
```

NavigationWindow

Le contrôle **NavigationWindow** est une alternative au contrôle **Window**. Il possède la spécificité d'avoir un comportement de navigateur web, gérant un historique de navigation et envisageant chaque fichier XAML affichée comme une page web : pour cela, il faut que le fichier XAML soit défini grâce à une balise **Page** et non Window.

Exemple : une **NavigationWindow** dont la source est la **Page** *Maison.xaml*

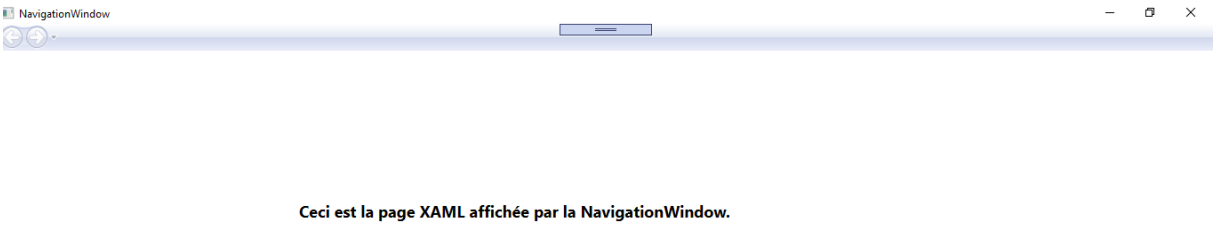
NavigationWindow.xaml

```
<NavigationWindow x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApplication1"
        mc:Ignorable="d"
        Source="Maison.xaml"
        Title="NavigationWindow" Height="350" Width="600">

</NavigationWindow>
```

Maison.xaml

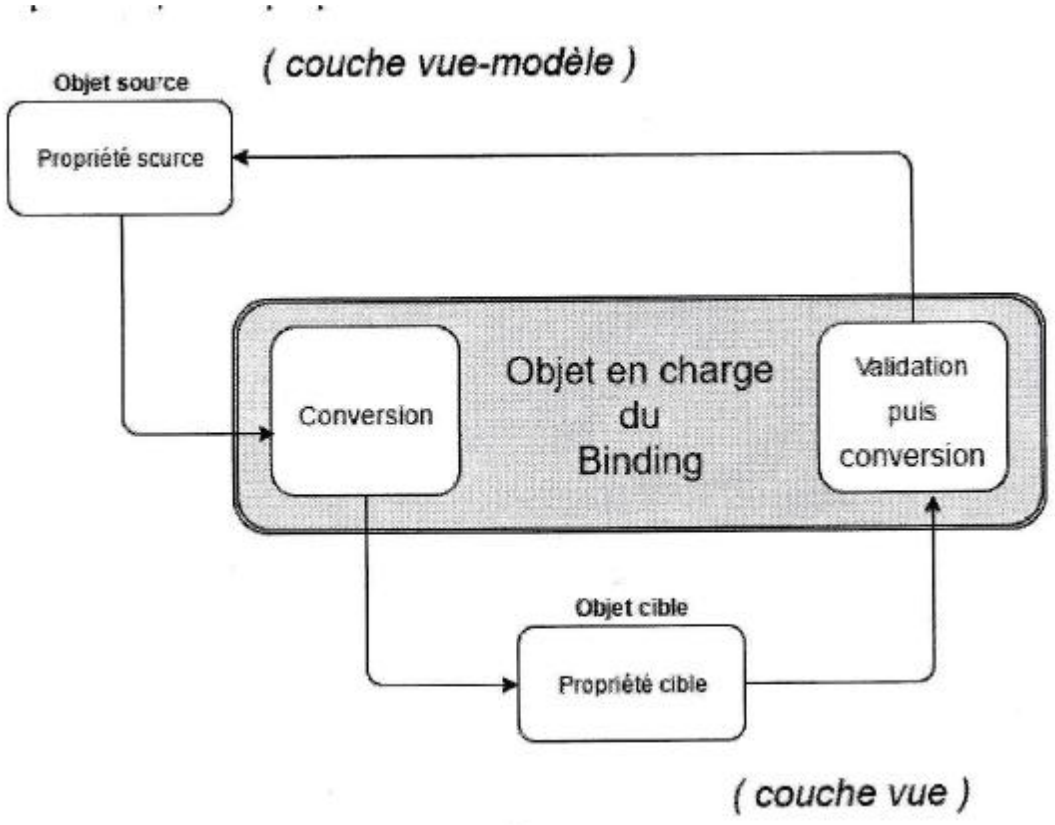
```
<Page x:Class="WpfApplication1.Maison"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:WpfApplication1"
      mc:Ignorable="d"
      Width="700"
      Title="Maison">
    <WrapPanel Width="700"
               Height="350"
               HorizontalAlignment="Center"
               VerticalAlignment="Center">
        <Label Content="Ceci est la page XAML affichée par la NavigationWindow."
               FontWeight="Bold"
               FontSize="18" />
    </WrapPanel>
</Page>
```

Résultat



DataBinding (liaison de données)

Le binding est un mécanisme qui lie deux sources de données et qui assure la synchronisation.



Ce schéma ne présente pas un cycle. Il met en exergue la synchronisation d'une propriété de la vue-modèle vers la vue et réciproquement.

L'objet source situé dans la couche vue-modèle contient une propriété source qui fait l'objet d'une synchronisation via le binding.

Coté vue, un objet, pas nécessairement visuel, contient une propriété **cible** qu'elle aussi peut faire l'objet d'une synchronisation via le binding.

L'objet **Binding** à même de réaliser le Binding dans les deux sens.

Binding côté vue exclusivement

Côté vue, dans le code XAML, l'objet ou la propriété « bindée » utilise une expression « Binding » pour réaliser cette synchronisation.

L'objet **Binding** statique utilise trois propriétés : **Source, RelativeSource** ou **ElementName** pour se lier avec l'objet source. Le Binding ne se réalise pas nécessairement avec la vue-modèle.

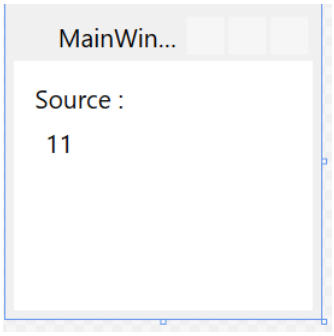
Propriété Source

Cette propriété permet de spécifier un Binding dont le chemin (*path*) est totalement connu et ne dépend pas du **DataContext**.

Exemple : affiche le mois courant dans un **TextBlock**

```
<Window x:Class="CH5_EX1_SOURCE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX1_SOURCE"
        mc:Ignorable="d"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        Title="MainWindow" Height="150" Width="150">
    <StackPanel Margin="10" Background="White">

        <TextBlock Text="Source :" />
        <TextBlock Margin="5"
            Text="{Binding Source={x:Static sys:DateTime.Today}, Path=Month}" />

    </StackPanel>
</Window>
```

Nous sommes au mois de novembre (donc 11 sera affiché)



Propriété RelativeSource

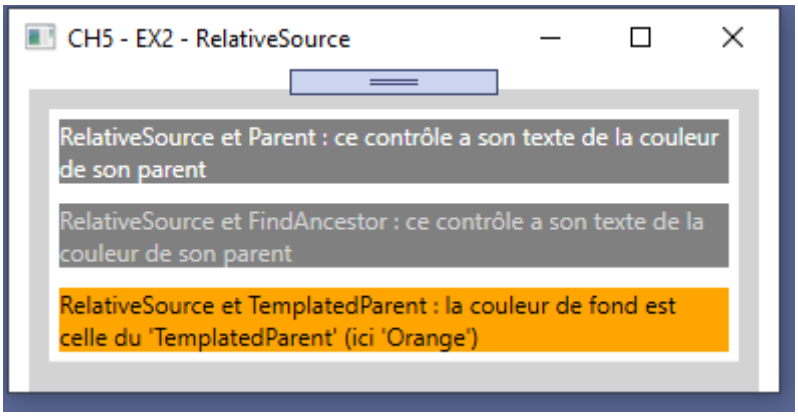
Cette propriété est relative contrairement à la propriété précédente, c'est-à-dire qu'elle envisage comme contexte l'arbre visuel défini par le code XAML dans lequel le contrôle s'insère.

Dans cet arbre visuel, on peut rechercher des informations sur le contrôle courant (**self**) ou sur un ancêtre, une balise située plus haut dans l'arbre (**FindAncestor**). Une dernière utilisation possible est d'appliquer un Binding aux éléments soumis à un template donné (**DataTemplate**).

Exemple :

- **Self** : une première écriture voit sa couleur de texte identique à la couleur de fond de son parent.
- **FindAncestor** : une seconde écriture voit sa couleur de texte identique à la couleur de fond d'un ancêtre donné (**StackPane**l)
- **TemplateParent** : une troisième écriture voit sa couleur de fond issue du contrôle qui accueille son Template.

```
<Window x:Class="CH5_EX2_RELATIVESOURCE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX2_RELATIVESOURCE"
        mc:Ignorable="d"
        Title="CH5 - EX2 - RelativeSource" Height="200" Width="400">
    <StackPanel Margin="10,10,10,0"  Background="LightGray" Height="249" VerticalAlignment="Top">

        <StackPanel Margin="10" Background="White">

            <TextBlock Margin="5"
            Background="Gray"
            Text="RelativeSource et Parent : ce contrôle a son texte de la couleur de son parent"
            TextWrapping="Wrap"
            Foreground="{Binding RelativeSource={RelativeSource Self}, Path=Parent.Background}"/>

            <TextBlock Margin="5"
            Background="Gray"
            Text="RelativeSource et FindAncestor : ce contrôle a son texte de la couleur de son parent"
            TextWrapping="Wrap"
            Foreground="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type StackPanel}, AncestorLevel=2}, Path=Background}"/>

            <ItemsControl Background="Orange" Margin="5">
                <TextBlock TextWrapping="Wrap">RelativeSource et TemplatedParent : la couleur de fond est celle du 'TemplatedParent' (ici 'Orange')</TextBlock>
                <ItemsControl.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Background="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=Background}"/>
                        </StackPanel>
                    </DataTemplate>
                </ItemsControl.ItemTemplate>
            </ItemsControl>


        </StackPanel>
    </StackPanel>

</Window>
```

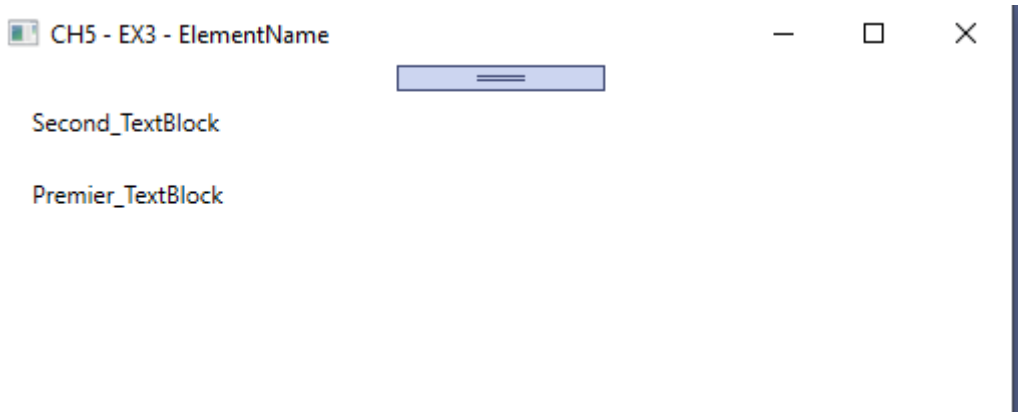


Propriété ElementName

Cette propriété permet de référencer un élément de la vue par son nom. Ainsi, une première **TextBlock** affiche le nom d'une seconde **TextBlock** et réciproquement. Le référencement de chaque contrôle se fait via le nom de chacune d'elles.

Exemple :

```
<Window x:Class="CH5_EX3_ElementName.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX3_ElementName"
        mc:Ignorable="d"
        Title="CH5 - EX3 - ElementName" Height="350" Width="525">
    <StackPanel Margin="10">
        <TextBlock Margin="10" Name="Premier_TextBlock" Text="{Binding ElementName=Second_TextBlock, Path=Name}" />
        <TextBlock Margin="10" Name="Second_TextBlock" Text="{Binding ElementName=Premier_TextBlock, Path=Name}" />
    </StackPanel>
</Window>
```



Binding entre vue et vue-modèle

Présentation de l'objet de Binding

En général, l'objet est syntaxiquement manipulé dans la source XAML. Mais, il est tout à fait possible de l'utiliser via du code C#.

Les principales propriétés de l'objet **Binding** sont les suivantes :

- **Source** : permet de définir la source du Binding
- **Path** : permet d'associer l'attribut de classe de la Source qui sera précisément « bindé ».
- **Mode** : plusieurs valeurs existent pour réaliser le binding : bidirectionnel ou unidirectionnel ?
- **UpdateSourceTrigger** : si le mode permet à la vue de modifier la vue-modèle, sur quel événement va se déclencher la mise à jour ?

Propriété Mode de l'objet de binding

- **OneWay** : le binding est unidirectionnel, de la source vers la cible (de la vue-modèle vers la vue)
- **OneWayToSource** : le binding est unidirectionnel, de la cible vers la source (de la vue vers la vue-modèle).
- **TwoWay** : le Binding est bidirectionnel. La vue-modèle affecte la vue et réciproquement.
- **OneTime** : correspond à un Binding OneWay qui ne s'effectue que la première fois.
- **Default** : c'est un Binding par défaut qui dépend du contrôle considéré. Par exemple TextBox, ComboBox, MenuItem, la valeur par défaut est TwoWay.

Propriété UpdateSourceTrigger de l'objet de binding

Cette propriété permet d'indiquer l'événement qui va notifier le Binding et donc déclencher le besoin de synchronisation.

Les valeurs sont :

- **Explicit** : la synchronisation sera effectuée sur l'appel explicite de la méthode **UpdateSource**.
- **LostFocus** : la synchronisation de la source est réalisée dès perte de focus par le contrôle.
- **PropertyChanged** : la synchronisation de la source est réalisée à chaque changement de valeur dans le contrôle.
- **Default** : correspond à la valeur par défaut associé à la propriété du contrôle considéré (Souvent : **PropertyChanged**)

Exemple Binding défini en C# (entre vue-modèle et vue)

MainWindow.xaml

```
<Window x:Class="CH5_EX4_BINDING_CODE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX4_BINDING_CODE"
        mc:Ignorable="d"
        Background="LightGray"
        Title="CH5 - EX4 - BINDING C#" Height="224.645" Width="346.277">
    <StackPanel Orientation="Vertical" Margin="10">
        <TextBox Margin="10" Name="ISBN" />
        <TextBox Margin="10" Name="Auteur" />
        <TextBox Margin="10" Name="Titre" />
        <TextBox Margin="10" Name="Edition" />
    </StackPanel>
</Window>
```

MainWindow.xaml.cs

```
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace CH5_EX4_BINDING_CODE
{
    public class Livre
    {
        public string ISBN { get; set; }
        public string Auteur { get; set; }
        public string Titre { get; set; }
        public string Modele { get; set; }

        public Livre(string i, string a, string t, string e)
        {
            this.ISBN = i;
            this.Auteur = a;
            this.Titre = t;
            this.Modele = e;
        }
    };


   public class VueModele : INotifyPropertyChanged
   {
       public Livre livre { get; set; }
       public VueModele()
       {
           this.livre = new Livre("978123456", "Jules Verne",
               "Le tour du monde en 80 jours", "Livre de poche");
       }

       public event PropertyChangedEventHandler PropertyChanged;
       protected virtual void OnPropertyChanged(string propertyName)
       {
           if (this.PropertyChanged != null)
           {
               this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
           }
       }
   }
```

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        this.DataContext = new VueModele();
        VueModele monContexte = (VueModele)this.DataContext;

        Binding monBindingISBN = new Binding();
        monBindingISBN.Source = monContexte;
        monBindingISBN.Path = new PropertyPath("ISBN");
        monBindingISBN.Mode = BindingMode.TwoWay;
        monBindingISBN.UpdateSourceTrigger = UpdateSourceTrigger.LostFocus;
        BindingOperations.SetBinding(ISBN, TextBox.TextProperty, monBindingISBN);

        Binding monBindingAuteur = new Binding();
        monBindingAuteur.Source = monContexte;
        monBindingAuteur.Path = new PropertyPath("Auteur");
        monBindingAuteur.Mode = BindingMode.TwoWay;
        monBindingAuteur.UpdateSourceTrigger = UpdateSourceTrigger.LostFocus;
        BindingOperations.SetBinding(Auteur, TextBox.TextProperty, monBindingAuteur);

        Binding monBindingTitre = new Binding();
        monBindingTitre.Source = monContexte;
        monBindingTitre.Path = new PropertyPath("Titre");
        monBindingTitre.Mode = BindingMode.TwoWay;
        monBindingTitre.UpdateSourceTrigger = UpdateSourceTrigger.LostFocus;
        BindingOperations.SetBinding(Titre, TextBox.TextProperty, monBindingTitre);

        Binding monBindingEdition = new Binding();
        monBindingEdition.Source = monContexte;
        monBindingEdition.Path = new PropertyPath("Edition");
        monBindingEdition.Mode = BindingMode.TwoWay;
        monBindingEdition.UpdateSourceTrigger = UpdateSourceTrigger.LostFocus;
        BindingOperations.SetBinding(Edition, TextBox.TextProperty, monBindingEdition);
    }
}
```

```
    }
}
```

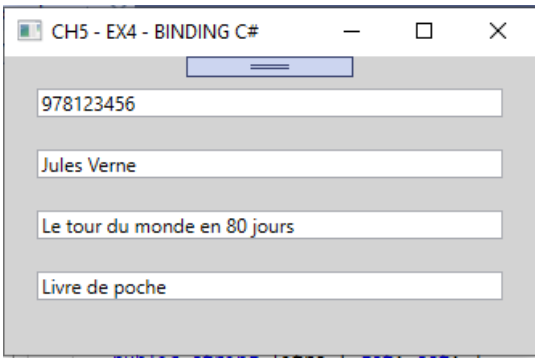
On remarque dans le code précédent que la création du binding est faite de manière explicite en C# :

```
Binding monBindingISBN = new Binding();
            monBindingISBN.Source = monContexte;
            monBindingISBN.Path = new PropertyPath("ISBN");
            monBindingISBN.Mode = BindingMode.TwoWay;
            monBindingISBN.UpdateSourceTrigger = UpdateSourceTrigger.LostFocus;
```

Elle se termine par une association entre **DependancyObject**, **DependancyProperty** et l'objet **Binding** créé.

```
BindingOperations.SetBinding(ISBN, TextBox.TextProperty, monBindingISBN);
```

Résultat:



Même exemple défini en XAML

MainWindows.xaml

```
<Window x:Class="CH5_EX4_BINDING_XAML.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX4_BINDING_XAML"
        mc:Ignorable="d"
        Background="LightGray"
        Title="MainWindow" Height="250" Width="250">
    <StackPanel Orientation="Vertical" Margin="10">
        <TextBox Margin="10" Text="{Binding ISBN, Mode=TwoWay, UpdateSourceTrigger=LostFocus}" />
        <TextBox Margin="10" Text="{Binding Auteur, Mode=TwoWay, UpdateSourceTrigger=LostFocus}" />
        <TextBox Margin="10" Text="{Binding Titre, Mode=TwoWay, UpdateSourceTrigger=LostFocus}" />
        <TextBox Margin="10" Text="{Binding Edition, Mode=TwoWay, UpdateSourceTrigger=LostFocus}" />
    </StackPanel>
</Window>
```

MainWindow.xaml.cs

```
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;

namespace CH5_EX4_BINDING_XAML
{

    public class Livre
    {
        public string ISBN { get; set; }
        public string Auteur { get; set; }
        public string Titre { get; set; }
        public string Edition { get; set; }

        public Livre(string i, string a, string t, string e)
        {
            this.ISBN = i;
            this.Auteur = a;
            this.Titre = t;
            this.Edition = e;
        }
    };
```

```
public class VueModele : INotifyPropertyChanged
{
    public Livre livre { get; set; }
    public VueModele()
    {
        this.livre = new Livre("978123456", "Jules Verne", "Le tour du monde en 80 jours", "Livre de poche");
    }

    public event PropertyChangedEventHandler PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
        if (this.PropertyChanged != null)
        {
            this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    public string ISBN
    {
        get { return this.livre.ISBN; }
        set
        {
            this.livre.ISBN = value;
            OnPropertyChanged("ISBN");
        }
    }
    public string Auteur
    {
        get { return this.livre.Auteur; }
        set { this.livre.Auteur = value; OnPropertyChanged("Auteur"); }
    }
    public string Titre
    {
        get { return this.livre.Titre; }
        set { this.livre.Titre = value; OnPropertyChanged("Titre"); }
    }
```

```
        public string Edition
        {
            get { return this.livre.Edition; }
            set { this.livre.Edition = value; OnPropertyChanged("Edition"); }
        }
    };

    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new VueModele();
        }
    }
}
```

Dans cet exemple, c'est le code ci-dessous qui se charge du binding, au sein du **DataContext** affecté dans le constructeur de la **Window**.

```
<TextBox Margin="10" Text="{Binding ISBN, Mode=TwoWay,
UpdateSourceTrigger=LostFocus}" />
```

Binding de collections

Quand on blinde une collection avec un contrôle (ITemsControl, ListBox, ListView, ComboBox, DataGrid, …) qui affiche une collection, l'idée est de blinder le contenu de la collection elle-même (élément inséré, élément supprimé) et blinder chaque élément de la collection.

- **INotifyCollectionChanged** participe à l'observation de la collection elle-même (éléments insérés, supprimés, …).
- **InotifyPropertyChanged** implémentée par chaque élément de la collection permet d'observer chaque élément lui-même et notamment ses composantes éventuellement modifiées.

Binding avec ObservableCollection<T>

Soit une grille comprenant une liste de voitures (immatriculation, couleur, marque, modèle) modifiable et dans laquelle l'utilisateur souhaite pouvoir supprimer une ligne ou en ajuter une nouvelle.

MainWindows.xaml

```
<Window x:Class="CH5_EX5_COLLECTION.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX5_COLLECTION"
        mc:Ignorable="d"
        Background="LightGray"
        Title="CH5 - EX5 - ObservableCollection" Height="450" Width="550">
    <StackPanel VerticalAlignment="Top">

        <DataGrid Width="500"
                  Margin="10"
                  Name="MaGrille"
                  HorizontalAlignment="Center"
                  ItemsSource="{Binding Collection}"
                  AutoGenerateColumns="True" />

        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button Margin="10" Content="Ajout" Click="Ajout_Click" />
            <Button Margin="10" Content="Suppression" Click="Suppression_Click" />
        </StackPanel>

    </StackPanel>
</Window>
```

MainWindows.xaml.cs

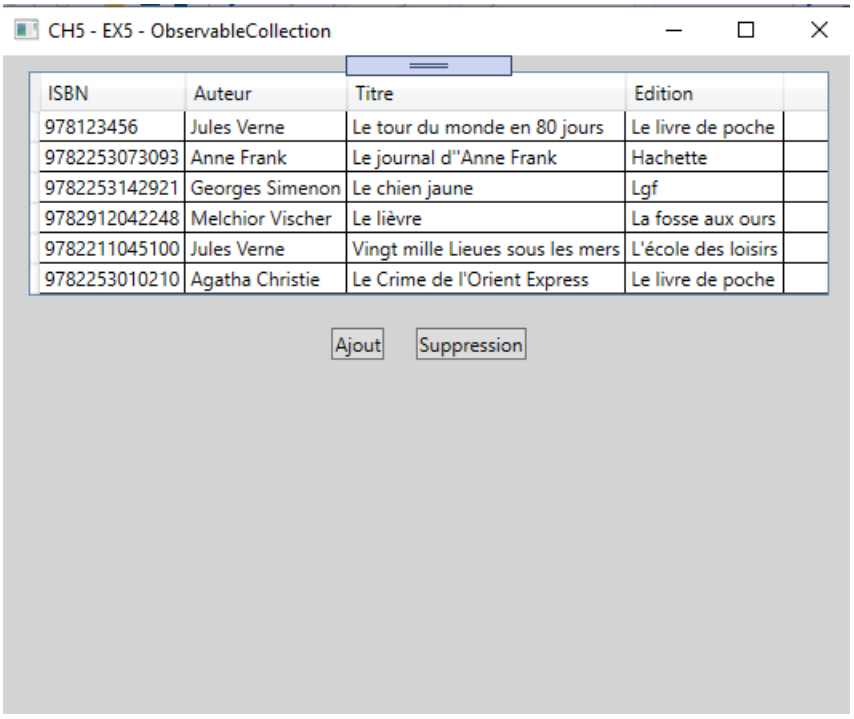
```
using System.Windows;

namespace CH5_EX5_COLLECTION
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new Vue_modele();
        }

        private void Suppression_Click(object sender, RoutedEventArgs e)
        {
            Vue_modele vm = (Vue_modele)this.DataContext;
            int index = MaGrille.SelectedIndex;
            if (index < 0)
            {
                MessageBox.Show("Sélectionnez une ligne à supprimer.");
                return;
            }

            vm.Supprimer(MaGrille.SelectedIndex);
        }

        private void Ajout_Click(object sender, RoutedEventArgs e)
        {
            Vue_modele vm = (Vue_modele)this.DataContext;
            vm.Ajouter();
        }
    }
}
```

Vue-modele.cs

```
using System.Collections.ObjectModel;
using System.ComponentModel;

namespace CH5_EX5_COLLECTION
{


    public class Livre : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        public Livre(string i, string a, string t, string e)
        {
            this.ISBN = i;
            this.Auteur = a;
            this.Titre = t;
            this.Edition = e;
        }

        string isbn;
        public string ISBN
        {
            get { return this.isbn; }
            set
            {
                this.isbn = value;
                OnPropertyChanged("ISBN");
            }
        }

        string auteur;
        public string Auteur
        {
            get { return this.auteur; }
            set { this.auteur = value; OnPropertyChanged("Auteur"); }
        }


        string titre;
        public string Titre
        {
            get { return this.titre; }
            set { this.titre = value; OnPropertyChanged("Titre"); }
        }

        string edition;
        public string Edition
        {
            get { return this.edition; }
            set { this.edition = value; OnPropertyChanged("Edition"); }
        }
    }
```

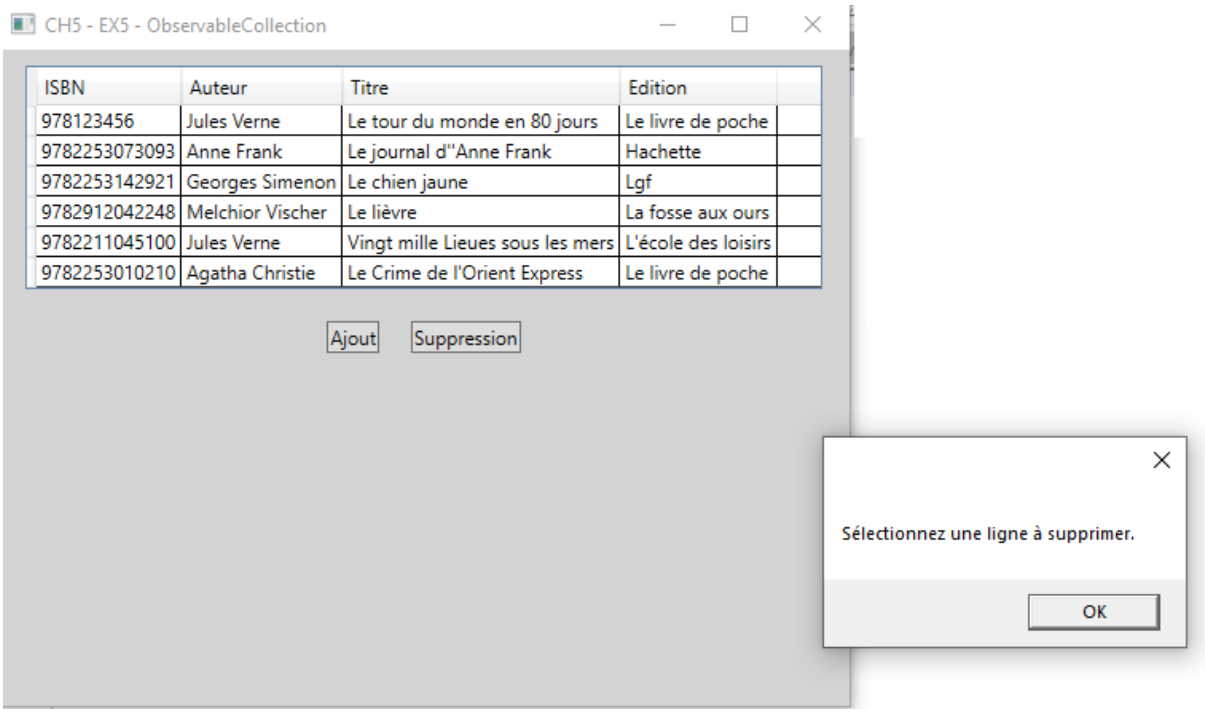
```
class Vue_modele
{
    public ObservableCollection<Livre> Collection { get; set; }

    public Vue_modele()
    {
        Collection = new ObservableCollection<Livre>();
        Collection.Add(new Livre("978123456", "Jules Verne",
            "Le tour du monde en 80 jours", "Le livre de poche"));
        Collection.Add(new Livre("9782253073093", "Anne Frank",
            "Le journal d''Anne Frank", "Hachette"));
        Collection.Add(new Livre("9782253142921", "Georges Simenon",
            "Le chien jaune",  "Lgf"));
        Collection.Add(new Livre("9782912042248", "Melchior Vischer",
            "Le lièvre",  "La fosse aux ours"));
        Collection.Add(new Livre("9782211045100", "Jules Verne",
            "Vingt mille Lieues sous les mers",
            "L'école des loisirs"));
        Collection.Add(new Livre("9782253010210", "Agatha Christie",
            "Le Crime de l'Orient Express",  "Le livre de poche"));
    }

    public void Supprimer(int index)
    {
        Collection.RemoveAt(index);
    }

    public void Ajouter()
    {
        Collection.Add(new Livre("isbn (valeur par défaut)",
            "auteur (valeur par défaut)", "titre (valeur par défaut)",
            "edition (valeur par défaut)"));
    }
}
```

Résultat

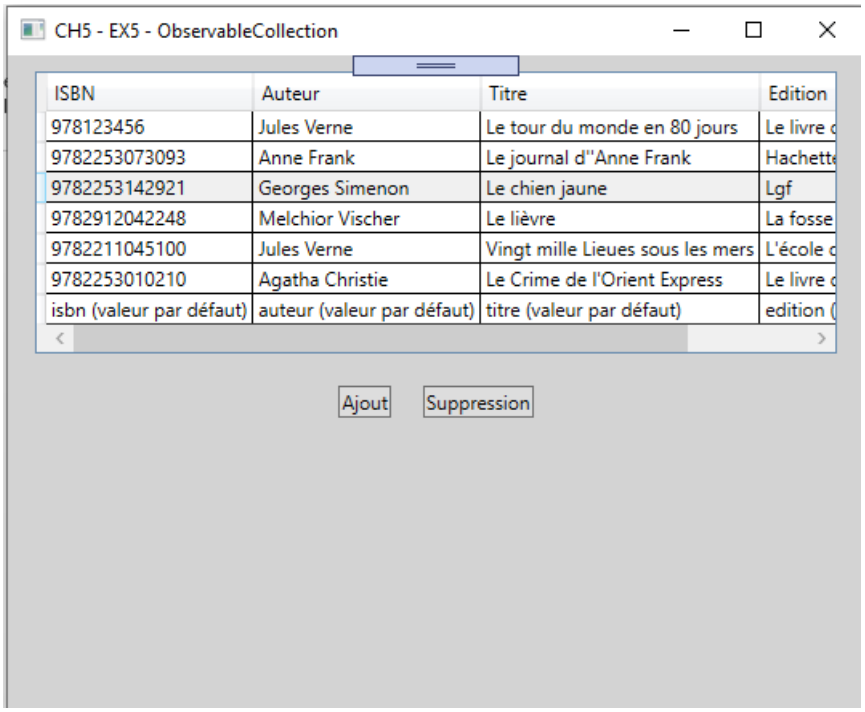


- Au lancement de l'application

- En cliquant sur suppression



- En cliquant sur ajout

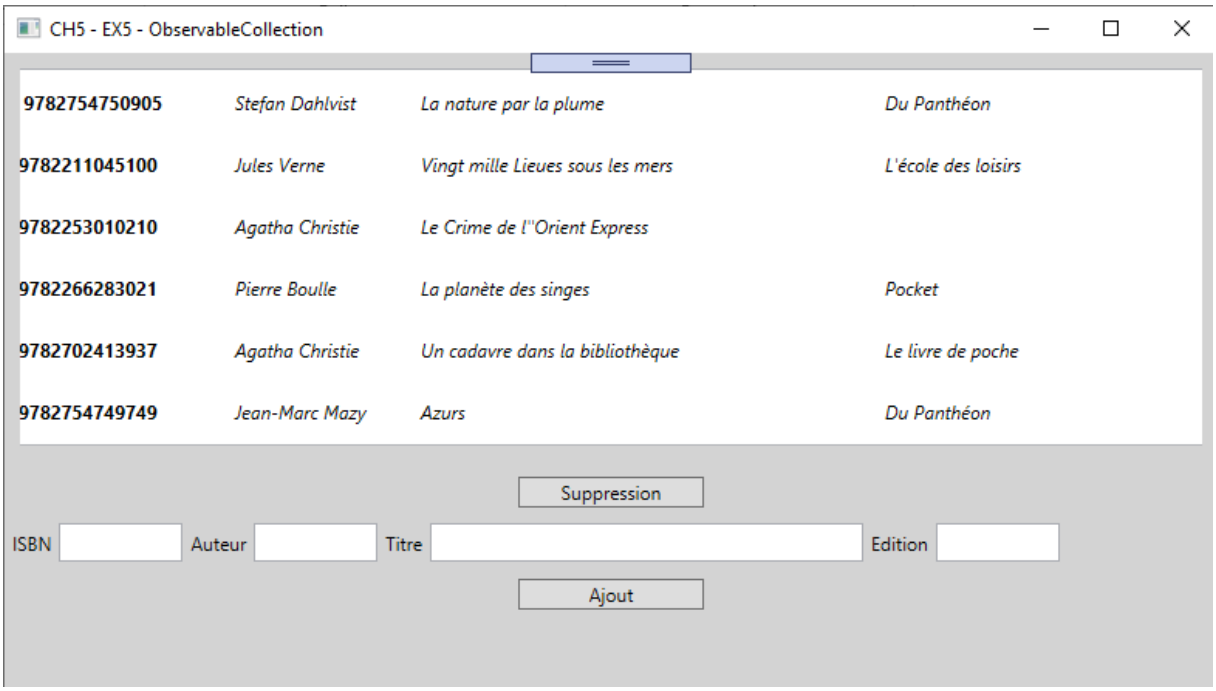


Binding avec DataView

Exemple : binding avec un **DataView** construit depuis une **DataTable** vers un contrôle de type **ListBox**. L'élément sélectionné fera l'objet d'un binding avec un **DataRowView**. Il sera possible de supprimer et d'ajouter un élément dans la liste.

Résultat :

Binding avec DataView



MainWindow.xaml

```
<Window x:Class="CH5_EX5_COLLECTION.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX5_COLLECTION"
        mc:Ignorable="d"
        Background="LightGray"
        Title="CH5 - EX5 - ObservableCollection" Height="450" Width="800">
    <StackPanel VerticalAlignment="Top">

        <ListBox Width="800"
                 Margin="10"
                 Name="MaListe"
                  HorizontalAlignment="Center"
                  ItemsSource="{Binding Collection}"
                  SelectedItem="{Binding Selection}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock FontWeight="Bold" Text="{Binding ISBN}" Margin="10" Width="120" />
                        <TextBlock FontStyle="Italic" Text="{Binding Auteur}" Margin="10" Width="100" />
                        <TextBlock FontStyle="Italic" Text="{Binding Titre}" Margin="10" Width="280" />
                        <TextBlock FontStyle="Italic" Text="{Binding Edition}" Margin="10" Width="100" />
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    <StackPanel>
```

```
        <Button Margin="10" Width="120" Content="Suppression" Click="Suppression_Click" />

        <StackPanel Orientation="Horizontal">
            <TextBlock Text="ISBN" Margin="5" />
            <TextBox Width="80" Height="25" Name="AjoutISBN"/>
            <TextBlock Text="Auteur" Margin="5" />
            <TextBox Width="80" Height="25" Name="AjoutAuteur"/>
            <TextBlock Text="Titre" Margin="5" />
            <TextBox Width="280" Height="25" Name="AjoutTitre"/>
            <TextBlock Text="Edition" Margin="5" />
            <TextBox Width="80" Height="25" Name="AjoutEdition"/>
        </StackPanel>

        <Button Margin="10" Width="120" Content="Ajout" Click="Ajout_Click" />

    </StackPanel>
</StackPanel>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;

namespace CH5_EX5_COLLECTION
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new Vue_modele();
        }

        private void Suppression_Click(object sender, RoutedEventArgs e)
        {
            Vue_modele vm = (Vue_modele)this.DataContext;
            vm.Suppression();
        }

        private void Ajout_Click(object sender, RoutedEventArgs e)
        {
            Vue_modele vm = (Vue_modele)this.DataContext;
            vm.Ajout(AjoutISBN.Text, AjoutAuteur.Text, AjoutTitre.Text, AjoutEdition.Text);
        }
    }
}
```

Vue-modele.cs

```
using System.ComponentModel;
using System.Data;

namespace CH5_EX5_COLLECTION
{
    class Vue_modele : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        DataView dv;
        public DataView Collection
        {
            get { return dv; }
            set { dv = value; OnPropertyChanged("Collection"); }
        }

        DataRowView drv;
        public DataRowView Selection
        {
            get { return drv; }
            set { drv = value; OnPropertyChanged("Selection"); }
        }
```

```
public Vue_modele()
{
    DataTable dt = new DataTable();
    dt.Columns.Add("ISBN");
    dt.Columns.Add("Auteur");
    dt.Columns.Add("Titre");
    dt.Columns.Add("Edition");

    DataRow dr1 = dt.NewRow();
    dr1["ISBN"] = "9782253010210";
    dr1["Auteur"] = "Agatha Christie";
    dr1["Titre"] = "Le Crime de l''Orient Express";
    dr1["Edition"] = "";
    dt.Rows.Add(dr1);

    DataRow dr2 = dt.NewRow();
    dr2["ISBN"] = "9782702413937";
    dr2["Auteur"] = "Agatha Christie";
    dr2["Titre"] = "Un cadavre dans la bibliothèque";
    dr2["Edition"] = "Le livre de poche";
    dt.Rows.Add(dr2);

    DataRow dr3 = dt.NewRow();
    dr3["ISBN"] = " 9782754750905";
    dr3["Auteur"] = "Stefan Dahlvist";
    dr3["Titre"] = "La nature par la plume";
    dr3["Edition"] = "Du Panthéon";
    dt.Rows.Add(dr3);

    DataRow dr4 = dt.NewRow();
    dr4["ISBN"] = "9782754749749";
    dr4["Auteur"] = "Jean-Marc Mazy";
    dr4["Titre"] = "Azurs";
    dr4["Edition"] = "Du Panthéon";
    dt.Rows.Add(dr4);
```

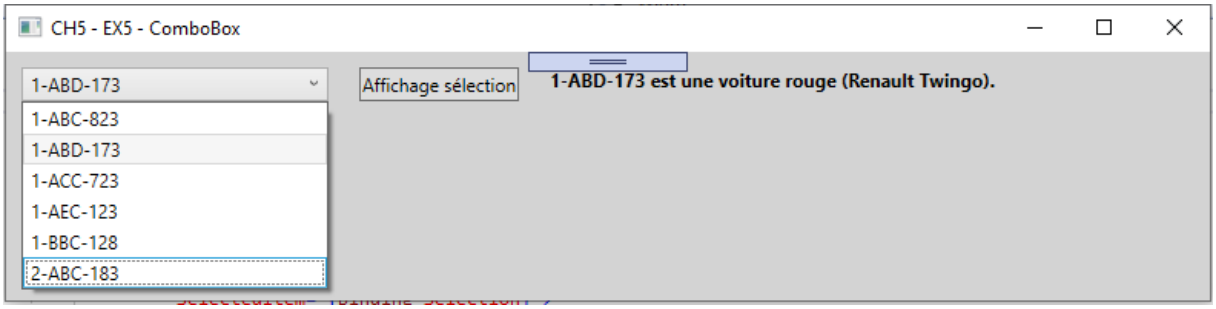
```
    DataRow dr5 = dt.NewRow();
    dr5["ISBN"] = "9782266283021";
    dr5["Auteur"] = "Pierre Boulle";
    dr5["Titre"] = "La planète des singes";
    dr5["Edition"] = "Pocket";
    dt.Rows.Add(dr5);

    DataRow dr6 = dt.NewRow();
    dr6["ISBN"] = "9782211045100";
    dr6["Auteur"] = "Jules Verne";
    dr6["Titre"] = "Vingt mille Lieues sous les mers";
    dr6["Edition"] = "L'école des loisirs";
    dt.Rows.Add(dr6);



    DataView dv = new DataView(dt);
    dv.Sort = "ISBN ASC";
    Collection = dv;
}
```

```
        public void Suppression()
        {
            if (Selection != null)
            {
                Collection.Table.Rows.Remove(Selection.Row);
            }
        }

        public void Ajout(string i, string a, string t, string e)
        {
            DataRow dr = Collection.Table.NewRow();
            dr["ISBN"] = i;
            dr["Auteur"] = a;
            dr["Titre"] = t;
            dr["Edition"] = e;
            Collection.Table.Rows.Add(dr);
        }
    }
}
```

Binding de collection et ComboBox

La spécificité de la **ComboBox** réside dans le fait qu'il faut choisir ce qui est affiché. Ceci est définit avec la propriété **DisplayMemberPath**.

Exemple :



MainWindow.xaml

```
<Window x:Class="CH5_EX6_COMBOBOX.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX6_COMBOBOX"
        mc:Ignorable="d"
        Background="LightGray"
        Title="CH5 - EX5 - ComboBox" Height="200" Width="800">
    <WrapPanel VerticalAlignment="Top" HorizontalAlignment="Left"  Orientation="Horizontal">

        <ComboBox Margin="10"
                  Width="200"
                  HorizontalAlignment="Center"
                  ItemsSource="{Binding Collection}"
                  DisplayMemberPath="ISBN"
                  SelectedItem="{Binding Selection}">
        </ComboBox>

        <Button Margin="10" Content="Affichage sélection" Click="Affichage_Click" />

        <TextBlock Margin="10" Text="{Binding SelectionString}" FontWeight="Bold" />

    </WrapPanel>
</Window>
```

MainWindow.xaml.cs

```
using System.Windows;

namespace CH5_EX6_COMBOBOX
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new Vue_modele();
        }

        private void Affichage_Click(object sender, RoutedEventArgs e)
        {
            Vue_modele vm = (Vue_modele)this.DataContext;
            vm.Affichage();
        }
    }
}
```

VueModele.cs

```
using System.ComponentModel;
using System.Data;
using System.Windows;

namespace CH5_EX6_COMBOBOX
{
    class Vue_modele : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propertyName)
        {
            if (this.PropertyChanged != null)
            {
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        DataView dv;
        public DataView Collection
        {
            get { return dv; }
            set { dv = value; OnPropertyChanged("Collection"); }
        }

        DataRowView drv;
        public DataRowView Selection
        {
            get { return drv; }
            set { drv = value; OnPropertyChanged("Selection"); }
        }

        string selectionString;
        public string SelectionString
        {
            get { return selectionString; }
            set { selectionString = value; OnPropertyChanged("SelectionString"); }
        }
```

```
public Vue_modele()
{
    DataTable dt = new DataTable();

    dt.Columns.Add("ISBN");
    dt.Columns.Add("Auteur");
    dt.Columns.Add("Titre");
    dt.Columns.Add("Edition");

    DataRow dr1 = dt.NewRow();
    dr1["ISBN"] = "9782253010210";
    dr1["Auteur"] = "Agatha Christie";
    dr1["Titre"] = "Le Crime de l'Orient Express";
    dr1["Edition"] = "";
    dt.Rows.Add(dr1);

    DataRow dr2 = dt.NewRow();
    dr2["ISBN"] = "9782702413937";
    dr2["Auteur"] = "Agatha Christie";
    dr2["Titre"] = "Un cadavre dans la bibliothèque";
    dr2["Edition"] = "Le livre de poche";
    dt.Rows.Add(dr2);

    DataRow dr3 = dt.NewRow();
    dr3["ISBN"] = " 9782754750905";
    dr3["Auteur"] = "Stefan Dahlvist";
    dr3["Titre"] = "La nature par la plume";
    dr3["Edition"] = "Du Panthéon";
    dt.Rows.Add(dr3);

    DataRow dr4 = dt.NewRow();
    dr4["ISBN"] = "9782754749749";
    dr4["Auteur"] = "Jean-Marc Mazy";
    dr4["Titre"] = "Azurs";
    dr4["Edition"] = "Du Panthéon";
    dt.Rows.Add(dr4);
```

```
    DataRow dr5 = dt.NewRow();
    dr5["ISBN"] = "9782266283021";
    dr5["Auteur"] = "Pierre Boulle";
    dr5["Titre"] = "La planète des singes";
    dr5["Edition"] = "Pocket";
    dt.Rows.Add(dr5);

    DataRow dr6 = dt.NewRow();
    dr6["ISBN"] = "9782211045100";
    dr6["Auteur"] = "Jules Verne";
    dr6["Titre"] = "Vingt mille Lieues sous les mers";
    dr6["Edition"] = "L'école des loisirs";
    dt.Rows.Add(dr6);

    DataView dv = new DataView(dt);
    dv.Sort = "ISBN ASC";
    Collection = dv;
}
```

```
public void Affichage()
{
    SelectionString = Selection["ISBN"].ToString() +
        " est un livre de " + Selection["Auteur"].ToString() +
        " (" + Selection["Titre"].ToString() + " " +
        Selection["Edition"].ToString() + ").";
}
}
```