

WPF

Programmation événementielle client lourd

Sommaire

- **Présentation**
- **Utilisation des contrôles**
- **Les principaux contrôles**
- **Interactions clavier et souris**
- **Introduction au Patron MVVM**

Présentation

- ➊ Développement d'une application graphique sous Windows.
- ➋ Le framework .NET propose depuis ses débuts la technologie **Windows Forms** vieillissante (en phase de maintenance depuis 2014 ... → plus de mise à jour majeure)
- ➌ Depuis 2006, une nouvelle technologie est apparue :
WPF (Windows Presentation Foundation)

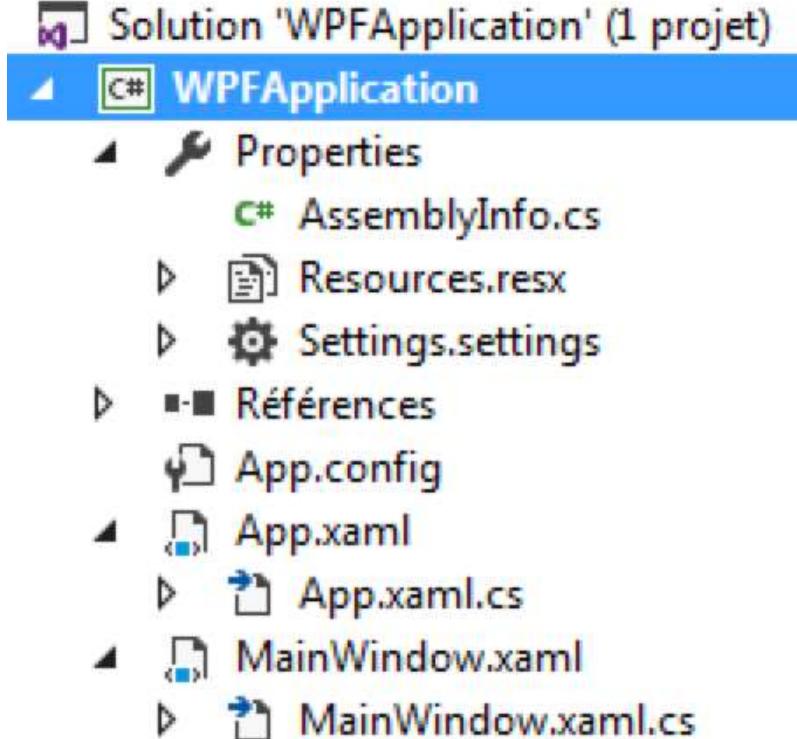
WPF



Apporte de nombreuses améliorations :

-  **Définition de l'interface à l'aide d'un langage de balisage **XAML****
-  **Les composants WPF utilise le dessin vectoriel**
-  **L'affichage s'appuie sur DirectX (GPU)**
-  **Découpage entre l'interface et le code métier (notion de **binding**) → testabilité et maintenabilité meilleurs**
-  **Création simple d'applications fluides à l'aspect moderne.**

Structure d'une application WPF



Nouvelle Application WPF

App.xaml et App.xaml.cs

Déclaration et implémentation (**code-behind**) de la classe principale

MainWindow.xaml et MainWindow.xaml.cs

Déclaration et implémentation (**code-behind**) de la fenêtre principale

App.config

Paramètres de configuration de l'application

AssemblyInfo.cs

Métadonnées : titre, version, copyright ...

Resources.resx

Chaines de caractères utilisables en plusieurs endroits de l'application

Settings.settings

Paramètres liés à l'application : polices, couleurs ...

XAML

- Utilisé pour créer l'interface graphique.
- Dialecte XML à la manière de HTML
- Imbrication d'objets pour former un arbre logique
- Rq :
Il est important de noter que tout ce qui peut être codé en XAML peut être codé en C#

XAML

The screenshot shows the Visual Studio IDE interface with the XAML editor open. The title bar indicates the project is named "App" and the file is "MainWindow.xaml". The XAML code defines a window with a grid containing a button labeled "Clic ici".

```
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="525">
<Grid>
    <Button
        x:Name="button" Content="Clic ici"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        Height="121" Margin="50,49,0,0" Width="260" FontSize="50"/>
</Grid>

```

Concept de binding

- À maîtriser pour une utilisation optimale des capacités de WPF.
- Liaison **unidirectionnelle** ou **bidirectionnelle** entre un élément XAML et un **contexte de données** (**DataContext**) et permet la propagation des données du code C# vers le code XAML (ou inversement) sans être obligé de faire du code C#.
- Mécanisme qui est la clé du **découpage** entre le code métier et l'interface graphique.
- On en reparle dans le prochain cours

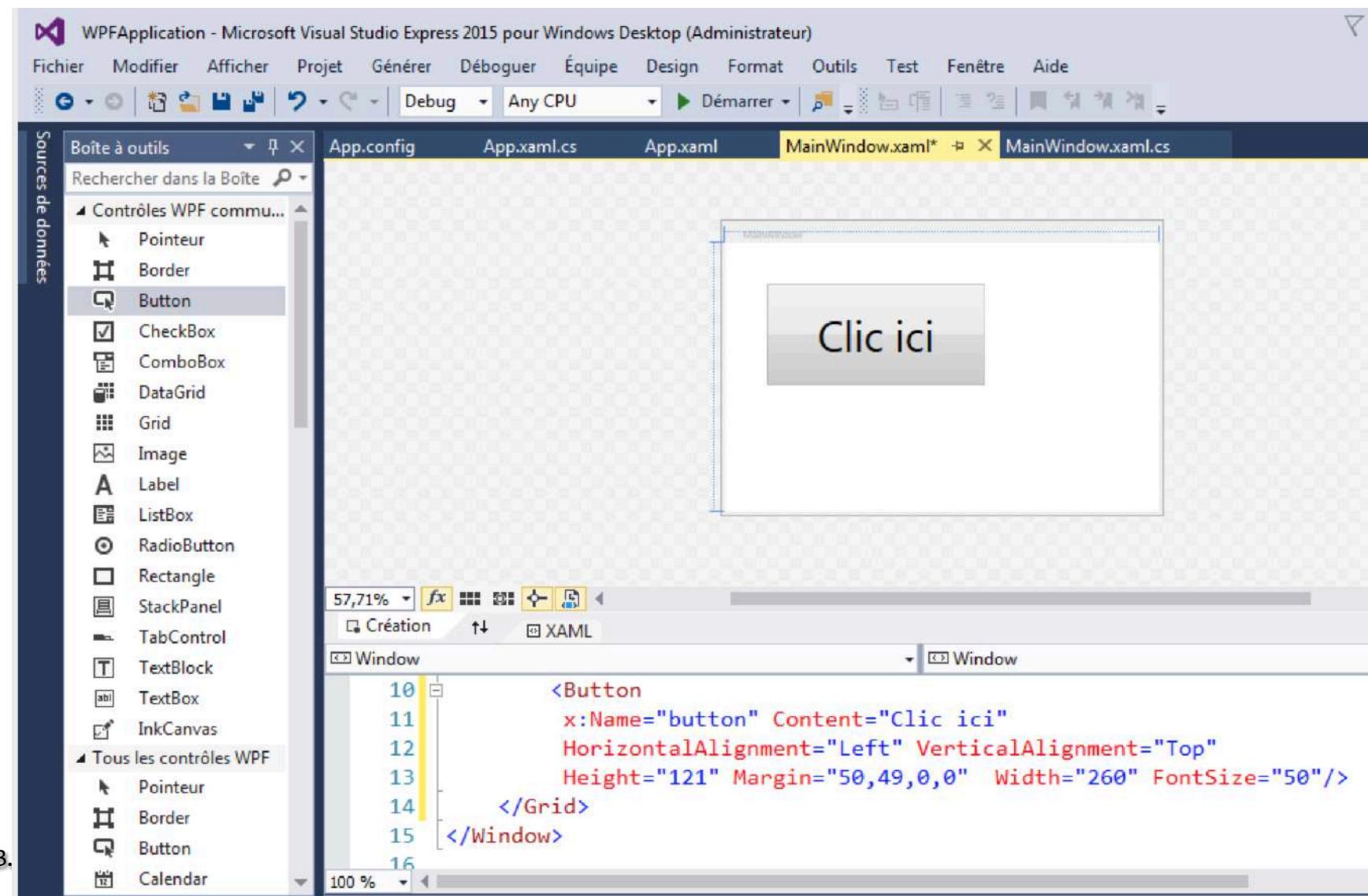
Utilisation des composants

- L'interface graphique est composée de contrôles définis par du code XAML (et plus rarement du C#)
- Ces contrôles font tous partie d'une hiérarchie leur permettant d'avoir des propriétés communes, la plupart du temps liées au positionnement.
- Utilisation d'un **concepteur visuel** qui génère le code XAML modifiable manuellement (contrairement à Windows Forms)

Ajout de contrôles



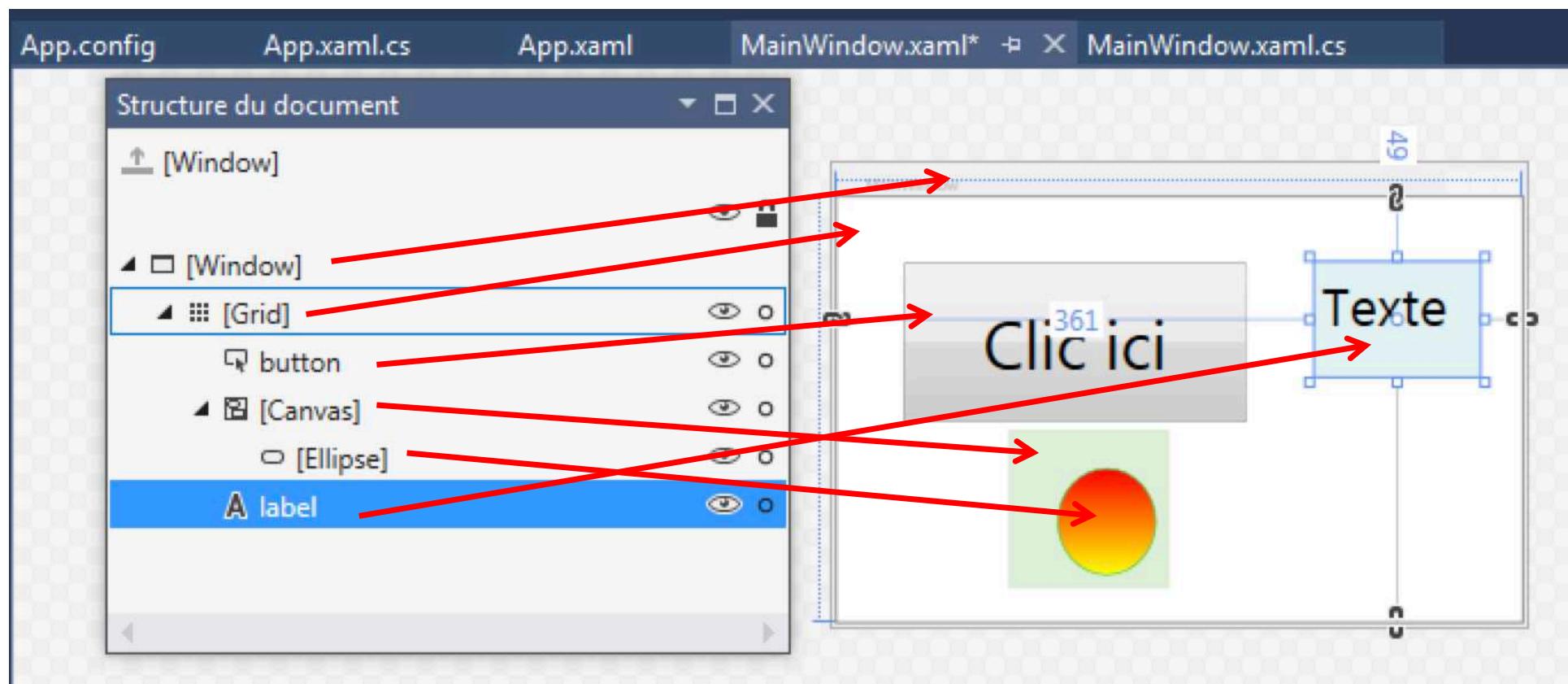
Une boîte à outils permet le placement à l'aide la souris :



Ajout de contrôles



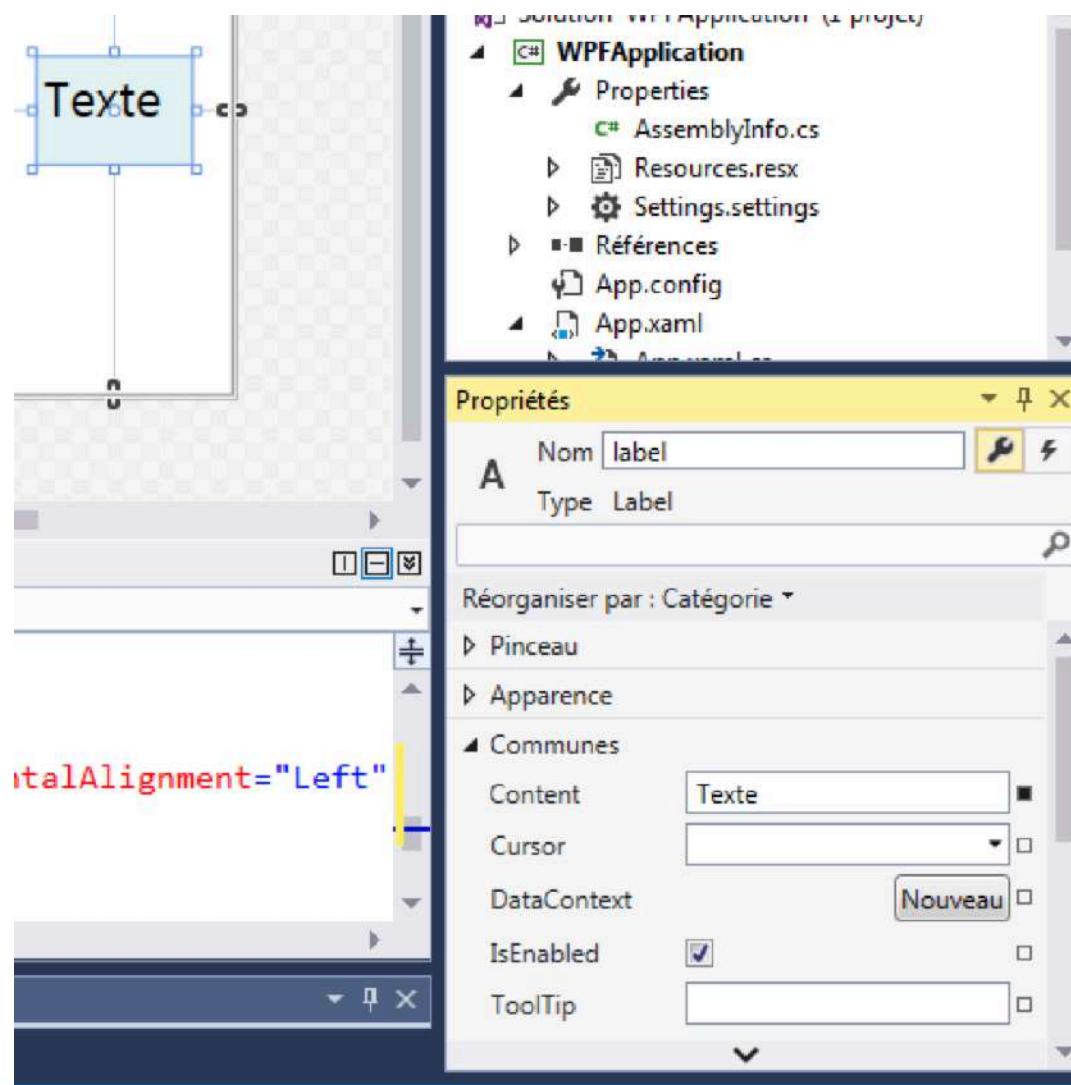
La fenêtre structure du document permet de visualiser l'arborescence de la fenêtre en cours d'édition



Ajout de contrôles



La fenêtre propriétés permet d'accéder aux informations utiles :



Ajout de contrôles



Positionnement et dimensionnement des contrôles :

- Par défaut, les contrôles sont positionnés aux coordonnées (0,0) de leur conteneur.**
- On peut modifier :**
 - Height et Width**
 - HorizontalAlignment VerticalAlignment**
 - Margin**
 - Padding**
- Positionnement dans un canvas (conteneur d'éléments graphique en absolu) :**
 - Canvas.Top**
 - Canvas.Left**
 - Canvas.Right**
 - Canvas.Bottom**

Ajout de contrôles

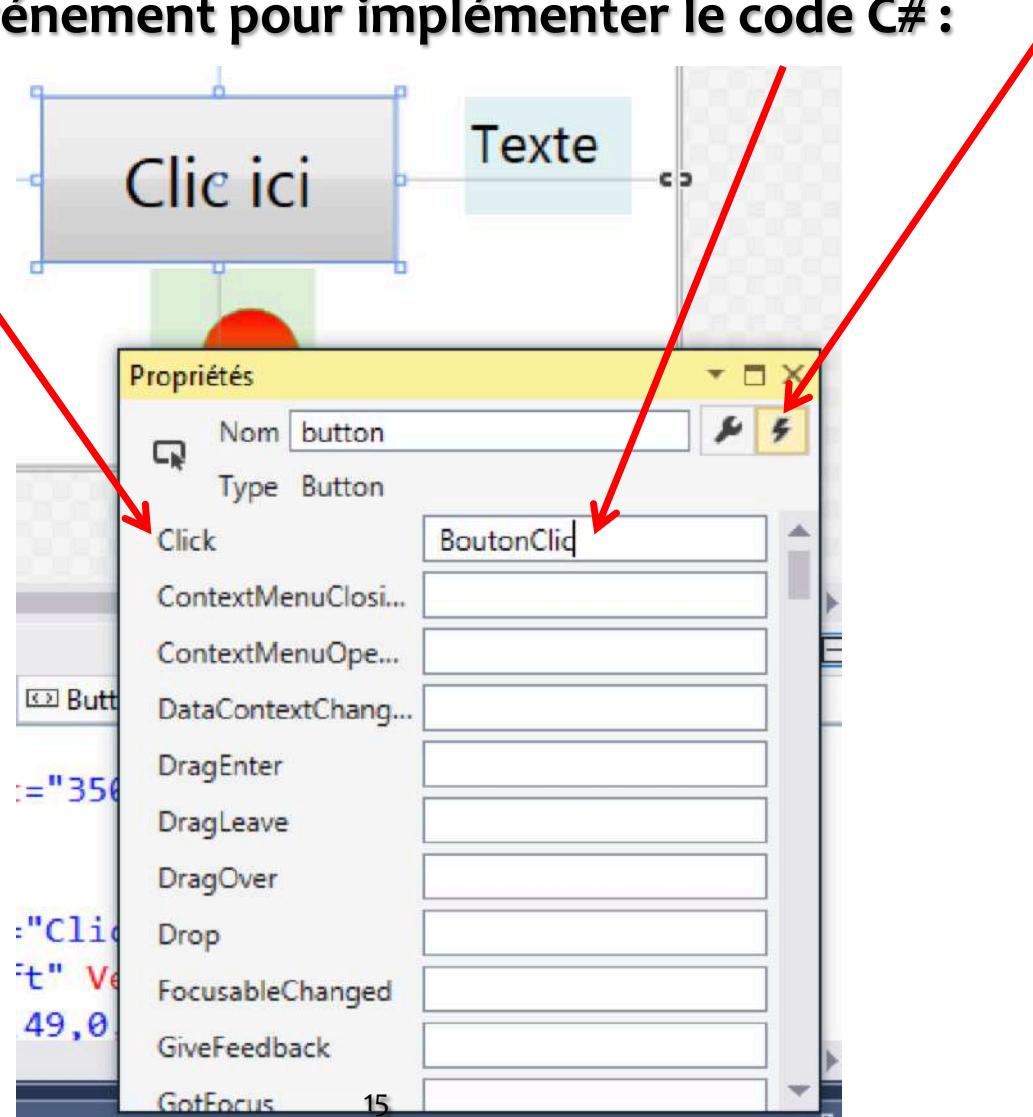


Propriétés supplémentaires :

- **BorderBrush** : couleur de la bordure du contrôle
- **BorderThickness** : épaisseur de trait de la bordure du contrôle
- **Visibility** : état de visibilité du contrôle
 - **Visible**
 - **Hidden (masqué mais zone occupée indisponible)**
 - **Collapsed (masqué mais zone occupée libérée)**

Ajout d'un gestionnaire d'événement à un contrôle

- Dans la fenêtre propriétés on active le gestionnaire d'événements, et on choisit le type d'événement pour implémenter le code C# :



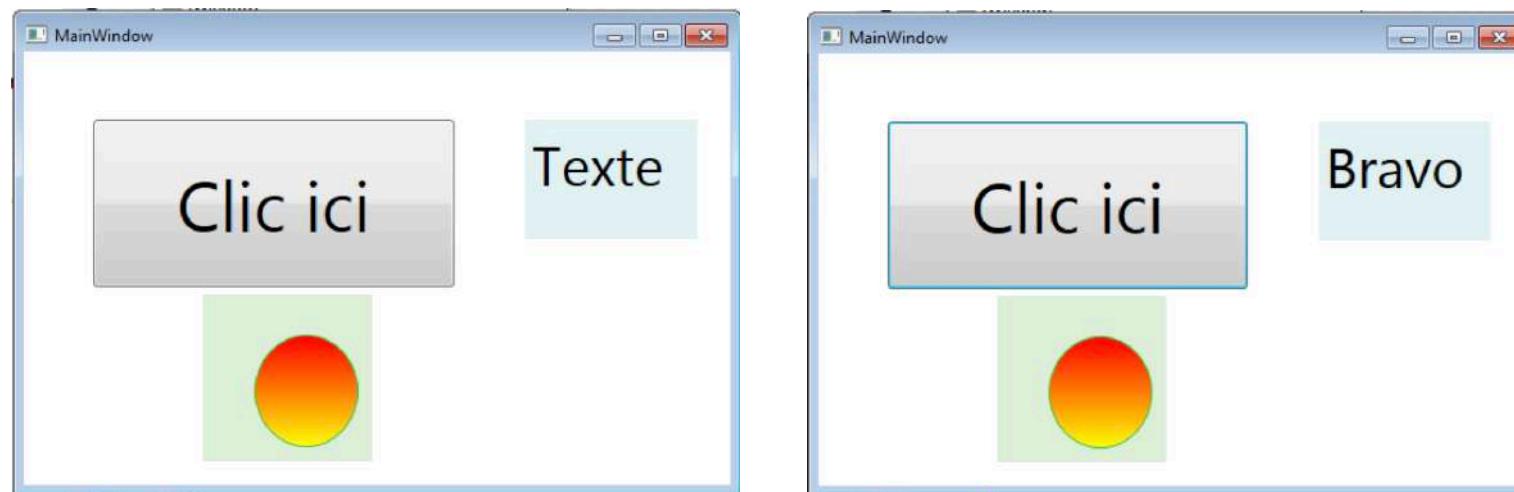
Ajout d'un gestionnaire d'événement à un contrôle



Il ne reste plus qu'à compléter le code généré dans la fenêtre en cours d'édition :

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

    private void BoutonClic(object sender, RoutedEventArgs e)
    {
        label.Content = "Bravo !!";
    }
}
```

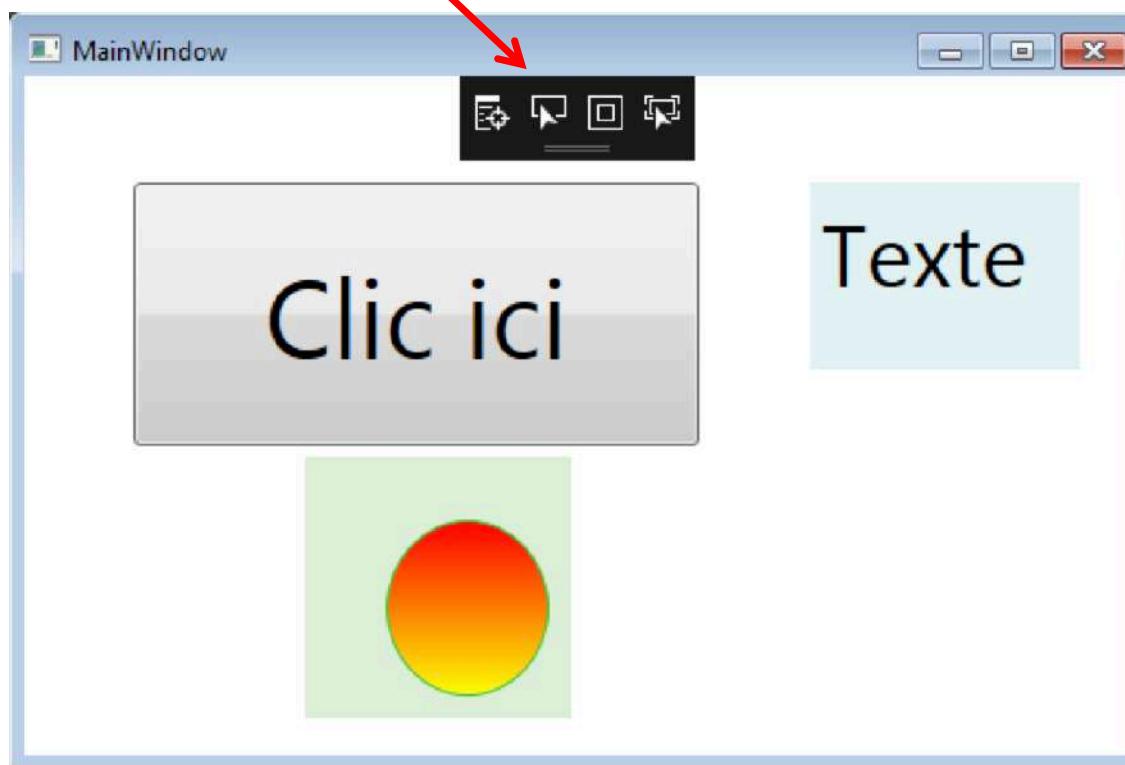


Ajout d'un gestionnaire d'événement à un contrôle



Remarque :

En mode débogage, un affichage supplémentaire apparaît :



Cela permet d'accéder à des informations utiles

Ajout d'un gestionnaire d'événement à un contrôle

- ➊ Si on veut le faire directement dans le code source XAML on ajoute à la déclaration :

```
<Button  
    x:Name="button" Content="Clic ici"  
    HorizontalAlignment="Left" VerticalAlignment="Top"  
    Height="121" Margin="50,49,0,0" Width="260" FontSize="50"  
    Click="BoutonClic"/>
```

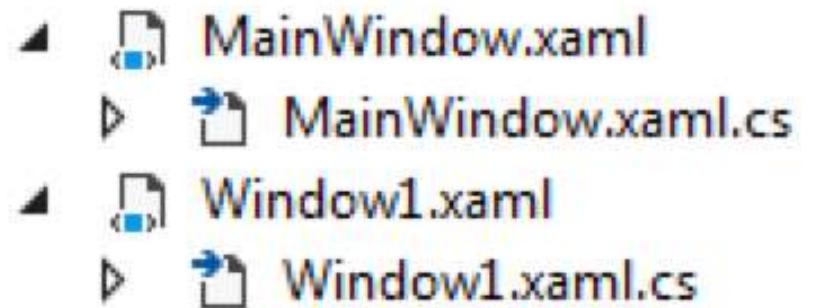
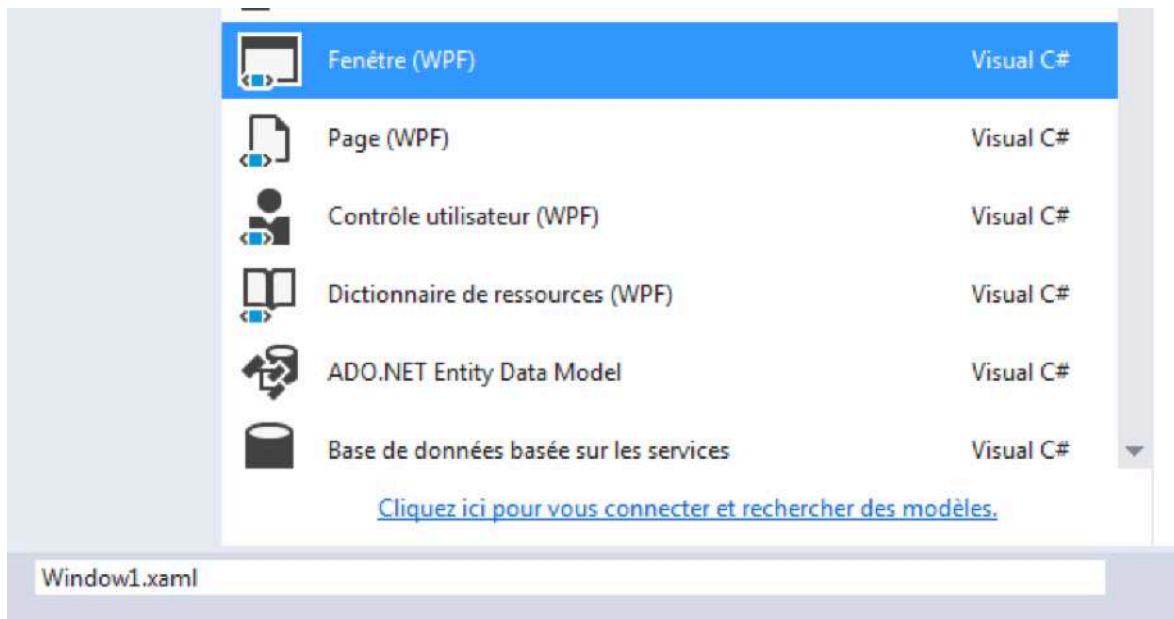
- ➋ Le double clic sur le contrôle génère également un gestionnaire pour l'événement par défaut du contrôle.

Contrôles de fenêtrage

- **Les contrôles de fenêtrage sont essentiels dans une application WPF. Ce sont eux qui sont les conteneurs de l'interface graphique :**
 - **Window : fenêtre d'application → mode multifenêtré**
 - **NavigationWindow : classe dérivée de Window → Fenêtre similaire à un navigateur web**

Window

- ➊ permet de définir une fenêtre d'application
- ➋ Ajouter un nouvel élément :



Window

- Le contrôle possède de nombreuses propriétés permettant de définir son aspect, son comportement ou son emplacement. Les plus importantes :
 - **Title**
 - **ResizeMode** : mode de redimensionnement disponibles pour la fenêtre
 - **ShowInTaskbar** : visible dans la barre des tâches
 - **WindowState** : maximisée, minimisée ou dimensionnée normalement
 - **WindowStartupLocation** : centrée par rapport à l'écran ou la fenêtre parent
 - **Topmost** : en avant-plan même si l'application n'a pas le focus
- Pour ouvrir une nouvelle fenêtre, il faut l'instancier avec sa méthode **Show** ou sa méthode **ShowDialog** si la fenêtre doit être modale :

```
Window1 maFenetre = new Window1();
maFenetre.Show();
```

Window

- Ce contrôle définit aussi plusieurs événements dont **Loaded**, **Closing** et **Closed** :
 - **Loaded** : exécution de portion de code lorsque le contenu de la fenêtre a été chargée mais pas encore affiché.
 - **Closing** et **Closed** sont déclenchés respectivement avant et après la fermeture de la fenêtre.
- La fenêtre de démarrage d'une application est définie dans le fichier **App.xaml** :

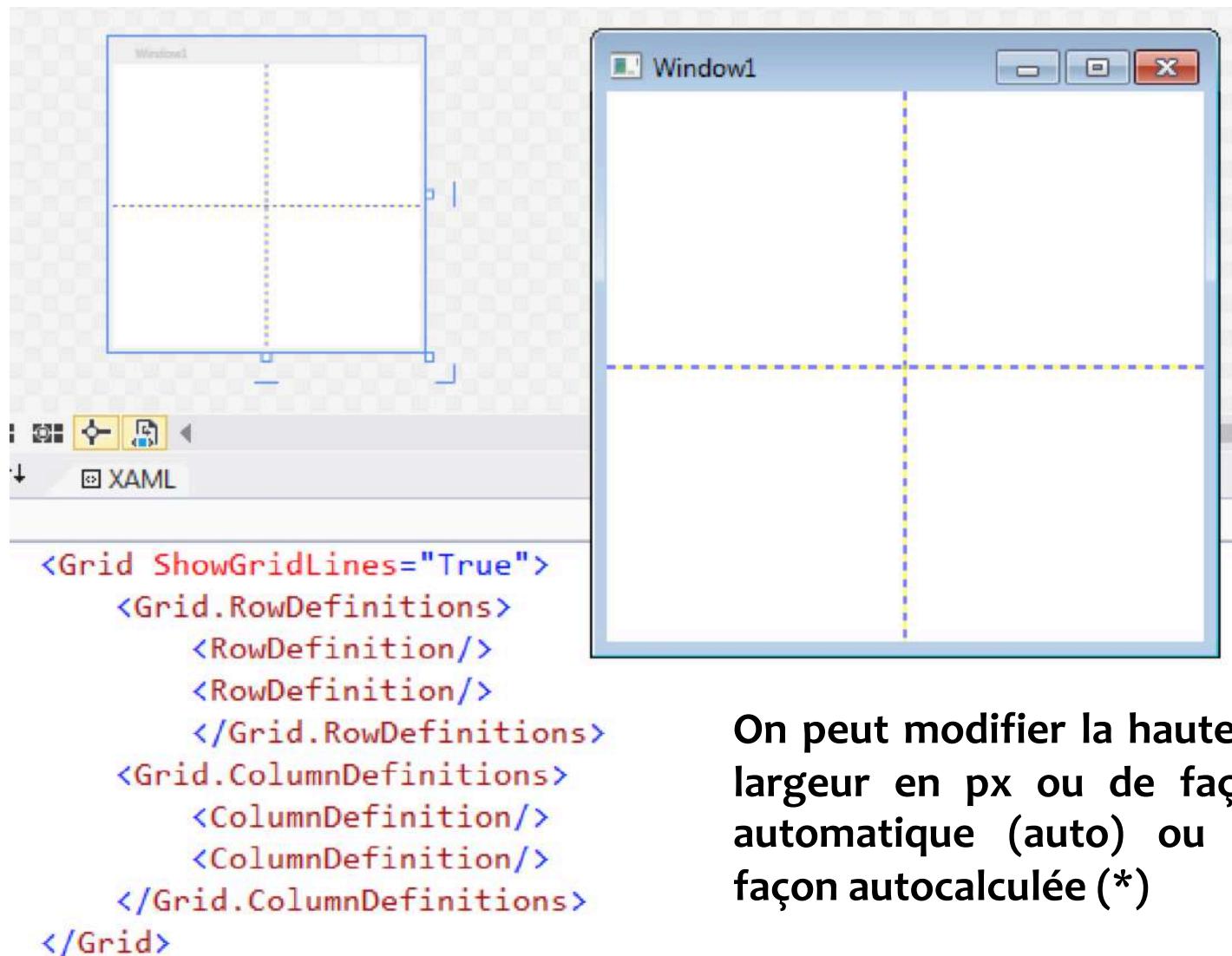
```
<Application x:Class="WPFAplication.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:WPFAplication"
    StartupUri="MainWindow.xaml">
    <Application.Resources>
        </Application.Resources>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

Contrôle de disposition

Conteneurs permettant de définir le positionnement de chacun des contrôles graphiques qu'ils contiennent.

Contrôle de disposition

- Grid : grille pour laquelle on définit des lignes et colonnes
(il est possible d'enlever la transparence)



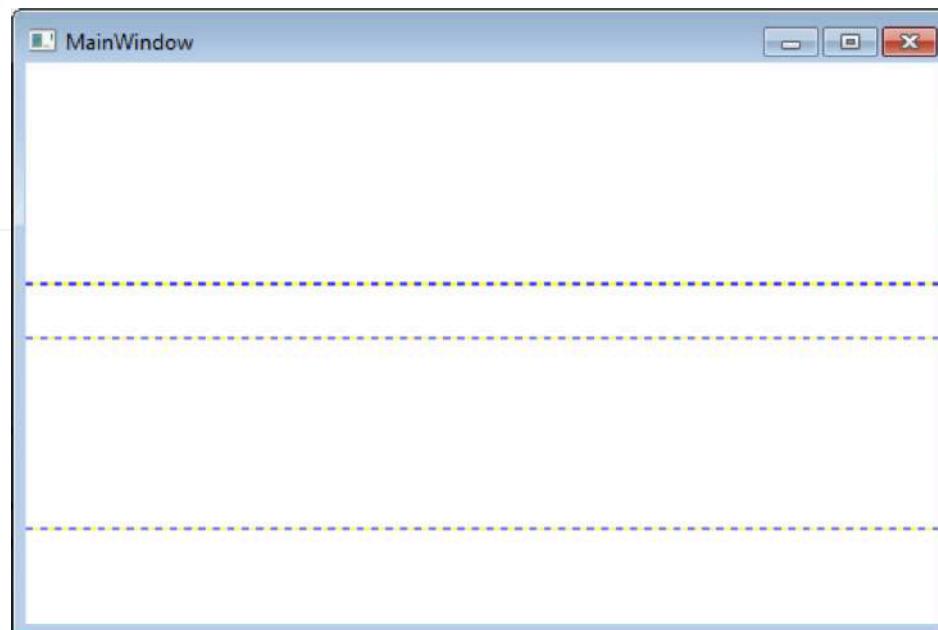
On peut modifier la hauteur, largeur en px ou de façon automatique (auto) ou de façon autocalculée (*)

Contrôle de disposition

- Grid : grille pour laquelle on définit des lignes et colonnes (il est possible d'enlever la transparence)

On peut modifier la hauteur, largeur en px ou de façon automatique (auto) ou de façon autocalculée (*)

```
<Grid ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height="123"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="30"/>
        <RowDefinition Height="2*"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
</Grid>
```



123px

0px

30px

2/3

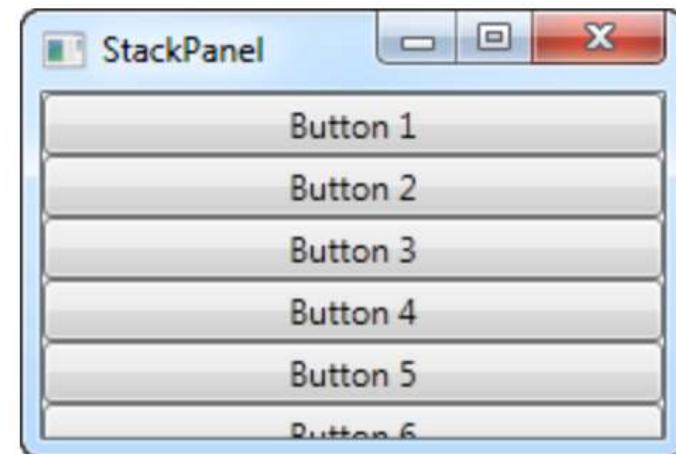
1/3

Contrôle de disposition



StackPanel

```
<StackPanel>
    <Button>Button 1</Button>
    <Button>Button 2</Button>
    <Button>Button 3</Button>
    <Button>Button 4</Button>
    <Button>Button 5</Button>
    <Button>Button 6</Button>
</StackPanel>
```



```
<StackPanel Orientation="Horizontal">
```

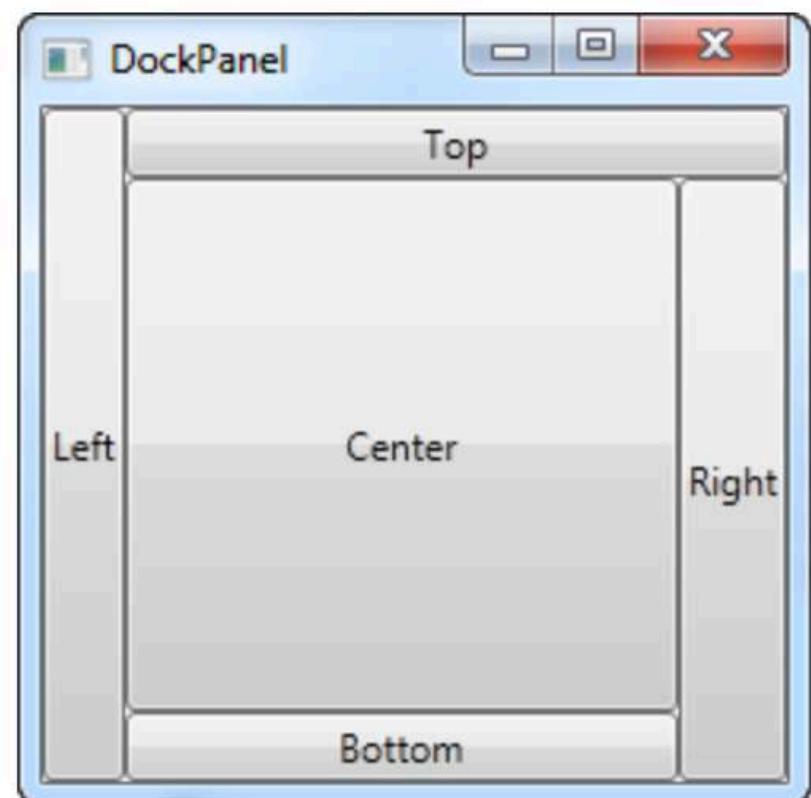


Contrôle de disposition



DockPanel

```
<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>
```



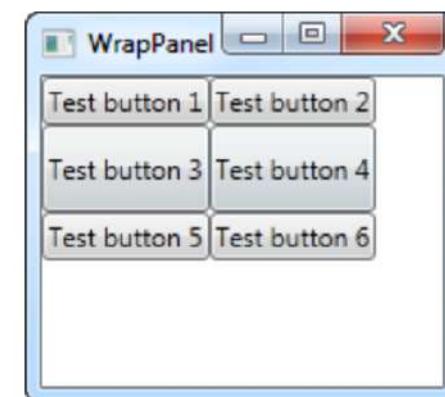
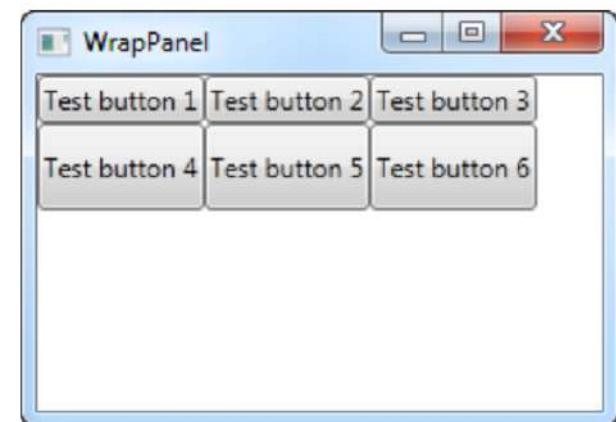
Contrôle de disposition



WrapPanel :

- Icon representing a control or component.**StackPanel amélioré qui renvoie le contenu sur une nouvelle ligne (ou colonne) quand l'espace disponible est insuffisant sur l'alignement en cours de remplissage.**
- Icon representing a control or component.**C'est un composant dynamique qui remplit tout l'espace disponible (y compris lors d'un redimensionnement)**

```
<WrapPanel>
    <Button>Test button 1</Button>
    <Button>Test button 2</Button>
    <Button>Test button 3</Button>
    <Button Height="40">Test button 4</Button>
    <Button>Test button 5</Button>
    <Button>Test button 6</Button>
</WrapPanel>
```

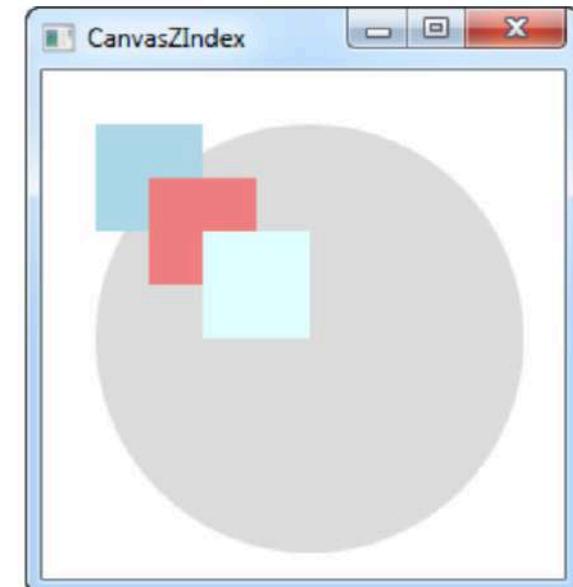


Contrôle de disposition



Canvas : destiné à contenir des contrôles graphiques positionnés de manière absolu

```
<Canvas>
    <Ellipse Fill="Gainsboro" Canvas.Left="25" Canvas.Top="25" Width="200" Height="200" />
    <Rectangle Fill="LightBlue" Canvas.Left="25" Canvas.Top="25" Width="50" Height="50" />
    <Rectangle Fill="LightCoral" Canvas.Left="50" Canvas.Top="50" Width="50" Height="50" />
    <Rectangle Fill="LightCyan" Canvas.Left="75" Canvas.Top="75" Width="50" Height="50" />
</Canvas>
```



Contrôle d'affichage de données

- **Affichage de texte :**
 - **TextBlock (propriété Text)**
 - **Label (propriété Content)**
 - **ScrollViewer (barre de défilement)**
 - **ItemControl (lié à une collection d'objets)**
- **Affichage d'image :**
 - Image (Source)**
- **Barre de statut :**
 - **StatusBar**
- **InfoBulle :**
 - **ToolTip**

Contrôle d'édition de texte

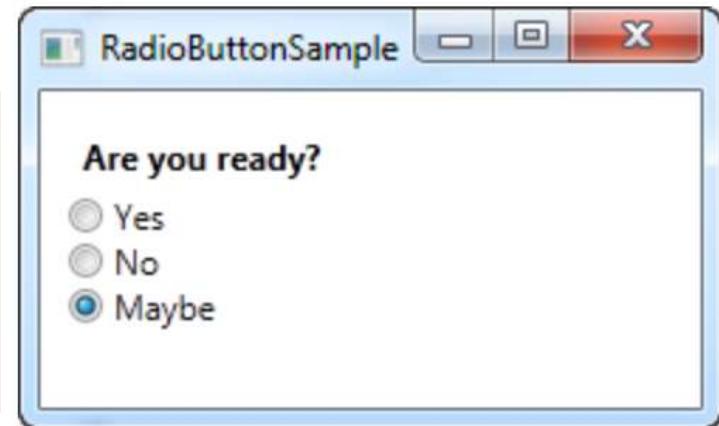
- Le plus simple : TextBox
- Le plus riche : RichTextBox (format RTF avec un copier/coller)
- PasswordBox (caractère de remplacement PasswordChar)

Contrôle de sélection



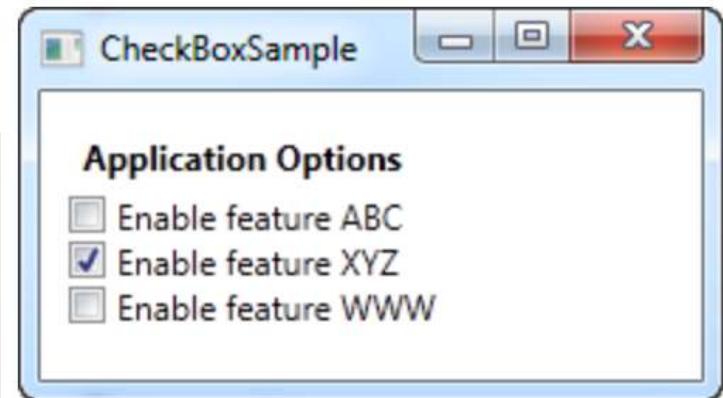
RadioButton :

```
<StackPanel Margin="10">
    <Label FontWeight="Bold">Are you ready?</Label>
    <RadioButton>Yes</RadioButton>
    <RadioButton>No</RadioButton>
    <RadioButton IsChecked="True">Maybe</RadioButton>
</StackPanel>
```



CheckBox :

```
<StackPanel Margin="10">
    <Label FontWeight="Bold">Application Options</Label>
    <CheckBox>Enable feature ABC</CheckBox>
    <CheckBox IsChecked="True">Enable feature XYZ</CheckBox>
    <CheckBox>Enable feature WWW</CheckBox>
</StackPanel>
```



ComboBox, ListBox, ListView, TreeView, Slider, Calender, DatePicker

Contrôle d'action

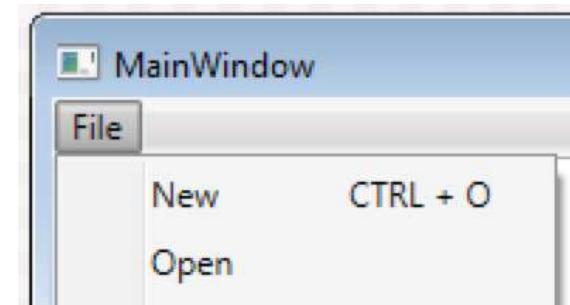
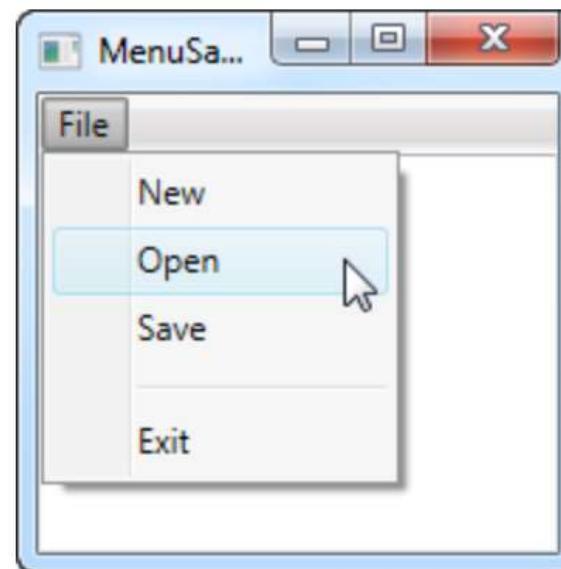
- Button (événement Click)
- Menu (événement Click sur MenuItem) :

```
<DockPanel>
    <Menu DockPanel.Dock="Top">
        <MenuItem Header="_File">
            <MenuItem Header="_New" />
            <MenuItem Header="_Open" />
            <MenuItem Header="_Save" />
            <Separator />
            <MenuItem Header="_Exit" />
        </MenuItem>
    </Menu>
    <TextBox AcceptsReturn="True" />
</DockPanel>
```

« _ » permet d'activer la première lettre

InputGestureText="CTRL + O"

À coder évidemment !!

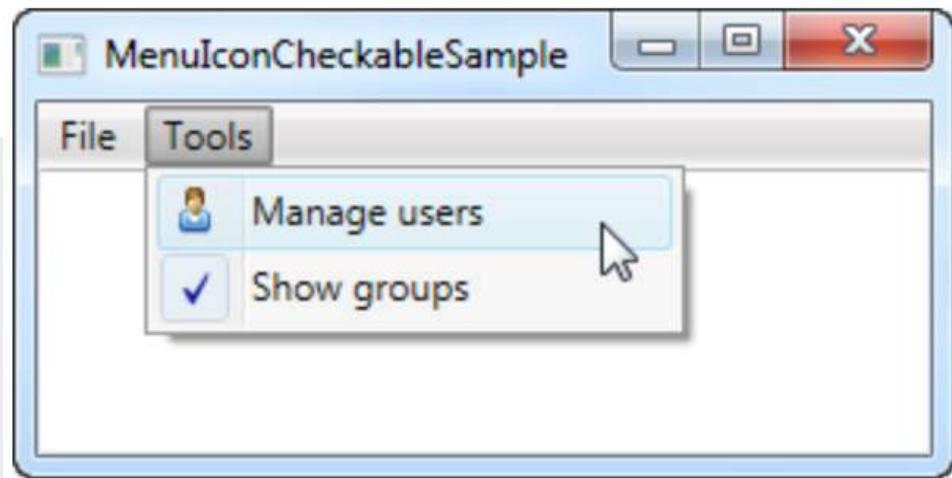


Contrôle d'action



Menu :

```
<DockPanel>
    <Menu DockPanel.Dock="Top">
        <MenuItem Header="_File">
            <MenuItem Header="_Exit" />
        </MenuItem>
        <MenuItem Header="_Tools">
            <MenuItem Header="_Manage users">
                <MenuItem.Icon>
                    <Image Source="/WpfTutorialSamples;component/Images/user.png" />
                </MenuItem.Icon>
            </MenuItem>
            <MenuItem Header="_Show groups" IsCheckable="True" IsChecked="True" />
        </MenuItem>
    </Menu>
    <TextBox AcceptsReturn="True" />
</DockPanel>
```



On peut faire des sous-menus (imbrication de MenuItem) et utiliser des séparateurs (Separator)

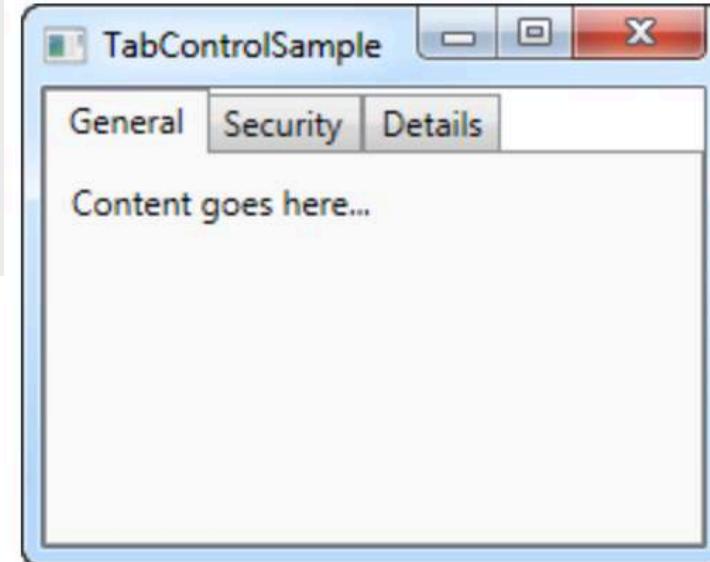
Contrôle d'action

ContextMenu : Menu contextuel

ToolBar : Barre d'outils

TabControl : Onglets

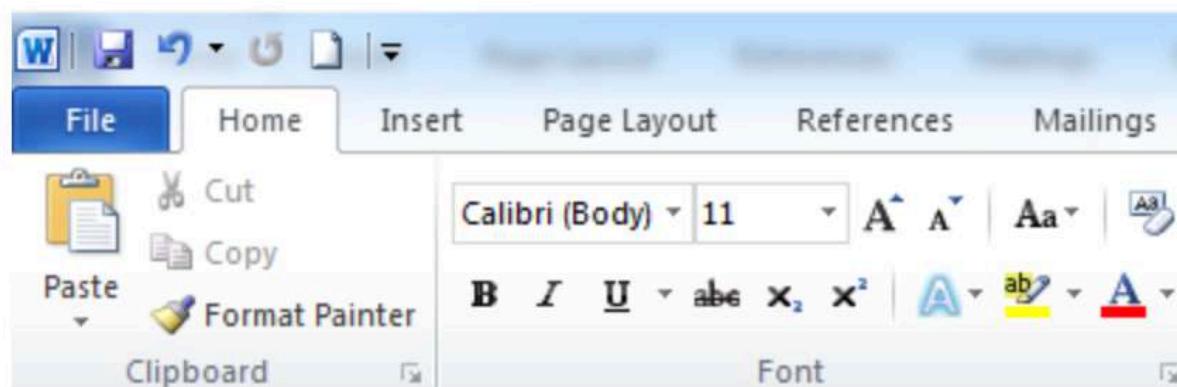
```
<Grid>
    <TabControl>
        <TabItem Header="General">
            <Label Content="Content goes here..." />
        </TabItem>
        <TabItem Header="Security" />
        <TabItem Header="Details" />
    </TabControl>
</Grid>
```



Contrôle d'action



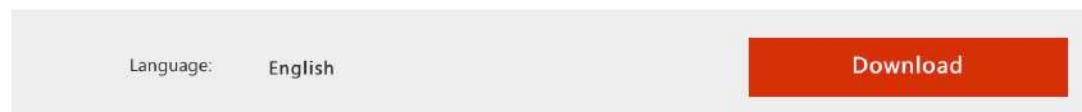
Le ruban :



Contrôle non disponible de base mais téléchargeable :

<https://www.microsoft.com/en-us/download/details.aspx?id=11877>

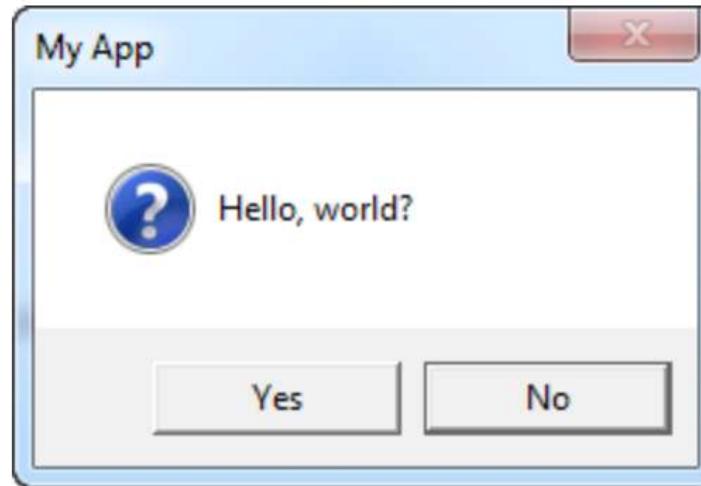
Microsoft Ribbon for WPF October 2010



Contrôle de dialogue



MessageBox :



OpenFileDialog



SaveFileDialog



Personalisée (Window ...):



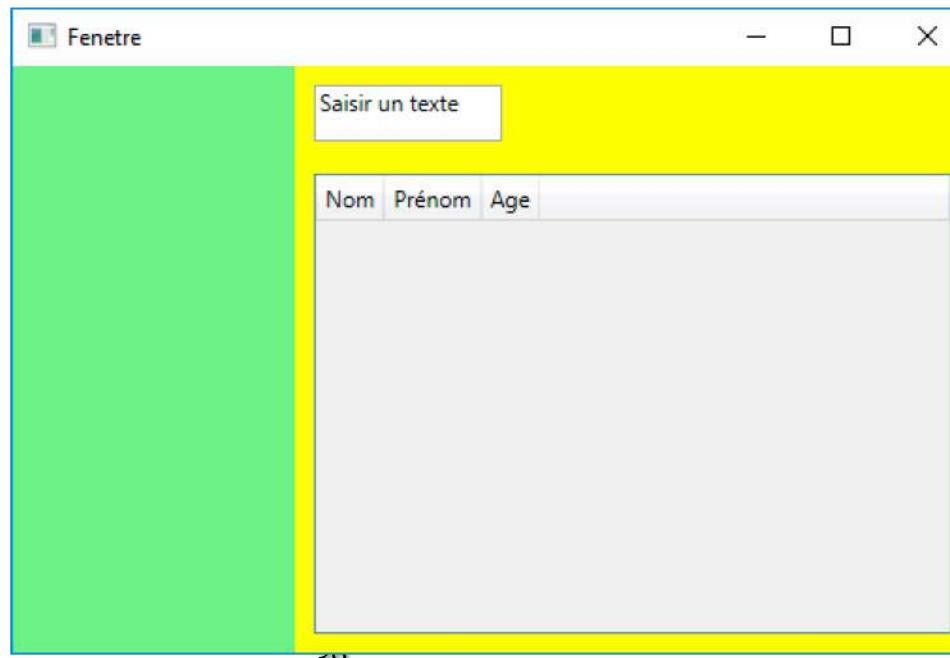
Stratégie de conception

- **Wysiwyg**
 - Mettre un conteneur (Grid généralement)
 - Positionner les composants
 - Editer le fichier XAML pour affiner le rendu

- **Astuce : ancrage**

Stratégie de conception

- **Wysiwyg**
 - Mettre un conteneur (Grid généralement)
 - Positionner les composants
 - Editer le fichier XAML pour affiner le rendu
- **Astuce : ancrage**
- **Exemple :**



Stratégie de conception

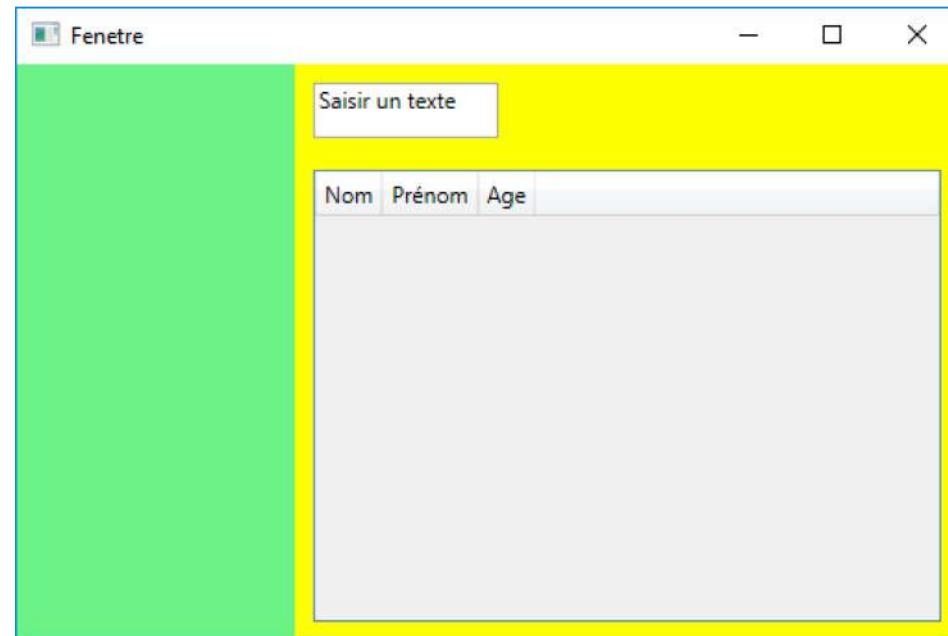
- **Créer une grille contenant 2 colonnes.**
- **Dans la colonne de droite déposer le TextBox et le DataGrid**
- **Corriger l'XAML si nécessaire et surtout pour supprimer le code superflu.**
- **Affiner l'alignement grâce aux points d'ancrage (utiles en cas de redimensionnement)**

Stratégie de conception



Code XAML

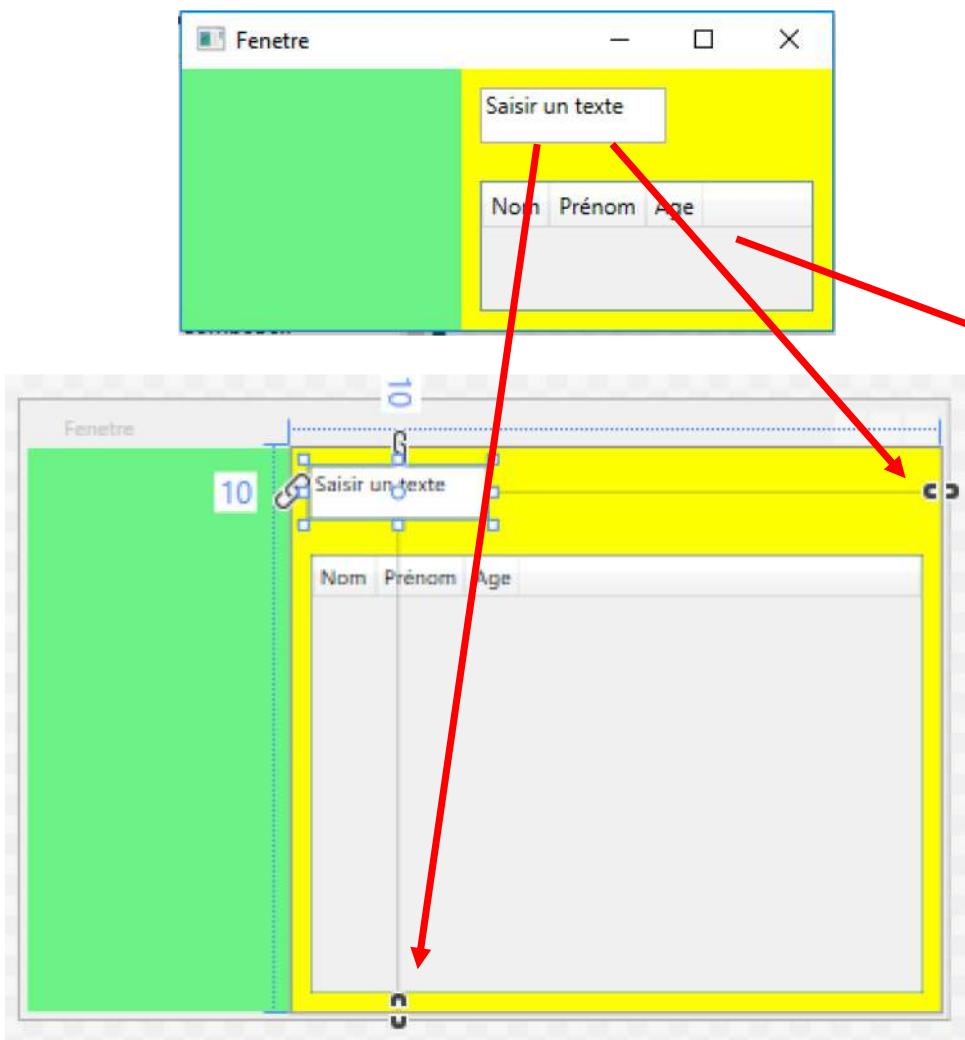
```
<Grid x:Name="Grille" Grid.ColumnSpan="2">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="150"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid Grid.Column="0">
        <Border Background="LightGreen"/>
    </Grid>
    <Grid Grid.Column="1">
        <Border Background="Yellow"/>
        <DataGrid Margin="10,60,10,10">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Nom"/>
                <DataGridTextColumn Header="Prénom"/>
                <DataGridCheckBoxColumn Header="Age"/>
            </DataGrid.Columns>
        </DataGrid>
        <TextBox Margin="10,10,0,0" HorizontalAlignment="Left" Text="Saisir un texte" Width="100" Height="30" VerticalAlignment="Top"/>
    </Grid>
</Grid>
```



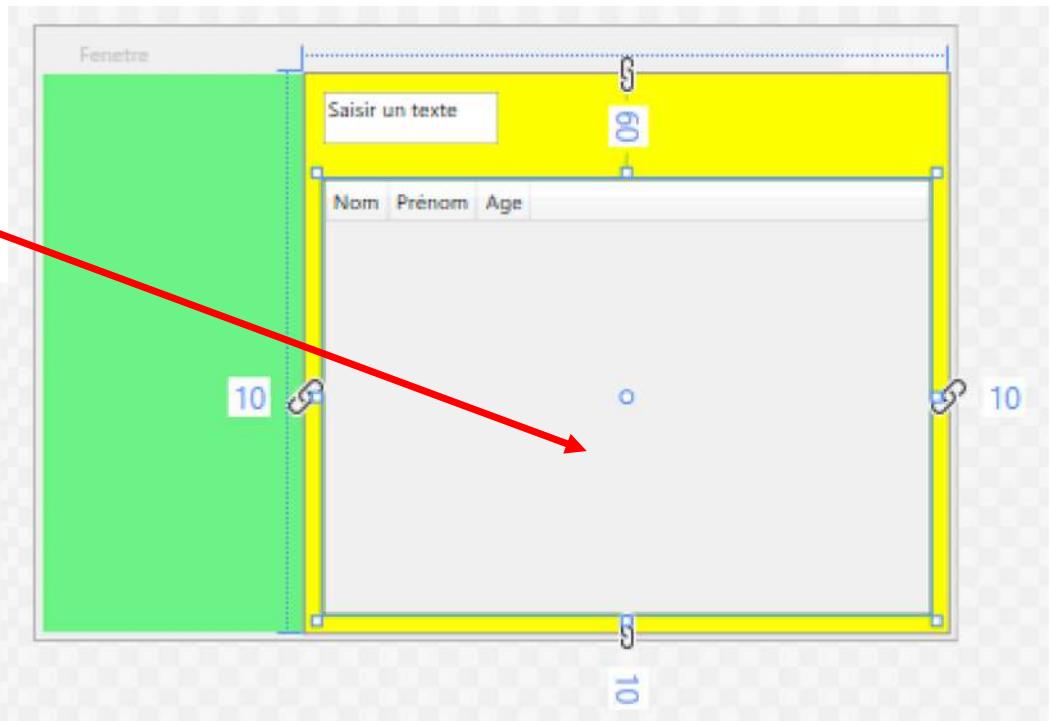
Stratégie de conception



Ancrage :



DataGrid adaptatif et champ de saisie fixe



Interaction clavier souris

Il est essentiel de pouvoir réagir aux interactions de l'utilisateur et le logiciel pour fournir une expérience fluide et cohérente.

WPF fournit différentes événements déclenchés lorsque l'utilisateur effectue une action au clavier ou à la souris.

Événements clavier

- 2 événements déclenchés :
 - **KeyDown**
 - **KeyUp**
- Le traitement s'appuie le paramètre **KeyEventArgs** qui permet de connaître la touche physique du clavier impliquée dans l'événement déclenché ainsi que son état :
 - **Key, IsUp IsDown et KeyStates**
- Pour les combinaison de touches (CTRL ALT SHIFT) on utilisera la propriété **Modifiers**
- Dans le code XAML on peut ajoute le gestionnaire l'événement sur le contrôle (**KeyUp="..."**)
- Pour un gestionnaire clavier sur l'ensemble de la fenêtre on utilisera l'événement **KeyUp** sur la fenêtre principale mais on peut l'ajouter de façon dynamique dans le code (concerne toute la page)

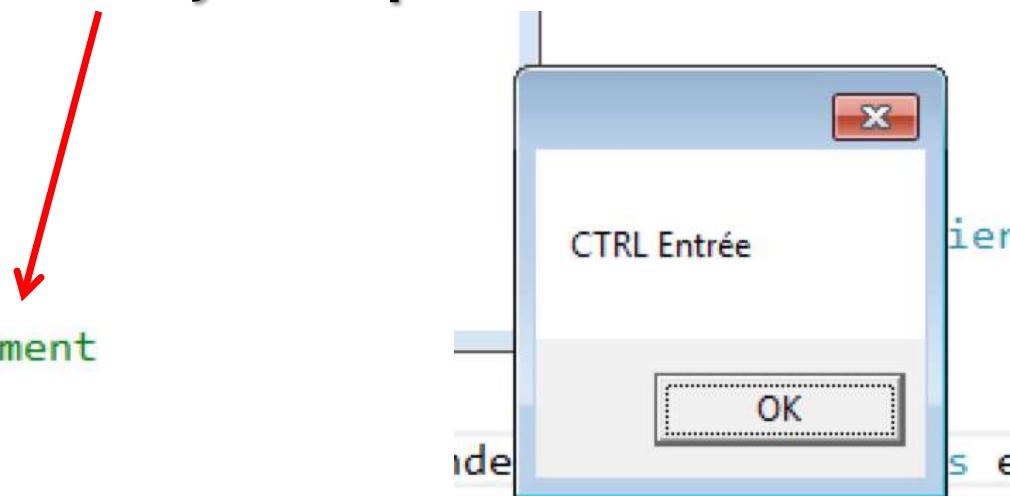
Événements clavier



Exemple avec combinaison de touches et gestionnaire d'événement dynamique sur la fenêtre principale :

```
public MainWindow()
{
    InitializeComponent();
    // ajout gestionnaire evenement
    KeyUp += Test;
}

private void Test(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter && Keyboard.Modifiers == ModifierKeys.Control)
        MessageBox.Show("CTRL Entrée");
}
```



Événements souris

- ➊ Beaucoup plus d'événements que le clavier (environ une dizaine)
 - ➌ **MouseDown MouseUp**
 - ➌ **MouseLeftButtonDown MouseLeftButtonUp**
 - ➌ **MouseRightButtonDown MouseRightButtonUp**
 - ➌ **MouseMove**
 - ➌ **MouseGetPosition(contrôle).X**
 - ➌ **MouseGetPosition(contrôle).Y**
 - ➌ **MouseEnter MouseLeave**

Glisser-déposer

- **Gestion du glisser déposer simplifié :**
 - Chaque contrôle peut-être une source ou une cible
 - La méthode statique **DoDragDrop** de la classe **DragDrop** implémente le stockage temporaire des données déplacées et s'occupe également de la gestion des effets visuels associée à l'opération
 - Plusieurs événements permettent de fournir des traitements et des retours visuels à l'utilisateur en fonction de l'évolution de l'opération
 - **Ex : déplacer un disque rouge sur un rectangle vert colore le rectangle en rouge**

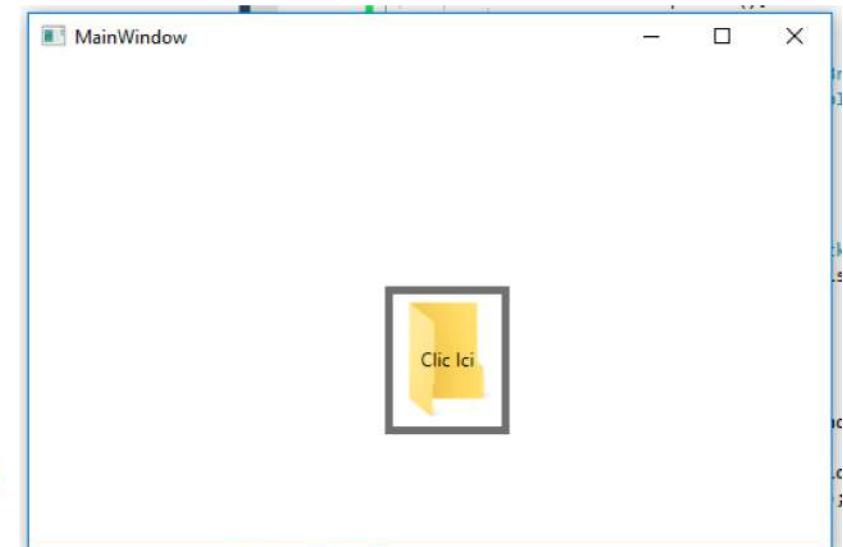
Code behind

- Il est possible d'accéder aux composants directement en c# (code behind)
- Il est possible de faire du code behind pour générer dynamiquement des composants y compris les événements associés

Code behind

```
public partial class MainWindow : Window
{
    public Button b;
    O références
    public MainWindow()
    {
        InitializeComponent();
        b = new Button();
        b.Content = "Clic Ici";
        ImageBrush imgB = new ImageBrush();
        BitmapImage img = new BitmapImage(new Uri("pack://application:,,,/Img/Capture.PNG"));
        imgB.ImageSource = img;
        b.Background = imgB;
        b.Margin = new Thickness(20, 50, 0, 0);
        b.Height = 94;
        b.Width = 79;
        b.BorderThickness = new Thickness(5);
        b.Visibility = Visibility.Visible;
        Grille.Children.Add(b);
        b.Click += OnbClick;
    }

    1 référence
    private void OnbClick(object sender, RoutedEventArgs e)
    {
        b.Visibility = Visibility.Hidden;
        MessageBox.Show("ça marche");
    }
}
```



Persistante



Base de données :



Serveur : SQL serveur par exemple



Local : LocalDB



Fichier :



Json avec le package de Newtonsoft :

Parcourir Installé Mises à jour Consolider

Gérer les packages de la solution

Source de package : nuget.org

json

JSON par Daniel Crenna, 50,5K téléchargements v1.0.1
A JSON parser in C#, supporting dynamic deserialization in .NET 4.0. A JSON parser in C#, supporting dynamic deserialization in .NET 4.0. Runs on .NET 3.5, .NET 4.0, Silverlight, a...

Newtonsoft.Json par James Newton-King, 52,9M téléchargements v10.0.1
Json.NET is a popular high-performance JSON framework for .NET

RestSharp par John Sheehan, RestSharp Community, 4,2M téléchargements v105.2.3
Simple REST and HTTP API Client

json-serialize par Kevin Malakoff, 10,6K téléchargements v1.1.2
JSON-Serialize.js provides conventions and helpers to manage serialization and deserialization of instances to/from JSON.

Chaque package vous est concédé sous licence par son propriétaire. NuGet n'est pas responsable des packages tiers et n'octroie aucune licence les concernant.

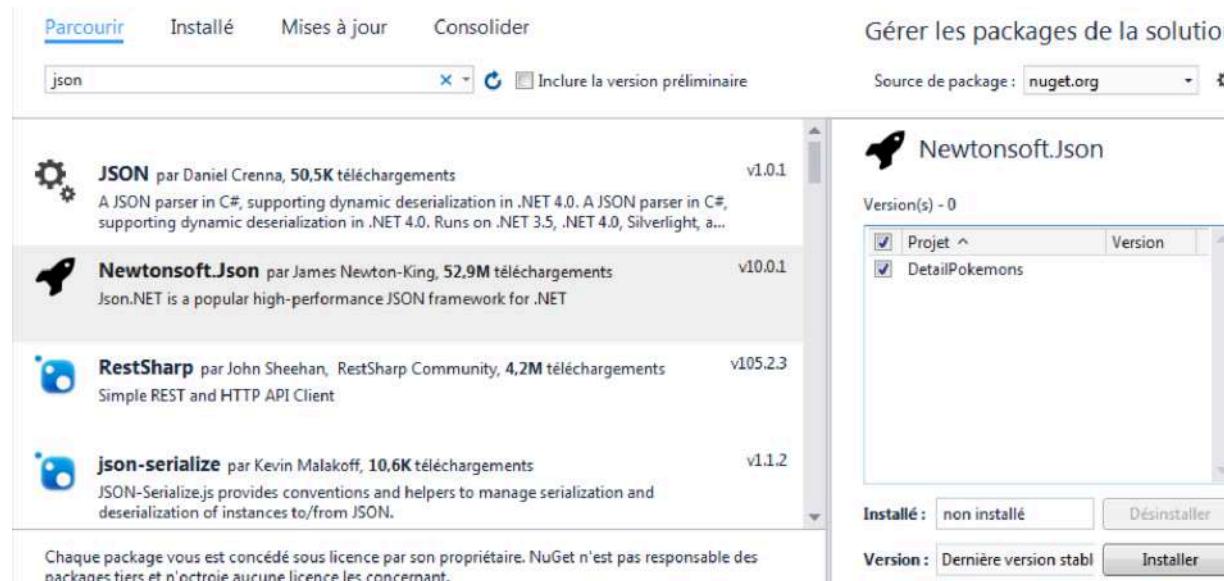
Newtonsoft.Json

Version(s) - 0

Projet	Version
DetailPokemons	v10.0.1

Installé : non installé Désinstaller

Version : Dernière version stable Installer



Persistante



Ex : Collection de Pokemon dans une ListBox

```
public void ChargeListePokemonJson()
{
    try {
        listePokemons = JsonConvert.DeserializeObject<List<Pokemon>>(File.ReadAllText("./Data/pokemon.json"));
        listBoxPokemons.ItemsSource = listePokemons;
    }
    catch {
        MessageBox.Show("Lecture fichier impossible, l'application va se fermer"
            , "erreur", MessageBoxButton.OK, MessageBoxImage.Error);
        Environment.Exit(0);
    }
}

public void SauveListePokemonJson()
{
    try {
        File.WriteAllText("./Data/pokemon.json", JsonConvert.SerializeObject(listePokemons));
    }
    catch {
        MessageBox.Show("Ecriture fichier impossible", "erreur", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Et après le développement ?

- ➊ La phase de **déploiement** est une étape indispensable afin de diffuser son application aux utilisateurs.
- ➋ On utilisera de préférence **Windows Installer** (service de déploiement Microsoft) à l'aide du logiciel **InstallShield Limited Edition**.
- ➌ InstallShield Limited Edition est mis à disposition gratuitement pour les utilisateurs VisualStudio, mais il doit être installé auparavant (Nouveau Projet – Autres types de projets – Activer InstallShield Limited Edition)
- ➍ Une fois installé, on peut alors créer un projet d'installation (Nouveau Projet – Autres types de projets – Configuration et déploiement - InstallShield Limited Edition)
- ➎ Un assistant de projet est alors ouvert pour guider l'utilisateur dans la création de son projet d'installation.