WPF

Windows Presentation Foundation

1- Présentation

- Structure d’une fenêtre WPF
- Le fichier App.Xaml
- Les styles

1-1 Structure d’une fenêtre WPF

- L’attribut Class définit le nom complètement qualifié de la classe qui représente la fenêtre (de type System.Windows.Window).
- Les attributs xmlns définissent les schémas xml utilisés par la fenêtre.
- Les contrôles d’une fenêtre sont placés dans un conteneur de type Panel qui gère leur positionnement (Grid par exemple).

```
<Window x:Class="wpfApp1.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">

    <Grid>

    </Grid>

</Window>
```

1-2 Le fichier App.xaml

- Le fichier App.xaml définit les propriétés de l'application WPF.
- StartupUri: nom duf fichier xaml de la fenêtre de démarrage.

```
<Application x:Class="wpfApp1.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

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

1-3 Les styles

- Pour appliquer un style nommé sur un contrôle, utilisez la propriété Style:
  - Style="{StaticResource styleBouton}" : le style est appliqué au moment de l'exécution
  - Style="{DynamicResource styleBouton}": le style est appliqué au moment de l'exécution

```
<Window.Resources>
        <!-- Un style nommé -->
        <Style x:Key="styleBouton" TargetType="Control">
            <Setter Property="Background" Value="Gray"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="FontFamily" Value="Comic Sans MS"/>
         </Style>
```

```
<!-- Un style qui s'applique sur tous les contrôles TextBox -->
        <Style TargetType="{x:Type TextBox}">
            <Setter Property="FontFamily" Value="Comic Sans MS"/>
            <Setter Property="HorizontalAlignment" Value="Center"/>

            <Setter Property="Foreground" Value="Blue" />
            <!-- Style qui s'applique si la propriété MouseOver
vaut True-->
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="Blue" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>
```

2 Les contrôles WPF

- 3 types de contrôles:
  - Simples
  - Listes
  - Conteneurs

2.1 Les contrôles simples de type Content

- Ces contrôles possèdent la propriété Content (de type object), si l'objet n'est pas de type UIElement alors sa méthode ToString() est invoquée.
- Label
  - Mnémoniques: `<Label  Target="{Binding ElementName=textBox1}">Afficher Alt+_A</Label>`
- Button
  - Propriétés
    - IsDefault, IsCancel
    - Raccourcis : `<Button>_Cliquez ici</Button>`
- CheckBox: ButtonBase
  - Prorpiétés:
    - IsChecked
    - `IsThreeState:si =True, alors 3 états sont possibles: True,False,Null`
- RadioButton:ButtonBase
- Propriétés:
  - GroupName: pour créer des groupes de boutons radio, les boutons radio qui sont dans des conteneurs différents forment aussi des groupes.

2.2 autres types de contrôles simples

- TextBlock: afficher du texte (Text)
- Image:
  - Propriétés
    - Source
    - Stretch: none,Fill, Uniform, UniformToFill.
- TextBox
  - Propriétés
    - IsReadOnly
    - TextWrapping="Wrap ou WrapWithOverflow«
    - VerticalScrollBarVisibility="Visible ou Auto'"
- ProgressBar
  - Propriétés
    - IsEnabled
    - IsIndeterminate: false: valeur actuelle , true: progression générique
    - *LargeChange, Maximum, Minimum, Orientation (Vert ou Horiz), SmallChange, Value.*
- *Slider*
  - *Prorpriétés: IsDirectionReversed, IsEnabled, LargeChange, Maximum, Minimum, Orientation (Vert ou Horiz), SmallChange, Value, TickFrequency, TickPlacement, Ticks*
  - *Evénements: ValueChanged*

2.3 Les listes

- ListBox
  - Est composé d'éléments d'un type quelconque (exemple: ListBoxItem, CheckBox).
  - Propriétés: *SelectedIndex, SelectedItem, SelectionMode (Single, Multiple, Exetended (shift ou ctrl), SelectedItems.*
  - *ListBoxItem*
    - *Propriétés: IsSelected.*
- *ComboBox:*
  - *Propriétés: IsReadOnly, IsEditable*

TreeView

- Composé de « TreeViewItem ».
- TreeViewItem:
  - Un TreeViewItem peut être composé d’un ou de plusieurs TreeViewItem.
  - Propriétés: header

Menu

- MenuItem
  - Propriétés: *Command, Header, Icon, IsChecked, IsEnabled, Items*
- *Un Menu contextuel peut être associé à un contrôle, des menus contextuls peuvent aussi être ajoutés dans la collection Window.Resources.*

```
<ListBox Margin="77,123,81,39" Name="listBox1">
                <ListBox.ContextMenu>
                    <ContextMenu>
                        <MenuItem Header="Cut"
Command="ApplicationCommands.Cut"/>
                        <MenuItem Header="Copy"
Command="ApplicationCommands.Copy"/>
                        <MenuItem
Header="Paste"
Command="ApplicationCommands.Paste"/>
                    </ContextMenu>
                </ListBox.ContextMenu>
            </ListBox>
```

```
<Menu Height="22" Name="menu1"
VerticalAlignment="Top"
HorizontalAlignment="Left"
Width="278">
            <MenuItem Header="_File">
                <MenuItem
Header="Open"/>
                <MenuItem
Header="Close"/>
                    <Separator/>
                <MenuItem
Header="Save"
Command="ApplicationCommands.Save"/>
            </MenuItem>
        </Menu>
```

Toolbar

- Propriétés
  - OverflowMode: Always, AsNeeded, Never.
- ToolbarTry est un conteneur pour les contrôles Toolbar
- StatusBar

```
<ToolBar Height="26"
Margin="43,23,35,0" Name="toolBar1"
VerticalAlignment="Top">
            <Button>Back</Button>
            <Button>Forward</Button>
            <TextBox Name="textbox1"
Width="100"/>
        </ToolBar>
```

```
<StatusBar Height="32" Name="statusBar1"
VerticalAlignment="Bottom">
            <Label>Application is
Loading</Label>
            <Separator/>
            <ProgressBar Height="20"
Width="100" IsIndeterminate="True"/>
        </StatusBar>
```

```
<ToolBarTray Name="toolBarTray1" Height="65"
VerticalAlignment="Top">
            <ToolBar Name="toolBar1"
Height="26" VerticalAlignment="Top">
                <Button>Back</Button>
                <Button>Forward</Button>
                <Button>Stop</Button>
            </ToolBar>
            <ToolBar>
                <TextBox Width="100"/>
                <Button>Go</Button>
            </ToolBar>
        </ToolBarTray>
```

2.4 les Conteneurs

- Propriétés
  - FlowDirection
  - Height
  - HorizontalAlignment: Left,Right, Center, et Stretch.
  - HorizonalContentAlignment
  - Margin: une instance de la structure Thickness (gauche,haut,droite,bas)
  - MaxHeight
  - MaxWidth
  - MinHeight
  - MinWidth
  - Padding
  - VerticalAlignment: Bottom, Center, et Stretch
  - VerticalContentAlignment
  - Width
- Propriétés attachées
  - Exemple: `<Button Grid.Row="1" Grid.Column="1"></Button>`
    NomClasseConteneur.NomPropriété
  - Dans certains cas le nom de la classe est optionnel, exemple : propriété TabIndex.

Cadres de positionnement

- Grid
  - Propriétés attachées: Grid.Row , Grid.Column, Grid.RwoSpan, Grid.ColumnSpan, IsSharedSizeScope.
- GridSplitter
  - Propriétés
    - *Grid.Column*
    - *Grid.ColumnSpan*
    - *Grid.Row*
    - *Grid.RowSpan*
    - *Height*
    - *HorizontalAlignment*
    - *Margin*
    - *ResizeBehavior*
    - *ResizeDirection*
    - *ShowsPreview*
    - VerticalAlignment
    - Width

```
 <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <GridSplitter Grid.Column ="0"
Background="Blue" Width="5"

HorizontalAlignment="Right"
VerticalAlignment="Stretch"/>
        </Grid>
```

```
<Grid.RowDefinitions >
            <RowDefinition Height="1*"/>
            <RowDefinition Height="2*"/>
            <RowDefinition Height="3*"/>
 </Grid.RowDefinitions>
```

- UniformGrid: fixer le nombre de lignes et de colonnes alors, fixe aussi le nombre de contrôles qui peuvent ajoutés dans le contrôle UniformGrid

```
<UniformGrid Rows="2" Columns="2">
        </UniformGrid>
```

- StackPanel: permet d'afficher une liste de contrôles du haut vers le bas, ou horizontalement de gauche à droite ou de droite à gauche.

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

- WrapPanel

```
<WrapPanel FlowDirection="RightToLeft">
        </WrapPanel>
```

- DockPanel
  - Propriétés :
    - Propriété attachée: DockPane.Dock (Left, Top, Right, Bottom)
    - LastChildFill: (True par défaut)

```
<WrapPanel FlowDirection="RightToLeft">
        </WrapPanel>
```

- Exemple 2

```
<DockPanel>
        <Menu DockPanel.Dock="Top">Un menu</Menu>
        <ListBox DockPanel.Dock="Left">Une Liste</ListBox>
        <Grid >
            <Label HorizontalAlignment="Center"
VerticalAlignment="Center">Une grille</Label>

        </Grid>
</DockPanel>
```

- Canvas: Permet un positionnement absolu des contrôles contenus.
  - Propriétés attachées:Canvas.Zindex, Canvas.Top, Canvas.Bottom, Canvas.Right, et Canvas.Left, la valeur de chacune de ces propriétés définit la distance entre le canvas et le contrôle. Les propriétés Left et Right ne peuvent être définies simultanément (idem pour Top et Bottom).

Accès aux éléments fils d'un conteneur par code

- Accéder à un élément fils

```
Button aButton;
aButton = (Button)grid1.Children[3];
```

- Ajouter un élément

```
Button aButton = new Button();
grid1.Children.Add(aButton);
```

- Supprimer un élément

```
grid1.Children.Remove(aButton);
grid1.Children.RemoveAt(3);
```

- Accéder à un élément fils

```
Button aButton;
aButton = (Button)grid1.Children[3];
```

La classe Binding

- La classe Binding permet d'établir une liaison entre un objet et une propriété cible.
- Propriétés:
  - ElementName: permet de définir l'élément WPF source de la liaison
  - Source: définit l'objet (dans le cas d'un élément WPF la propriété ElementName doit être utilisée au lieu de la propriété Source
  - FallbackValue: Valeur à utiliser par défaut.
  - Mode: définit la direction, valeurs: *TwoWay, OneWay, OneWayToSource*
  - NotifyOnSourceUpdated=true: l'événement SourceUpdated est déclenché si une valeur est transférée de la cible à la source de la liaison.
  - NotifyOnTargetUpdated=true: Déclenche l'événement TagrgetUpdated si une valeur est transférée de la source à la cible de la liaison.

- Path: définit le chemin à la propriété source de la liaison.
- TargetNullValue: définit la valeur de la cible à utiliser si la valeur de la source est nulle.

- Liaison à un élément WPF

```
<Slider Name="s1" Minimum="2" Width="100" Maximum="72"  Value="12"
/>
<RichTextBox Name="r1" Grid.Column="2" FontSize="{Binding
ElementName=s1, Path=Value}" />
```

- La propriété DataContext
  - Définit le contexte de données pour un élément et ses éléments fils.
  - Cette propriété sera utilisée comme source de données pour les éléments liés, et dont l'instance de la classe Binding ne définit ni la propriété Source ni la propriété ElementName.

Liaison à une liste (ListBox,ComboBox)

- Propriétés
  - DisplayMemberPath: propriété de la collection liée à afficher.
  - IsSynchronizedWithCurrentItem: Détermine si l'élément sélectionné est synchronisé avec la propriété CurrentItem dans la collection Items.
  - ItemsSource: une collection qui fournit les données pour la liste.