



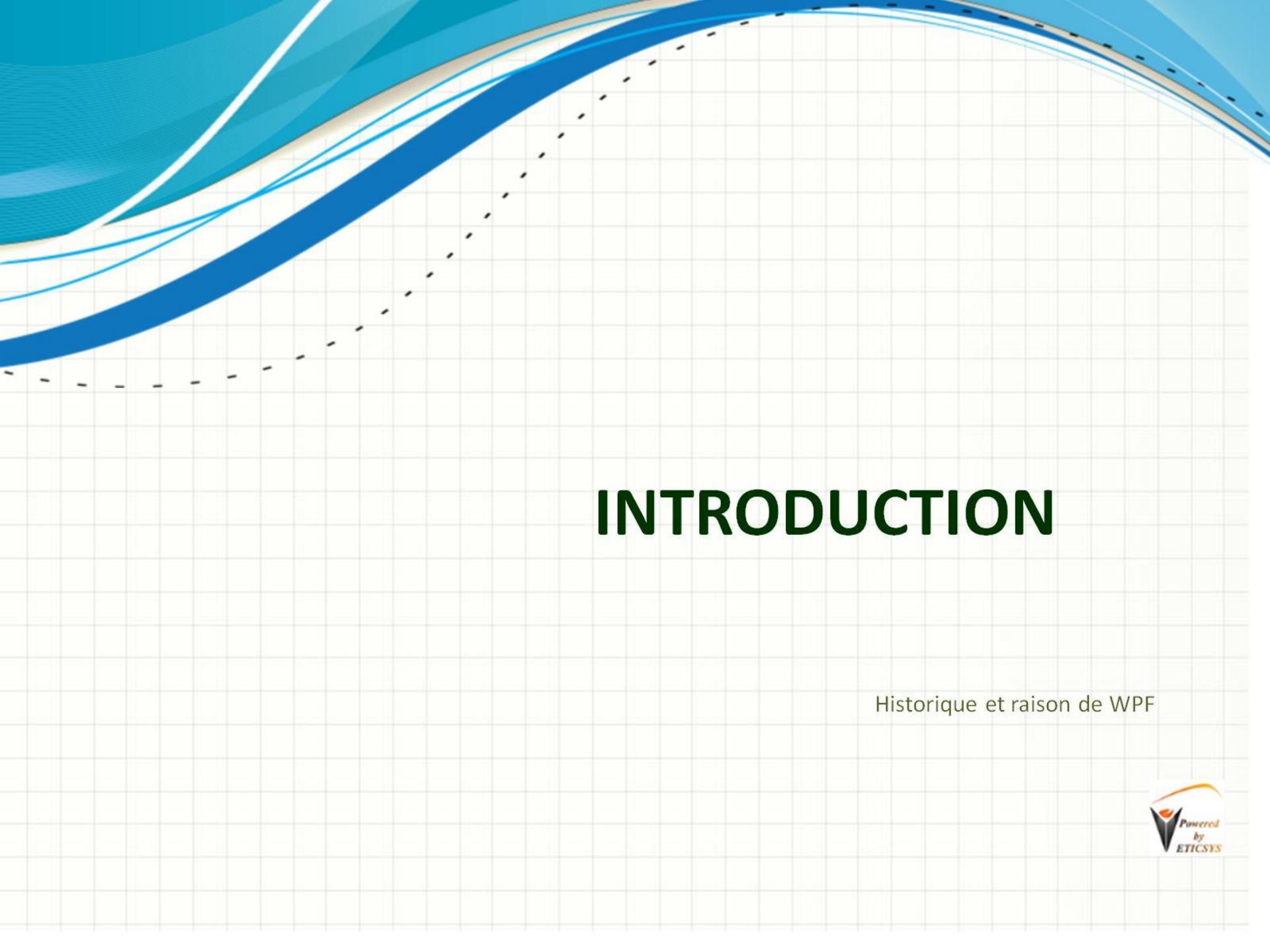
INTRODUCTION À WINDOWS PRESENTATION FOUNDATION - WPF -

François-Xavier BAILLET



Sommaire

- Introduction et présentation du XAML
- Les contrôles
- La liaison de données
- Formater les données



INTRODUCTION

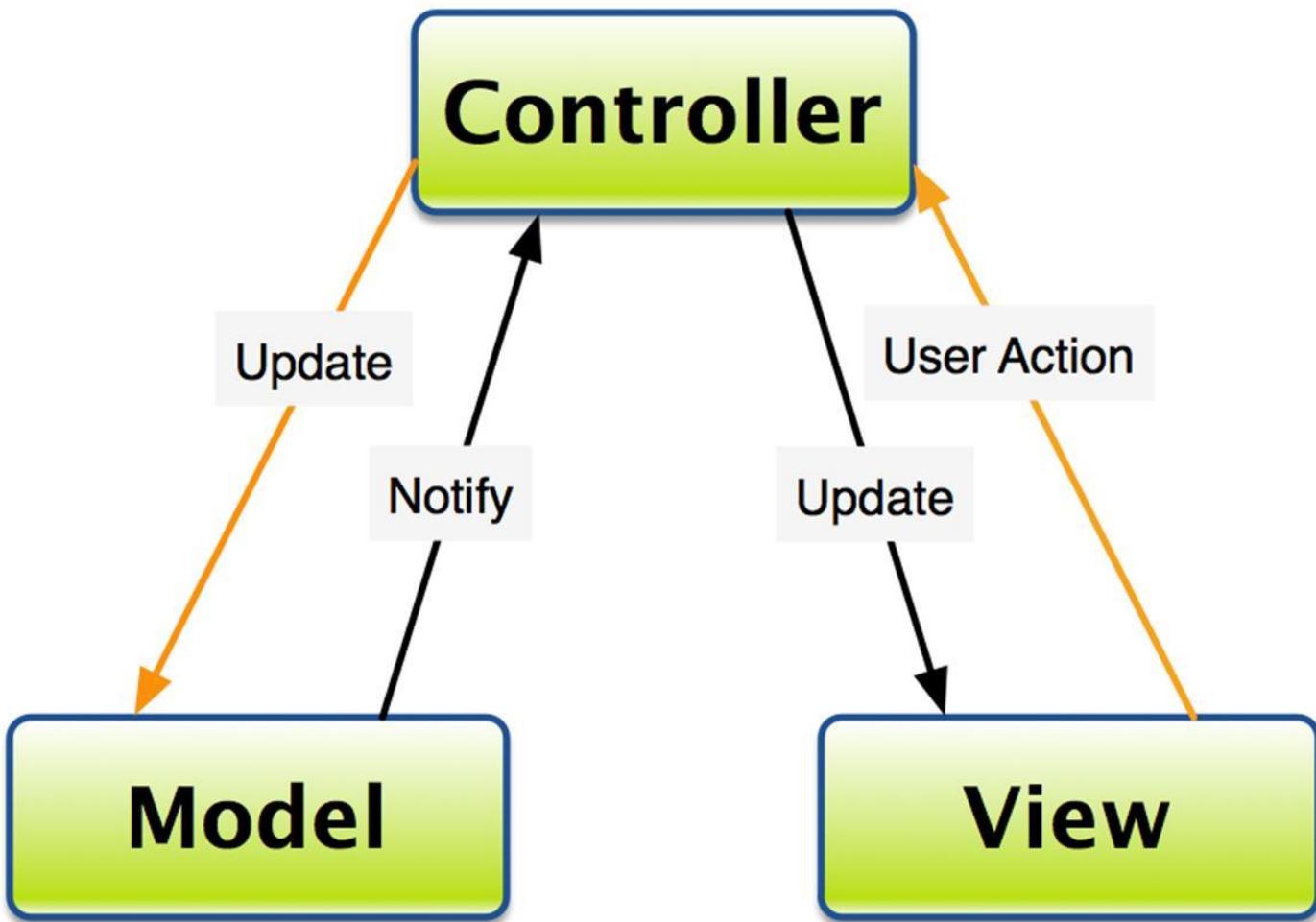
Historique et raison de WPF



Evolution des IHM Windows

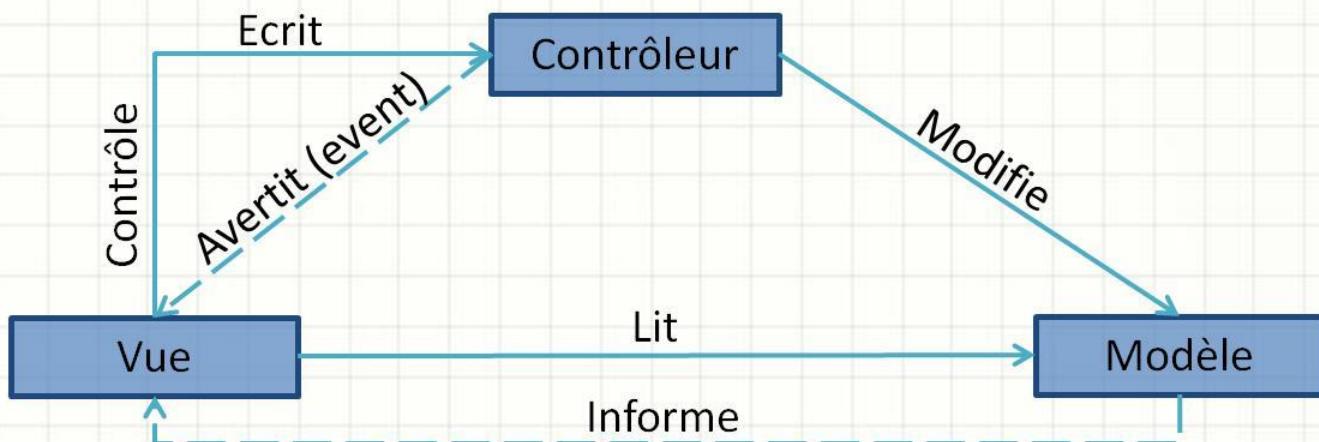
- Depuis plus de 15 ans, les graphiques sont basés sur User32 et GDI/GDI+
- Les interfaces :
 - VB6
 - MFC (c++)
 - Windows Forms
- Un changement important : WPF (2006)
- Un de plus : WinRT (2011)
- UWP avec Windows 10 ...

Le MVC



Le MVC => MVVM

- Architecture logiciel à 3 niveaux séparés permettant de séparer :
 - Les objets applicatifs (Modèle) : Les données
 - Leur représentation (Vue) : Interface graphique
 - Les interactions (Contrôleur) : Logique des actions utilisateurs



MVVM & C#

<http://japf.developpez.com/tutoriels/dotnet/mvvm-pour-des-applications-wpf-bien-architecturees-et-testables/>

- V : VIEW

C'est la partie XAML

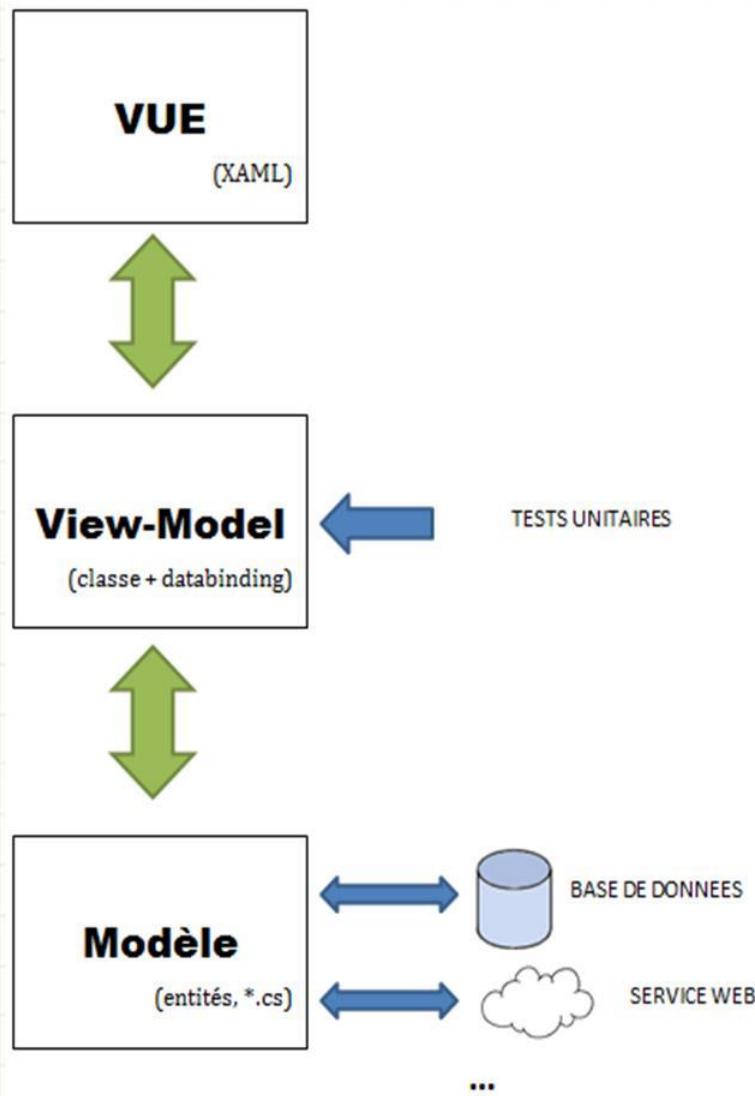
- M : MODEL

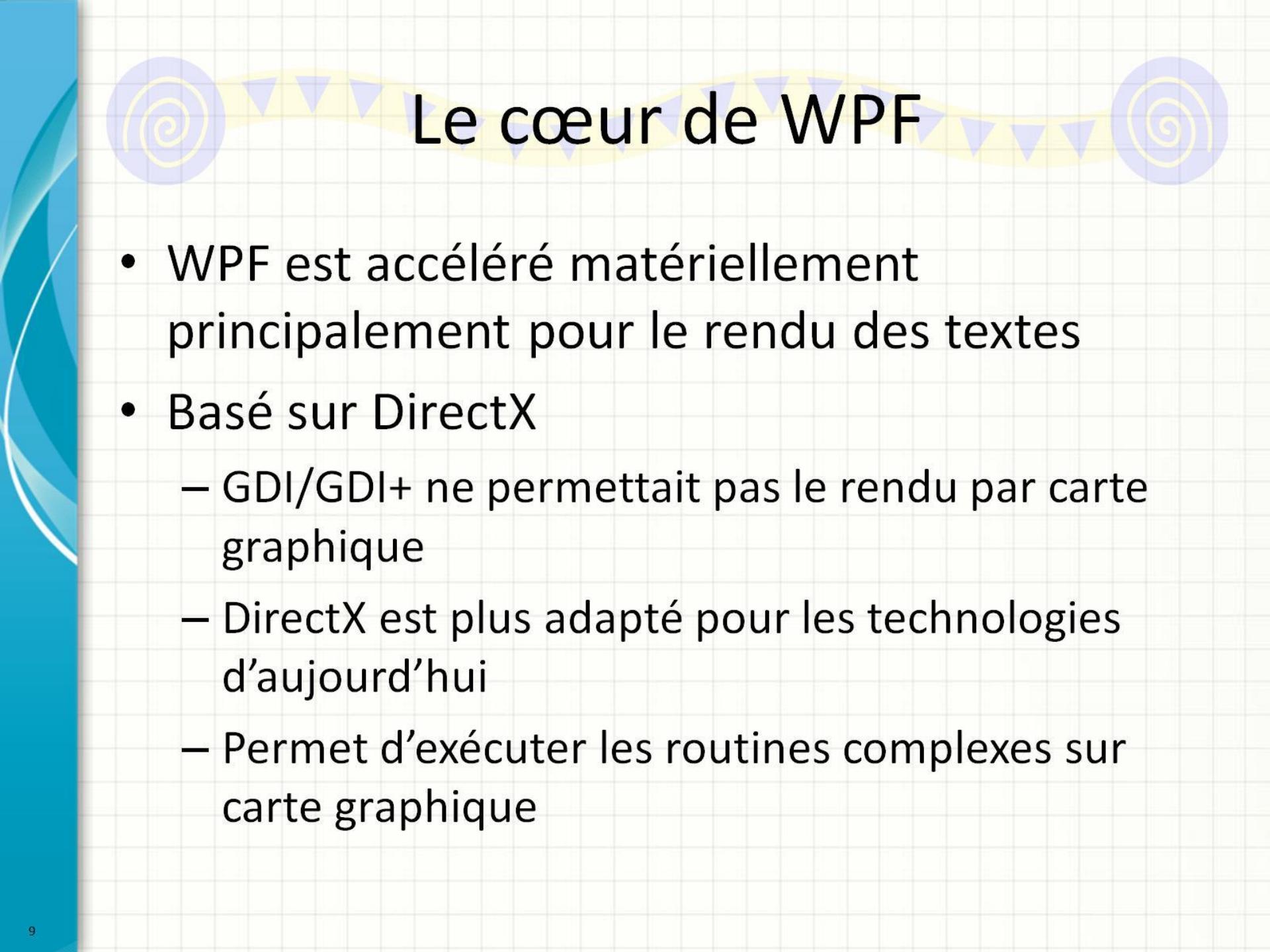
La gestion des données sous toutes ses formes

- VM : VIEWMODEL

Classe d'abstraction de la vue qui permet de la liaison entre V et M (databinding) et la gestion des commandes

MVVM & C# : Schéma





Le cœur de WPF

- WPF est accéléré matériellement principalement pour le rendu des textes
- Basé sur DirectX
 - GDI/GDI+ ne permettait pas le rendu par carte graphique
 - DirectX est plus adapté pour les technologies d'aujourd'hui
 - Permet d'exécuter les routines complexes sur carte graphique

WPF : Une API de haut niveau

- Positionnement "Web-like"
- Rich User Interface
- Rich Text Model
- Animations intégrées (programmation déclarative)
- Support Audio et Vidéo
- Permet les styles et templates
- Interface par programmation déclarative

Architecture WPF

<http://msdn.microsoft.com/fr-fr/library/ms750441.aspx>

PresentationFramework

PresentationCore

Common Language Runtime

milcore

User32

DirectX

Kernel



Intérêt du XAML

- eXtensible Application Markup Language
- Langage déclaratif basé sur le XML
- Le compilateur convertit les balises XAML en objets et attributs
- Permet de réaliser aussi bien des boutons classiques que des animations 3D
- Séparation possible des rôles entre designer et développeur

Inconvénients du XAML

- Moins efficace que du Windows Form.
- Plus limité sur les interfaces complexes.
- Lent sur certaines plateformes.
- Peu trivial au premier abord...
- Très lourd à manipuler
- Compliqué à maintenir.
- Une norme de plus (!)

Les bases du XAML

- Chaque tag XAML permet d'instancier un objet
- Comme le XML, on peut imbriquer des éléments
- On peut paramétrer les tags avec des attributs qui sont les propriétés des objets créés
- Tous les éléments XAML peuvent être créés en C#

Les bases du XAML

- Chaque contrôle WPF crée une classe
 - Qui hérite d'une classe conteneur (Control, Window, UserControl...)
 - Qui contient une méthode « InitializeComponent » dans le « code-behind »
- On nomme les éléments avec la propriété "Name" pour utiliser l'objet depuis le code C#

Les bases du XAML

- Propriétés simples :
 - <Grid UnePropriété="Valeur"/>
- Propriétés complexes :

```
<Grid UnePropriété="Valeur">
    < Grid.PropriétéComplexe> Value
    </Grid.PropriétéComplexe>
</Grid>
```

Composition du XAML

- Pour chaque fichier classe XAML, un fichier C# est généré, son code associé en C# est une classe partielle

OK

XAML

```
<Button Width="100px"> OK  
<Button.Background>  
    LightBlue  
</Button.Background>  
</Button>
```

C#

```
Button b1 = new Button();  
b1.Content = "OK";  
b1.Background = new  
SolidColorBrush(Colors.LightBlue);  
b1.Width = new Length(100);
```



Comment faire une belle interface en TP ?

LES CONTRÔLES

- DE POSITIONNEMENT (CONTENEURS)

Pourquoi placer ses éléments par Layout ?

- Le but premier : faire une interface utilisateur
 - Agréable
 - Attractive
 - **Flexible** !
- L'interface doit s'adapter au contenu et à son environnement
- WPF propose plusieurs types de conteneurs

Le contrôle Canvas

- Permet de positionner des éléments en coordonnées X et Y, avec les propriétés
 - Left,Top,Right,Bottom
 - utilise le DIP (pixel logique indépendant de la résolution) : 1"=2,54cm=96Dip
- Comment positionner un élément :

```
<CheckBox Canvas.Left="12" Canvas.Top="25" Content="ma_checkbox"/>
```

ou

```
<Canvas>
```

```
    <Image Source="logo.gif" Canvas.Left="100" Canvas.Top="50"/>
```

```
</Canvas>
```

Exemple

Le contrôle StackPanel

- Empile les éléments verticalement ou horizontalement
 - Propriété Orientation (Vertical ou Horizontal)
- Possibilité de virtualiser l'affichage du contenu pour plus de réactivité : VirtualizedStackPanel
- Le StackPanel coupe les contrôles si la fenêtre est trop petite : utiliser le WrapPanel pour résoudre le problème (mêmes propriétés)

→ Utile pour les menus dynamiques

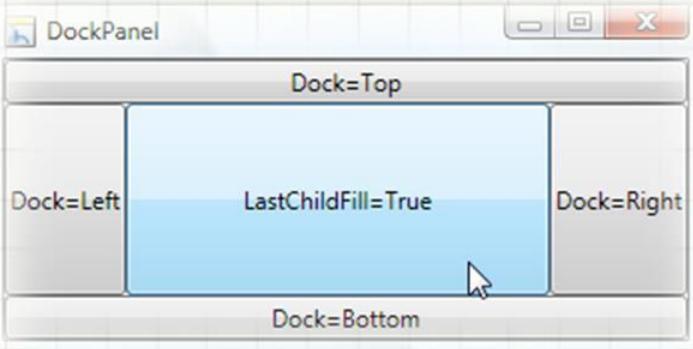
```
<StackPanel Margin="10">
    <Label>Username :</Label>
    <TextBox></TextBox>
    <Button>Connexion</Button>
    <Label>Menu n°1</Label>
    <Label>Menu n°2</Label>
    <Label>Menu n°3</Label>
</StackPanel>
```

Le contrôle Grid

- Permet de placer les éléments dans une grille
 - Donner le numéro de ligne et de colonne pour chaque élément interne
 - <CheckBox Grid.Row=« 2 » Grid.Column=« 3 »/>
- Définition des tailles de lignes et colonnes : Grid.RowDefinitions : taille fixe, automatique ou remplissage de l'espace vide (Width = 25, auto, *)
- Comme en HTML, il est possible de fusionner les lignes et colonnes : Grid.ColumnSpan=« 2 »

Le contrôle DockPanel

- Permet de placer les éléments autour d'un élément central (on connaît bien en WinForm)
- Positionner les éléments sur les bords (Left, Top, Right, Bottom) puis placer un élément au centre qui va remplir l'espace restant
- `<CheckBox DockPanel.Dock=<< Top >>/>` va placer une Checkbox en haut de l'interface



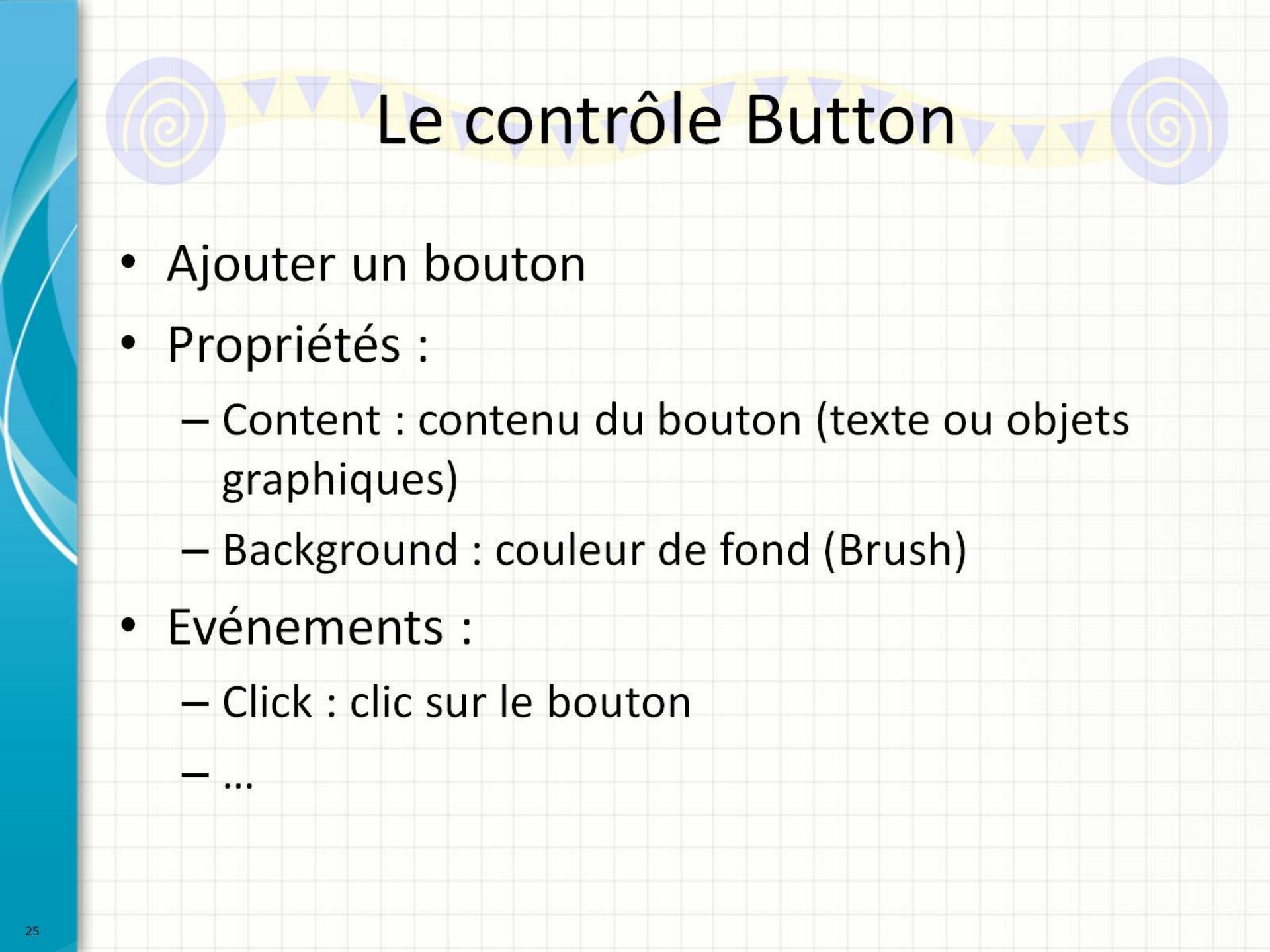


Comment faire une belle interface en TP ?

LES CONTRÔLES

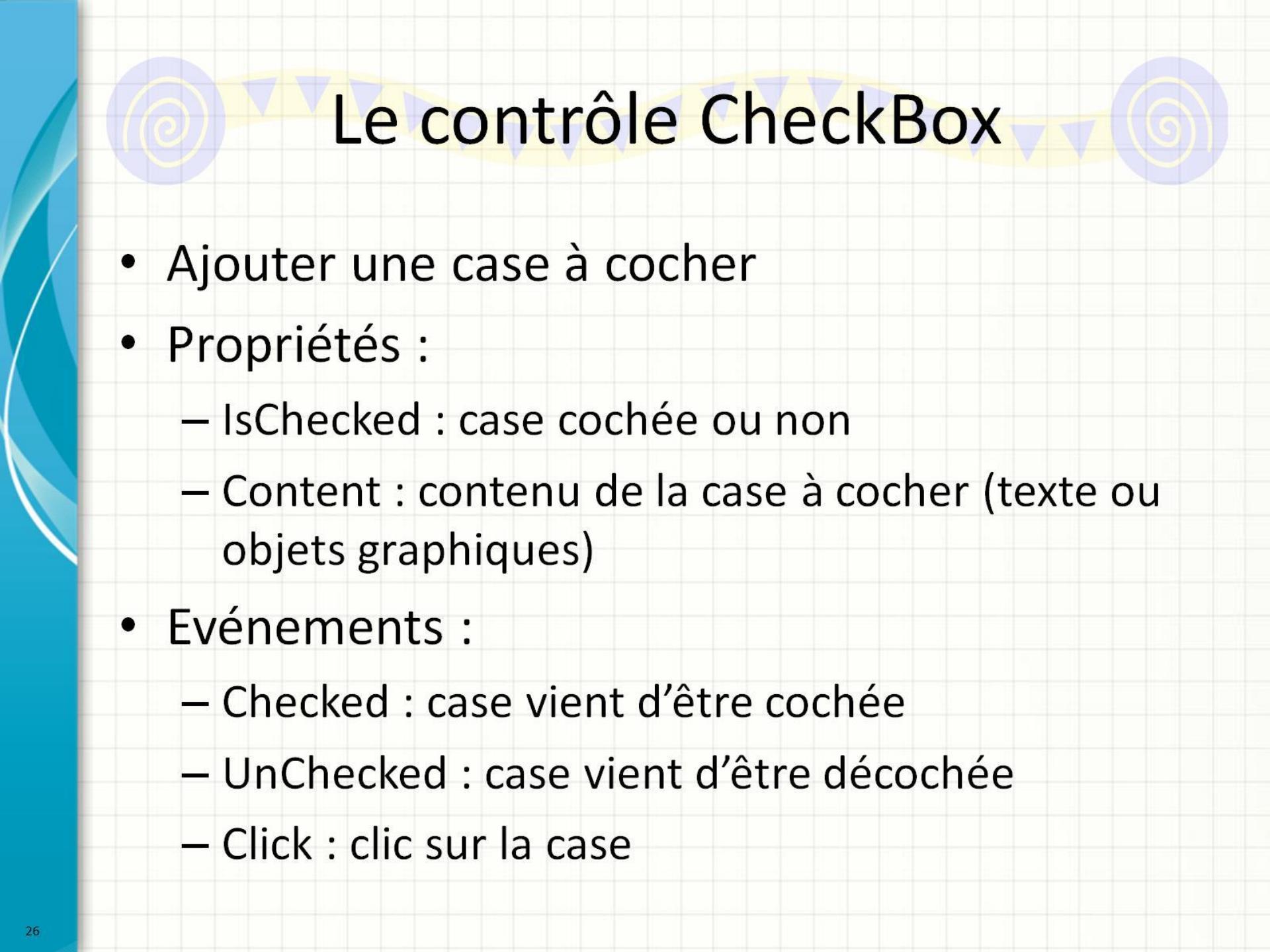
- BASIQUES





Le contrôle Button

- Ajouter un bouton
- Propriétés :
 - Content : contenu du bouton (texte ou objets graphiques)
 - Background : couleur de fond (Brush)
- Evénements :
 - Click : clic sur le bouton
 - ...



Le contrôle CheckBox

- Ajouter une case à cocher
- Propriétés :
 - IsChecked : case cochée ou non
 - Content : contenu de la case à cocher (texte ou objets graphiques)
- Evénements :
 - Checked : case vient d'être cochée
 - UnChecked : case vient d'être décochée
 - Click : clic sur la case

Le contrôle RadioButton

- Ajouter un bouton radio (choix unique parmi une liste)
- Propriétés :
 - IsChecked : bouton coché
 - Content : contenu du bouton (texte ou objets graphiques)
 - GroupName : nom du groupe de radio (pour la sélection/désélection automatique)
- Evénements :
 - Checked : le bouton vient d'être coché
 - UnChecked : le bouton vient d'être décoché
 - Click : clic sur le bouton radio

Le contrôle ListBox

- Ajouter une liste d'éléments
- Propriétés :
 - Content : ajout de « ListBoxItem » ou d'autres éléments de contenu
 - SelectedIndex : n° de l'élément sélectionné
 - SelectedItem : objet sélectionné
- Evénements :
 - SelectionChanged : l'élément sélectionné à changé

Le contrôle TreeView

- Comme ListBox, mais un TreeViewItem peut contenir d'autres TreeViewItem :
 - ➔ Permet de représenter des hiérarchies, tables des matières
 - ➔ [Exemple simple](#)
Et en prime et en avant première
le même exemple mais avec
[un "binding" de données](#)

Faux espoir, on y reviendra ;-)

Le contrôle ComboBox

- Comme ListView, mais sous forme de liste déroulante

Le contrôle Label

- Afficher du texte ou du contenu
- Propriétés :
 - Content : contenu du label (texte ou objets graphiques)

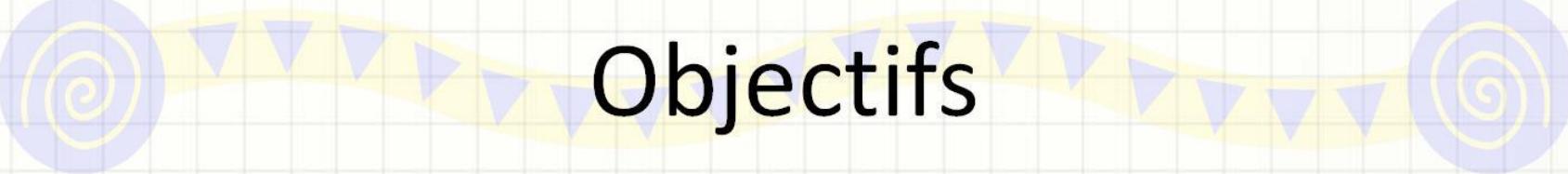
Le contrôle TextBox

- Afficher du texte modifiable
- Propriétés :
 - Text : contenu du TextBox affiché
 - IsReadOnly : modifiable ou non
- Evénements :
 - TextChanged : un caractère a été inséré ou supprimé



Mise en place du DataBinding

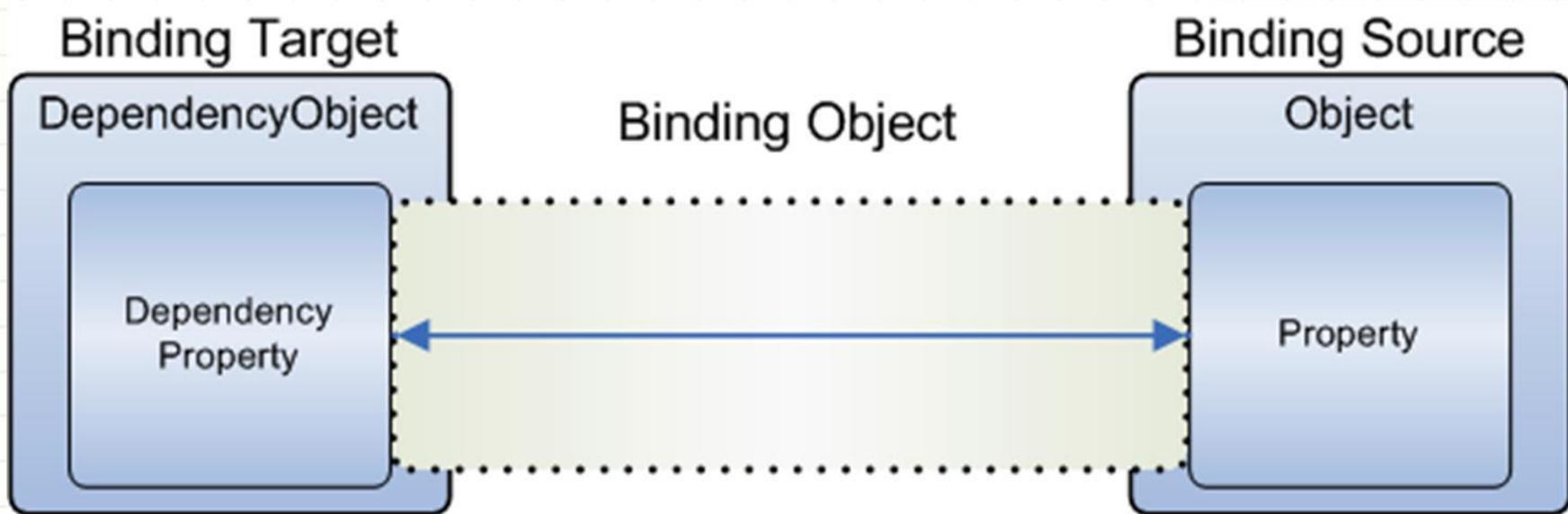
LA LIAISON DE DONNÉES



Objectifs

- La liaison de données permet de "lier" une interface utilisateur et les données sous-jacentes
 - Ex : Nous voulons lier un booléen et une CheckBox. Quand la donnée sera mise à jour, le CheckBox sera coché ou décoché et inversement.
- Permet de supprimer une dépendance forte (de code) entre la programmation métier et le graphisme.
 - Il devient "simple" de changer une interface graphique sans refaire le code métier

Modèle du DataBinding

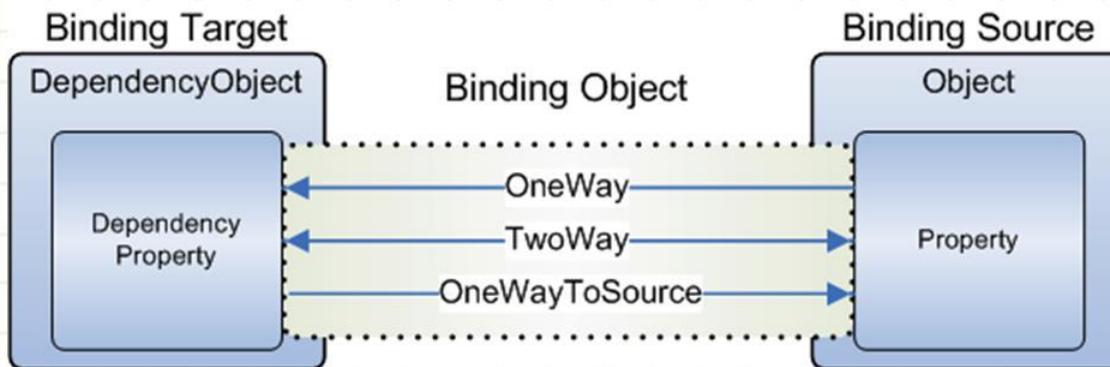


Modèle du DataBinding

- Une liaison est composée de :
 - Un objet cible (Ex : Composant CheckBox)
 - Une propriété cible (Ex : IsChecked)
 - Une source de liaison (Ex : Un objet métier)
 - Une propriété source (Ex : Un booléen)
- Il faut que la propriété cible soit une *Dependency Property*
 - La majorité des propriétés des contrôles le sont !
 - A garder en tête si on réalise son propre contrôle

Options du DataBinding

- Le mode :
 - OneWay
 - TwoWay : mode par défaut pour la majorité des contrôles
 - OneWayToSource
 - OneTime : source initialise la cible sans propagation ultérieure des changements



Options du DataBinding

- Définition du moment de mise à jour de la donnée : propriété UpdateSourceTrigger
 - **PropertyChanged** : Source mise à jour dès que la cible a changé
 - **LostFocus** : Source mise à jour lorsque l'élément graphique perd le focus
 - **Explicit** : Mise à jour lors de l'appel de la méthode UpdateSource (sur un objet BindingExpression)
 - **Default** : Pour la majorité des composants = PropertyChanged

Les `Dependency Property'

- Propriété classique du langage avec les particularités de pouvoir :
 - Etre animées
 - Etre transformées
 - Etre liées à l'interface utilisateur
 - Notifier l'interface lorsque la valeur associée est modifiée
- Exemple :

```
public static readonly DependencyProperty OrderCountProperty =  
    DependencyProperty.Register("OrderCount", typeof(int),  
    typeof(Window1));
```

Mise en place du DataBinding : XAML

- Ajouter la valeur {Binding XXXX} à la propriété voulue :

```
<TextBox Visibility="{Binding ElementName =  
MonComposant, Path=Visibility}" />
```
- Liaison de deux paramètres de l'IHM :
 - Utilisation de la propriété "ElementName" pour référencer l'élément source
 - Utilisation de la propriété "Path" pour référencer le chemin d'accès à la propriété source

Mise en place du DataBinding : XAML

- XAML : <TextBox IsEnabled= "{Binding Path=IsEnabled}" />
⇒ Binding entre la propriété IsEnabled de mes données et la propriété IsEnabled de l'élément TextBox
- Le DataContext :
 - Propriété de tout élément graphique pour référencer une source de données
 - Par défaut, hérite de l'élément hiérarchiquement supérieur. Sinon on peut le redéfinir.
 - Peut prendre comme valeur n'importe quel objet

Mise en place du DataBinding en C#

- Déclaration de la liaison directement en C# :
 - Création d'un objet Binding : `Binding binding = new Binding()`
 - Définition des propriétés comme vu précédemment :
 - Mode, Path...
 - Déclaration de la source : `binding.Source = monObjet`
 - Liaison sur l'élément graphique :
`myTextBox.SetBinding(TextBox.IsEnabled, binding);`

Mise en place du DataBinding en C#

- Dans la partie XAML :

```
<TextBox Name="MyBox" />
```

- Dans le code C# :

```
public partial class Window1 : Window
{
    public Window1()
    {
        InitializeComponent();

        string s = "Exemple de texte";
        Binding ma_box = new Binding("");
        ma_box.Mode=BindingMode.OneTime;
        ma_box.Source = s;

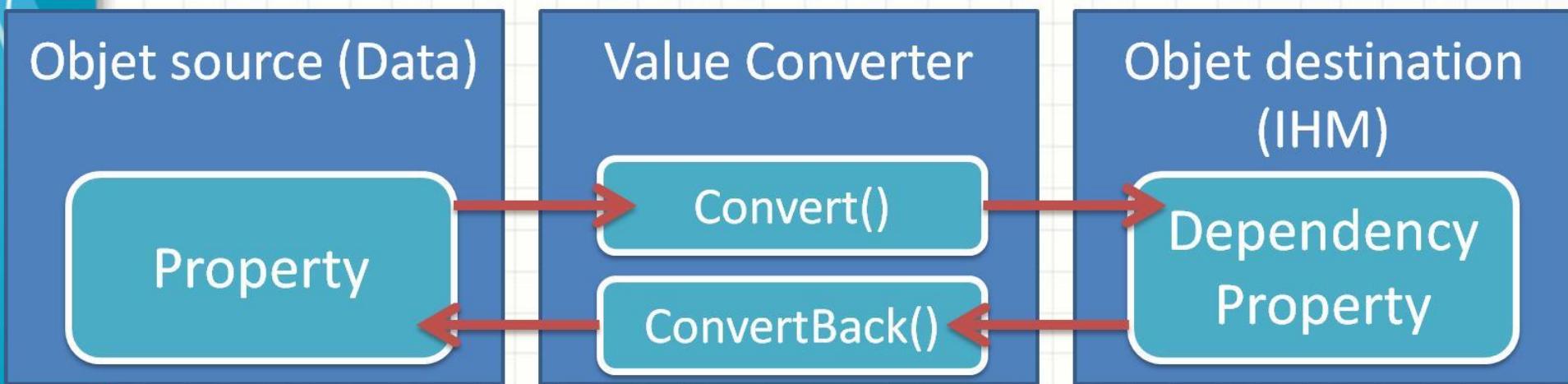
        MyBox.SetBinding(TextBox.TextProperty, ma_box);
    }
}
```

- En direct dans Visual => le .sln

@DataBinding : Les convertisseurs

- Comment lier deux propriétés de type différent ?
- Le moteur WPF est capable de convertir beaucoup de types implicitement.
 - Ex : Un Brush en Color
- Sinon, création d'un convertisseur : une classe qui implémente l'interface `IValueConverter`
 - Méthode `Convert` : convertit un type source vers la destination (View/IHM)
 - Méthode `ConvertBack` : ...

@DataBinding : Les convertisseurs



@DataBinding : Les convertisseurs

- Utilisation de l'interface :

- Ajouter un espace de noms en XAML :

```
xmlns:Converters="clr-namespace:MonNamespace.MaClasse;  
assembly=MonAssembly"
```

- Déclaration du convertisseur dans les ressources de l'IHM

```
<Window.Resources>  
    <Converters:MaClasse x:Key « MyConverter »/>  
</Window.Resources>
```

- Utilisation :

```
IsEnabled="{Binding Path=isEnabled, Converter={StaticResource  
MyConverter}}"
```

Mise à jour de la source

- Méthode 1 : Ajouter un événement // PAS MVVM
 - <Button Content="MonBouton" Click="monBtn_Click"/>
 - Private void monBtn_Click(...)
{ myCheckBox.IsEnabled= myData.IsEnabled; }
- Méthode 2 : Avec le binding sur les propriétés CLR
 - Non automatique, il faut ajouter des évènements d'information de mise à jour : l'interface
INotifyPropertyChanged

Mise à jour de la source

```
public event PropertyChangedEventHandler PropertyChanged;  
public void NotifyPropertyChanged (propertyName)  
{  
    if (PropertyChanged != null)  
    {  
        PropertyChanged (  
            this,  
            new PropertyChangedEventArgs ( propertyName )  
        );  
    }  
}  
-> NotifyPropertyChanged( "MaPropriete");
```



La présentation des données dans les composants graphiques

FORMATER LES DONNÉES

Pourquoi utiliser des styles ?

- On écrit souvent la même chose !
- Objectifs :
 - Factoriser ces éléments
 - Simplifier les mises à jour
 - Externaliser le style
 - Développement séparé (Designer / Développeur)
- Styles WPF similaires aux styles CSS

Style : Définition en XAML

- Tout contrôle possède l'attribut "Style"
- Propriétés :
 - "x:Key" : Nom de la ressources statique
 - "TargetType" : Composant cible (optionnel)
 - "BasedOn" : Surcharge d'un style d'un type de base
- La clé et le type cible permettent de définir la portée du style

Style : Définition en XAML

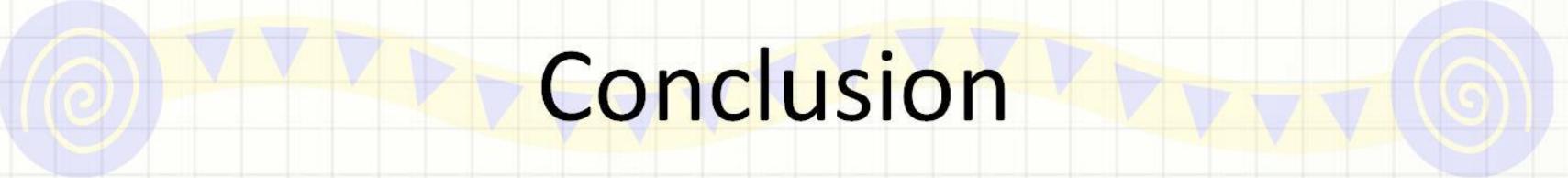
- Pour définir un élément de style : utiliser la classe "Setter" qui contient les propriétés :
 - "Property" : Nom de la propriété impactée
 - "Value" : Valeur associée
- Possibilité d'ajouter des déclencheurs "Trigger" pour définir un style sur un événement
 - Utilise les mêmes propriétés pour le déclenchement
 - Sur les propriétés, les données, les événements
- Exemples binding : 1 puis 2 puis 3 => Le .sln

Redéfinition du design de Control

- Les "ControlTemplate" permettent de redéfinir le visuel du graphique sans changer le comportement
- Surcharge de l'arbre visuel (et non logique)
- Placer le ControlTemplate dans les ressources
- Insérer les éléments graphiques dans le CT, ainsi qu'un "ContentPresenter" permettant de placer le contenu défini par l'utilisateur

Redéfinition du Template d'un élément

- Possibilité de surcharger le Template visuel du contenu d'un élément (Visual Layout)
 - En appliquant un DataTemplate sur le contenu de l'élément
 - Ajout d'une liste de contrôles
 - Ajouter les données par Binding



Conclusion

- Le Xaml, le WinForm.
- La différence sur un exemple d'une affreuse fenêtre
- La première / La seconde
- Bruissement, fin du WPF, arrivé et fin de WinRT (Windows 8.1). Rien ne semble joué à ce jour... bien au contraire (Qui connaît WPF, sait à 80% faire du WinRT)
- W10 => UWP : <https://msdn.microsoft.com/en-us/magazine/dn973012.aspx> &
<https://msdn.microsoft.com/en-us/library/windows/apps/dn958439.aspx>