

```
<Button Content="Prof-MTI">  
  <Button.Triggers>  
    <EventTrigger RoutedEvent="Button.Click">  
      <EventTrigger.Actions>  
        <BeginStoryboard Storyboard="{StaticResource StudentLearning}"/>  
      </EventTrigger>  
    </EventTrigger>  
  </Button.Triggers>  
</Button>
```



COURS WPF

PART 1

Lemettre Arnaud

Arnaud.lemettre@gmail.com

SOMMAIRE



- Présentation du cours
- Un peu de théorie
- Les principes de base
- Le Binding
- Les styles
- 2D – pixel shader

PRÉSENTATION



- Permettre d'appréhender la technologie RDA de Microsoft ainsi que l'environnement technologique lié à cette technologie.
- Etudier les nouveaux principes architecturaux des applications RIA/RDA



PRÉSENTATION



- Au programme :
 - Les bases
 - La 2D
 - Les bindings
 - Les styles
 - Les commandes
 - Le MVVM (design pattern)
 - La 3D
 - L'utilisation des ressources
 - Les animations
 - Sketchflow
 - Blend 4.0
 - RxFramework
 - ...



PRÉSENTATION



- L'évaluation :
 - Réalisation d'un projet sur une interface WPF, devant intégrer un ensemble de fonctionnalités délivrées par des services WCF qui seront mis à votre disposition.
- Notation :
 - Visualisation sur l'interface de l'ensemble des services, filtrage des données, animation, MVVM, DataBinding, Ressources.
 - Bonus : 3D, pixel shader, storyboard sous sketchflow



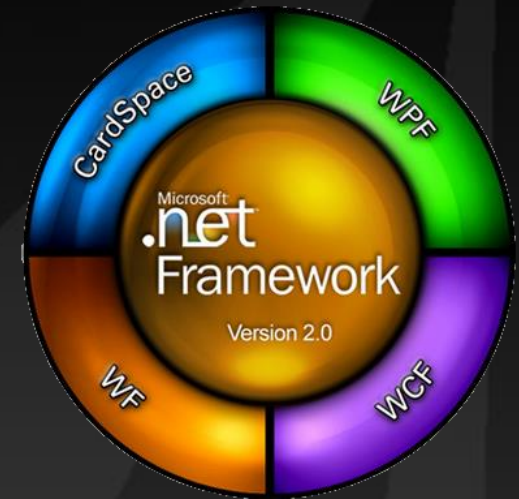
THÉORIE



THÉORIE



WPF est apparu avec le framework 3.0 de .NET.
Cette version du framework a également apporté
WCF, WF, CardSpace.



Lors du 1^{er} semestre, nous avons déjà rencontré
WCF et WF ayant respectivement pour
fonctionnalités les communications et le moteur
de workflow.

THÉORIE



WPF quand à lui propose de faire évoluer les interfaces qui auparavant étaient réalisées en Winform, en proposant un plus large panel de fonctionnalités.

A screenshot of a WPF application window titled 'Add Contact - Mr Cameron Shaw'. The window contains a form with various input fields and controls. The form is organized into sections: 'Personal Info' (Title, First Name, Middle, Surname, Occupation, Organisation), 'Physical Address' (Address, Town/Suburb, State, Postcode), 'Mailing Address' (with a 'Same Mailing Address' checkbox), 'Home/Phone' (Home Phone, Work Phone, Email Address, Date of Birth), and 'Categories (Optional)' (a list box with 'Sales Rep', 'Supplier', and 'Supporter'). The form is styled with a light beige background and blue borders.

Cela n'est pas la seule avancée, car WPF permet également d'avoir des interfaces beaucoup plus ergonomiques et riches d'où le terme de Rich Desktop Application (RDA)

THÉORIE



Pour travailler sur l'ergonomie des applications ou sur le rendu visuel, il n'est pas rare de faire appel à des designers en entreprise. Avant le WPF, ils ne pouvaient proposer que des ébauches des interfaces, mais depuis l'apparition du WPF, nous pouvons maintenant intégrer directement dans les équipes le designer car son travail peut être intégré en même temps que le développement de l'application.

THÉORIE



Ceci illustre un des principaux avantages du WPF à savoir :

la séparation du code de l'application (.CS) et de l'interface.

THÉORIE

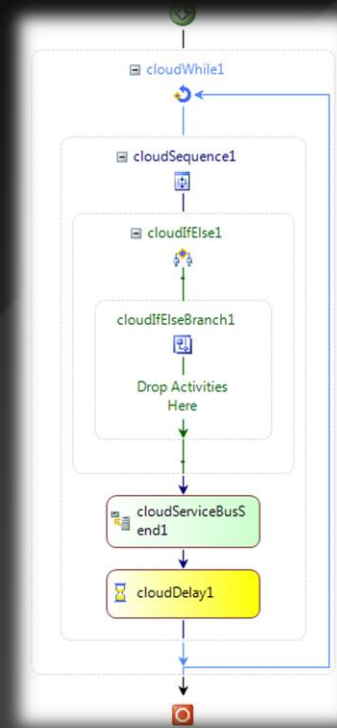


WPF est depuis Vista directement intégré dans le noyau du système d'exploitation. Bien que toutes les interfaces de Windows ne sont pas encore réalisées en WPF (à cause des performances, utilisation de Direct X) beaucoup de logiciels intègrent la technologie (ex : Visual Studio 2010, ...). Cependant une application WPF pourra parfaitement fonctionner sur un Windows XP.

THÉORIE



On peut également observer cette fusion des technologies avec le designer de workflow qui utilise le « langage de WPF », pour décrire les modèles de workflow.



THÉORIE



Un peu plus de concret ...

Les fichiers WPF se reconnaissent par leur extension « .xaml ». Ce nom dérive de XML car le langage utilisé au sein du fichier est proche d'une syntaxe XML. Et signifie : Extensible Application Markup Language



THÉORIE



```
<Window x:Class="WpfTP.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Grid>

  </Grid>
</Window>
```



- Quel genre d'applications peut-on réaliser ?
 - Bien entendu des applications Windows
 - Mais également des applications qui peuvent être lancées dans un navigateur (XBAP) en indiquant simplement un chemin réseau. (Fonctionne avec internet explorer, mais depuis la version 3.5 avec FireFox également).

Très peu utilisé, voir quasiment jamais.
 - Navigation Application, pareil que les XBAPs du même type qu'un site web, mais s'exécute en local

THÉORIE



- Pour l'instant, tout ceci se limite à windows. Mais avec nos amis qui développent le portage Mono (nom de code projet Olive) on peut bénéficier de certains avantages de WPF sous mono. Malheureusement ce sous module ne semble plus actif depuis quelque temps.



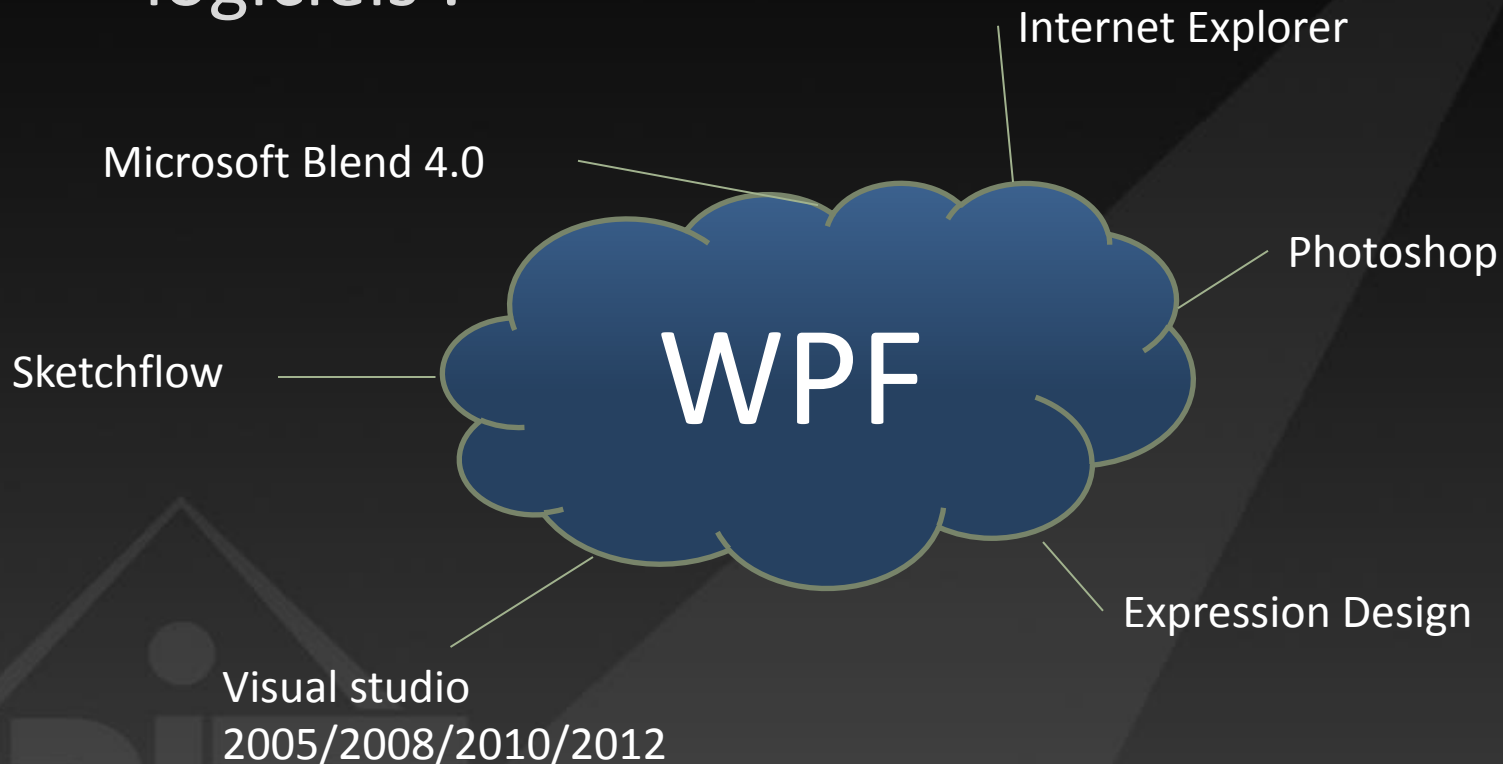
mono™



THÉORIE



- Tout autour de WPF gravitent plusieurs logiciels :





LES BASES

LES BASES

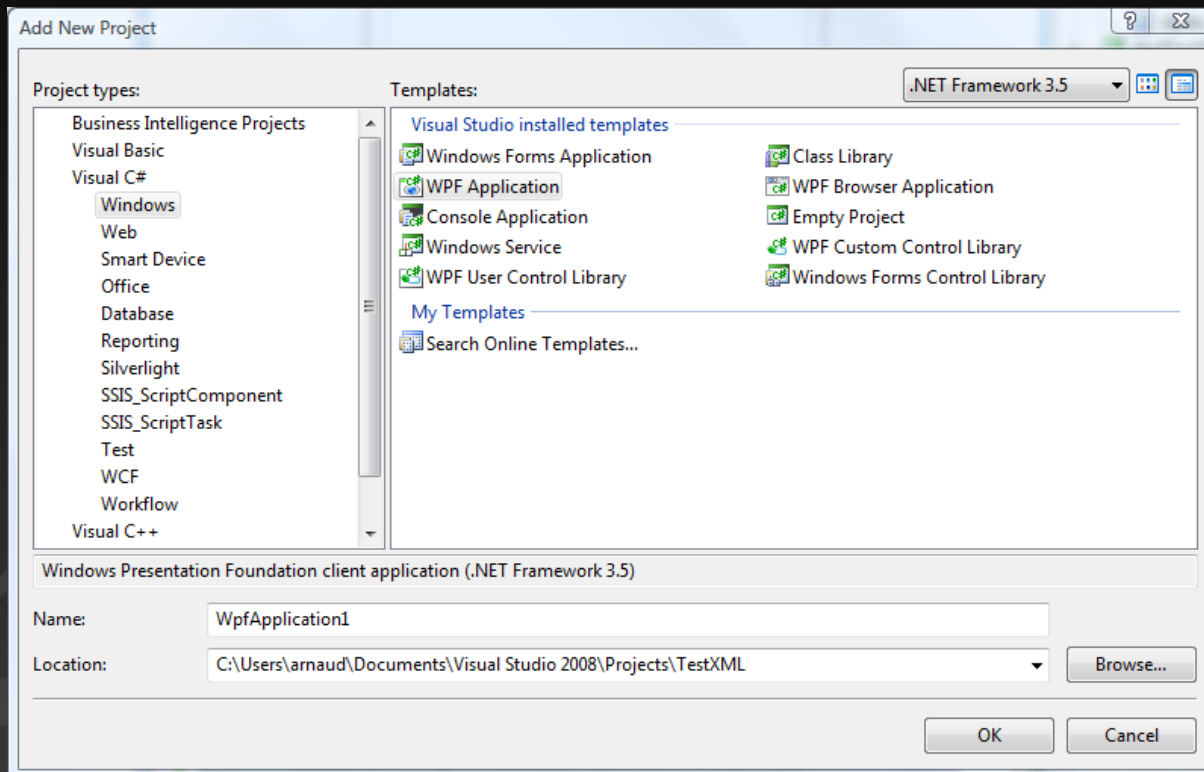


- Dans un 1^{er} temps, nous allons explorer les concepts de base, afin de pouvoir réaliser une application WPF.

LES BASES



- Ouvrir Visual Studio, sélectionner un projet de type C# et WPF Application.



LES BASES



Code de base pour une fenêtre WPF

Nom de la classe du fichier CS

```
<Window x:Class="WpfTP.Window1"
  xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Window1" Height="300" Width="300">
  <Grid>

  </Grid>
</Window>
```

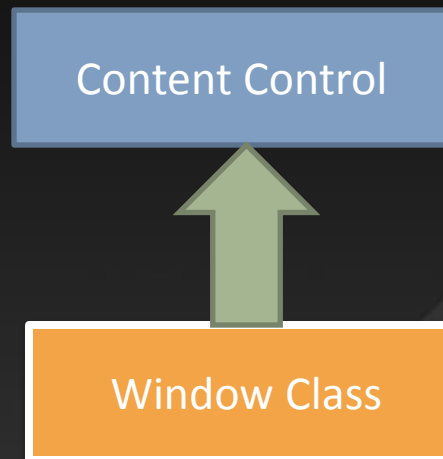
Les xmlns
correspond
aux « using »

LES BASES



- Pourquoi l'élément « window » ne peut avoir qu'un seul fils ?

Ceci s'explique par l'architecture logicielle de ce composant

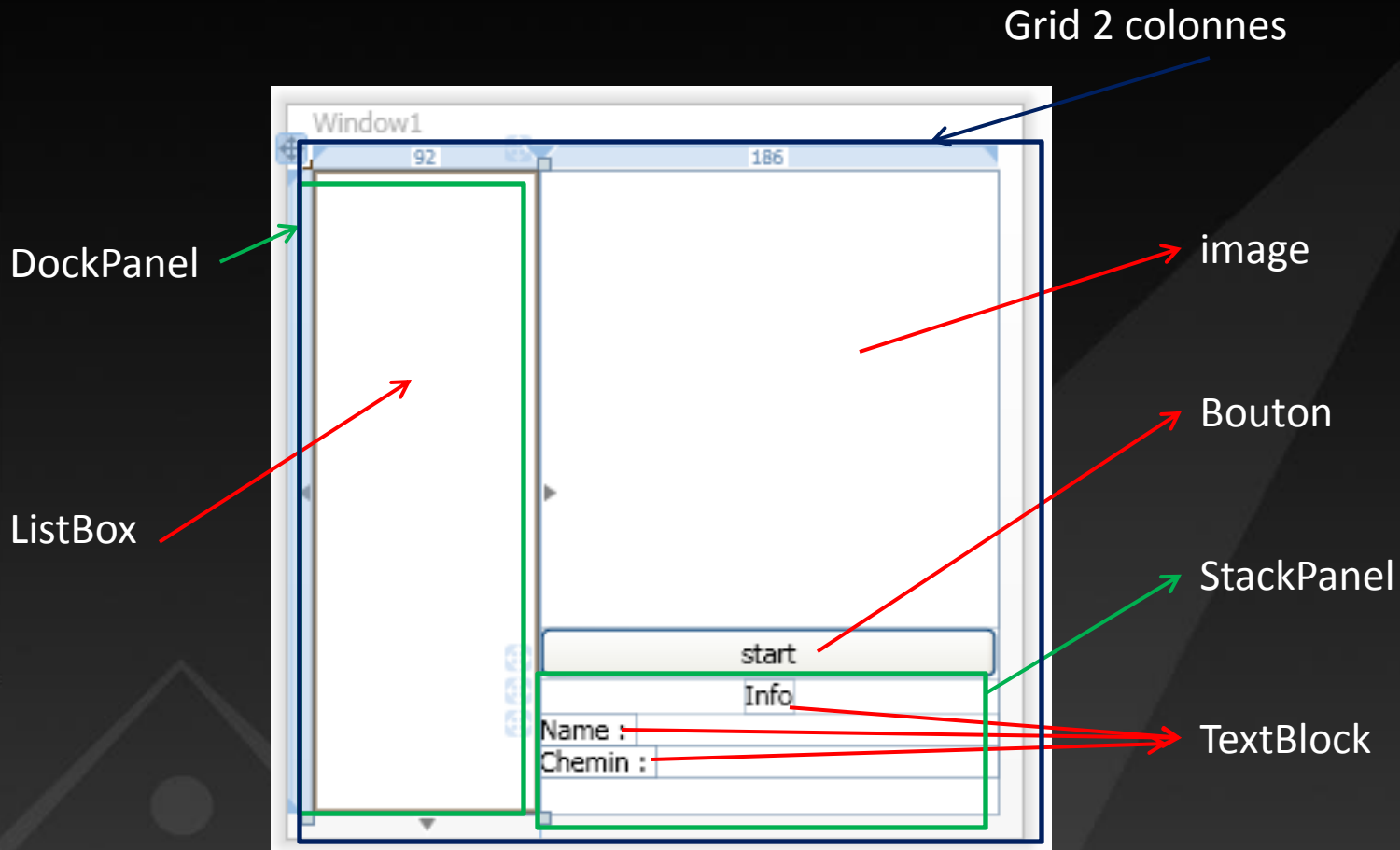


→ Classe mère ne permettant de n'avoir qu'un seul fils

LES BASES



- Voici un exemple d'Interface à réaliser :



LES BASES



- Pour réaliser cette interface, 2 méthodes :
 - Drag & drop
 - Directement par le code

LES BASES



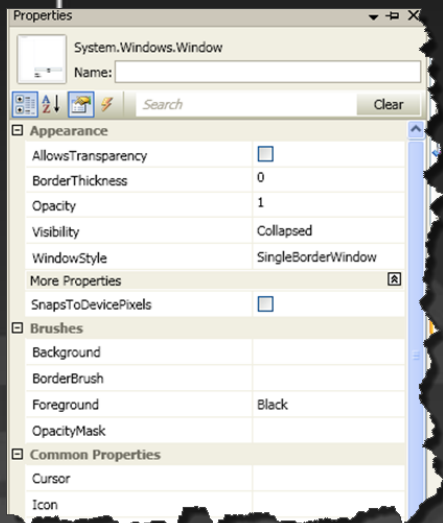
```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="92*" />
    <ColumnDefinition Width="186*" />
  </Grid.ColumnDefinitions>
  <DockPanel Grid.Column="0">
    <ListBox DockPanel.Dock="Top"></ListBox>
  </DockPanel>
  <StackPanel Grid.Column="1">
    <Image></Image>
    <Button>start</Button>
    <StackPanel>
    </StackPanel>
  </StackPanel>
</Grid>
```

Exemple de Code
XAML généré

LES BASES



- Un peu de détail :
 - Pour le window, il y a plusieurs paramètres possibles. (Title, largeur, hauteur, ...) Ces paramètres sont visibles dans la partie properties de l'IDE
 - De base, le conteneur est une grid, une grid permet de pouvoir organiser les composants en créant des lignes et des colonnes.



LES BASES



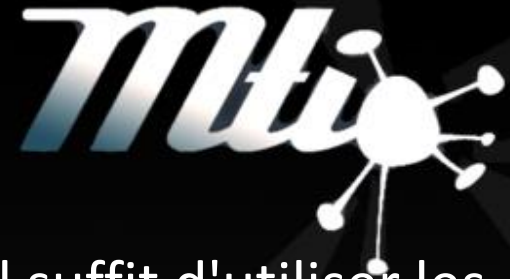
Permet de créer une grille de ce type :


```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="92*" />
    <ColumnDefinition Width="186*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="100" />
  </Grid.RowDefinitions>
</Grid>
```

Pour indiquer les hauteurs et largeurs :

- Directement la valeur => taille fixe
- En utilisant *, ceci à pour but de faire réduire ou agrandir la taille de manière proportionnelle par exemple si on a 10* et 20* alors une taille sera toujours le double de l'autre.

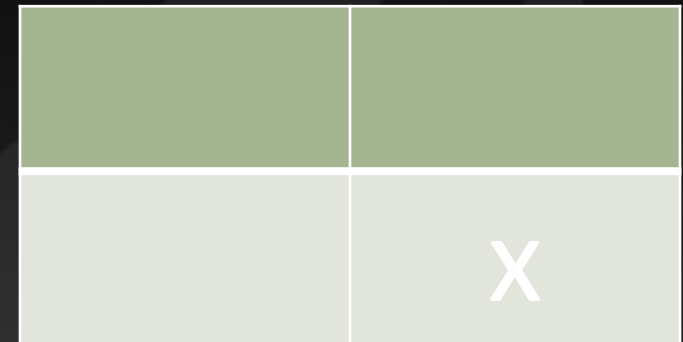
LES BASES



Pour indiquer l'endroit où situer le composant il suffit d'utiliser les propriétés de la grille. Les index commencent à 0

```
Grid.Column="1"
```

```
Grid.Row="1"
```

A diagram of a 2x2 grid. The top row consists of two olive-green cells. The bottom row consists of two light beige cells. The cell at the bottom-right (row 1, column 1) contains a white 'X' character. A large curly brace on the left side of the grid groups the two rows together.

	X

LES BASES



2 autres propriétés sont utiles :

- ColumnSpan
- RowSpan

Elles sont de la taille maximum du nombre de colonnes et du nombres de lignes. Cela permet de positionner un élément sur 2 colonnes/lignes

`Grid.Column="1" Grid.ColumnSpan="2"`

	X	

LES BASES



Les stackpanels sont eux aussi des conteneurs, mais on ne peut pas spécifier de colonne ni de ligne. Ils sont utiles pour regrouper des composants suivant une orientation et peuvent contenir plusieurs fils

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

Les orientations peuvent être soit « Horizontales » ou « Verticales »

LES BASES



Les DockPanels sont eux aussi des conteneurs qui permettent de positionner les composants fils de manière fixe sur une position. Par rapport à ce DockPanel.

```
<DockPanel Grid.Column="0">  
    <ListBox  
        DockPanel.Dock="Top"></ListBox>  
</DockPanel>
```

La position s'indique bien entendu sur le composant fils et peut prendre comme valeur :

- Top
- Left
- Right
- Bottom

LES BASES



Le composant image permet de mettre une image en indiquant le chemin dans la source.

```
<Image Source="~/image.png"></Image>
```



Par le code, on ne peut pas modifier la source en indiquant une nouvelle URL, il faut rajouter un peu de code pour que la liaison se fasse.

LES BASES



Une différence majeure par rapport aux composants Winforms, ceux de WPF pour la plupart peuvent contenir d'autres composants. Ainsi un bouton peut contenir une checkbox, ou alors un texte peut également contenir une vidéo

....

```
<Button x:Name="button1">
    <StackPanel
        Orientation="Horizontal">
        <CheckBox
            IsChecked="False" />
        <TextBlock Text="Start"/>
    </StackPanel>
</Button>
```

LES BASES

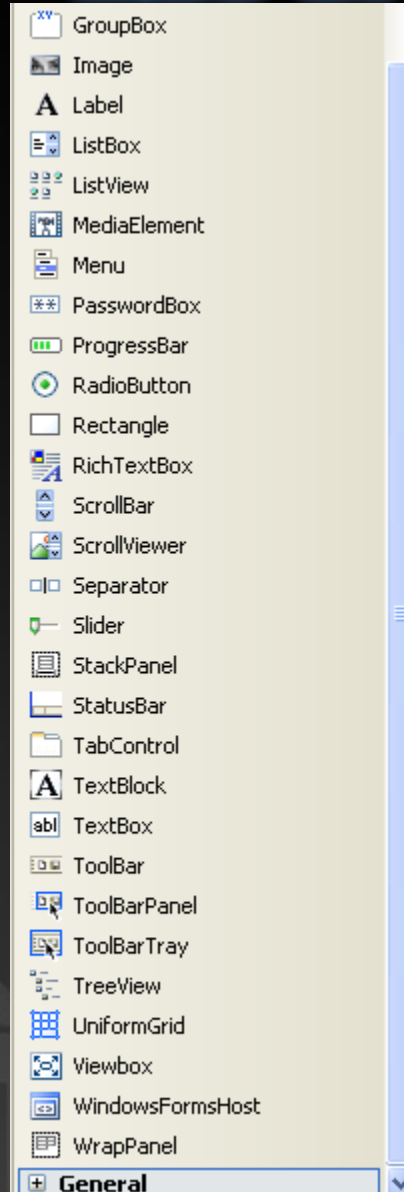
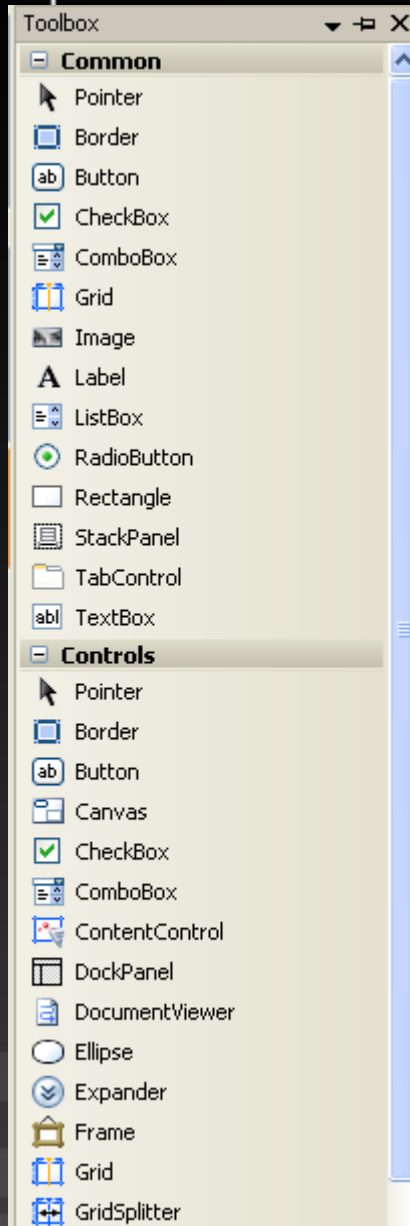


Pour rajouter une action lors du clic sur le bouton :

```
<Image ></Image>  <New Event Handler>  
<Button Click="">start</Button>
```

```
<Button Click="Button_Click">start</Button>
```

LES BASES



- WPF est très riche en composants, mais le principe de fonctionnement reste le même pour tous.



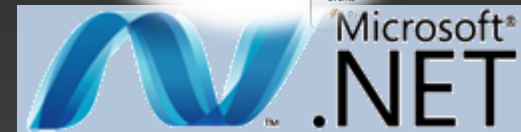
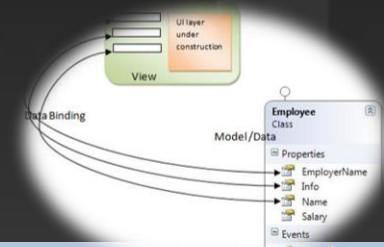
LE BINDING



LE BINDING



- **Définition** : Le DataBinding décrit le processus d'une création de dépendance entre la valeur d'une propriété appelée « target property » et une autre propriété appelée « source property »
- Il en existe plusieurs types :
 - Binding composant – composant (ex : label et slider)
 - Binding composant – objet (ex : liste)
 - Binding composant – fichier (ex: xml)



LE BINDING



- Généralement, on le déclare directement dans le XAML, mais cela impose que cette opération soit réalisée avant la compilation.
- Il existe une possibilité de le définir par le code pour le rendre dynamique, mais c'est beaucoup plus lourd en terme d'écriture.

LE BINDING



- Il existe également plusieurs modes de binding selon les utilisations que l'on souhaite en faire :
 - OneWay

Le binding n'est mis à jour que lorsque la source change de valeur, si on change la destination alors cela n'impacte pas la source
 - TwoWay

Si on change la propriété source ou de destination alors l'autre propriété est automatiquement mis à jour

LE BINDING



- OneTime

Le binding de la destination n'est mis à jour qu'au démarrage de l'application, ou au changement de DataContext

- OneWayToSource

Le binding source est mis à jour lorsque le binding de destination change, le fait de changer la source n'influence pas le binding de destination

LE BINDING



- Supposons que nous avons cette classe :

```
public class PictureInfo
{
    #region variables
    private string _path;
    private string _name;
    #endregion

    [getter / setter]
}
```

LE BINDING



Nous avons ensuite une fonction qui nous renvoie une liste d'objets de type `PictureInfo`.

Pour pouvoir indiquer à notre listbox que nous souhaitons afficher ces informations, il faut lui passer dans son contexte :

```
listPicture.DataContext = list;
```

LE BINDING



La liste est composée d'objets , et une listbox ne sait pas quoi afficher dans cet objet. Le cas d'une liste est un peu particulier car on peut utiliser la propriété DisplayMemberPath. Pour donner un autre exemple nous utiliserons ici un DataTemplate. Les DataTemplates permettent de décrire la façon dont sera affiché l'objet.

```
<Window.Resources>
  <DataTemplate x:Key="templateList">
    <TextBlock Text="{Binding Path=Name}"/>
  </DataTemplate>
</Window.Resources>
```

Nom du
template
pour l'utiliser
dans le code

Nom de la propriété à afficher

LE BINDING



Pour spécifier quel template utiliser pour la liste :

```
<ListBox x:Name="listPicture" ItemsSource="{Binding}"  
        ItemTemplate="{StaticResource templateList}" />
```


LE BINDING



Pour illustrer un binding composant :

Nom du composant

Propriété à binder

```
<StackPanel DataContext="{Binding ElementName=listPicture, Path=SelectedItem}">
  <StackPanel Orientation="Horizontal" >
    <TextBlock Text="{Binding Path=Name}"/>
  </StackPanel>
  <StackPanel Orientation="Horizontal" >
    <TextBlock Text="{Binding Path=Path}"/>
  </StackPanel>
</StackPanel>
```

LE BINDING



Une application riche doit rester réactive en toutes circonstances. Pour cela on peut différer le binding.

```
<ListBox Grid.Row="1" x:Name="listRes">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Name}"/>
        <TextBlock Text="{Binding Res, IsAsync=True,
          FallbackValue=Compute...}"/>
      </StackPanel>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Permet d'activer le chargement asynchrone

Valeur qui sera affichée tant que la valeur du binding ne sera pas prête à être affichée

Le chargement asynchrone ne monopolise pas le thread UI mais utilise un second thread pour le calcul de la valeur

LE BINDING



- Ceci n'est qu'une présentation sommaire de ce que l'on peut faire avec le binding.
- On peut réaliser un binding directement sur un fichier XML ou encore sur des objets ADO.NET

LE BINDING



- Une autre partie à mi chemin entre les listes (objet) et le binding permet d'effectuer certaines actions sur l'affichage.
- Pour cela il faut regarder la classe :

[ICollectionView](#)

On peut réaliser des filtres, des tris, naviguer dans une liste



LES STYLES



LES STYLES



- Les styles sont analogues aux fichiers CSS de l'HTML.
- Cela permet de substituer l'apparence standard par une autre.
- Le changement peut se faire soit de manière statique (avant la compilation) ou de manière dynamique par le code

LES STYLES



- Les styles peuvent être appliqués à tous les composants WPF.
- Dans cet exemple on va rajouter un style sur le bouton, en lui ajoutant un fond rouge.

LES STYLES

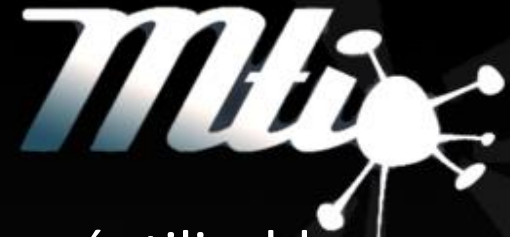


Code permettant de mettre un style directement sur le bouton.

```
<Button Click="Button_Click">
    <Button.Style>
        <Style>
            <Setter
Property="Button.Background" Value="Red"/>
        </Style>
    </Button.Style>
    start
</Button>
```

Le mot clé SETTER est très important car c'est grâce à lui que l'on peut changer les valeurs. Property doit être de la forme Element.PropertyName

LES STYLES



Mettre un style dans un composant, cela n'est pas réutilisable car Il faut le recopier pour tous les autres composants. Pour cela on peut mettre directement en ressource le style. Dans Window.Resources :

```
<Style TargetType="Button">  
  <Setter Property="Button.Background" Value="Red"/>  
</Style>
```

TargetType dans un window.Resource, permet d'appliquer le style à tous les composants fils, sans avoir besoin de le spécifier. Pour éviter qu'un bouton ne prenne le style par défaut on peut lui mettre comme paramètre :

```
<Button Click="Button_Click" Style="{x:Null}">
```

LES STYLES



Les styles ne s'appliquent pas uniquement sur l'aspect visuel mais aussi sur les événements qui peuvent être attachés aux styles.

```
<Style.Triggers>
  <Trigger Property="Button.IsMouseOver" Value="True">
    <Setter Property="Button.FontWeight" Value="Bold"/>
  </Trigger>
</Style.Triggers>
```

LES STYLES



Cependant on peut définir des styles pour les appliquer uniquement sur certains composants. Dans les ressources :

```
<Style x:Key="styleButton"/>
```

Permet de pouvoir utiliser le style, dans la suite de l'arbre XAML

Pour utiliser le style il suffit de rajouter ce code sur le bouton :

```
<Button Click="Button_Click" Style="{StaticResource styleButton}">
```

LES STYLES



On peut également hériter des styles que l'on a déjà conçus, ainsi on gagne du temps sur le développement.

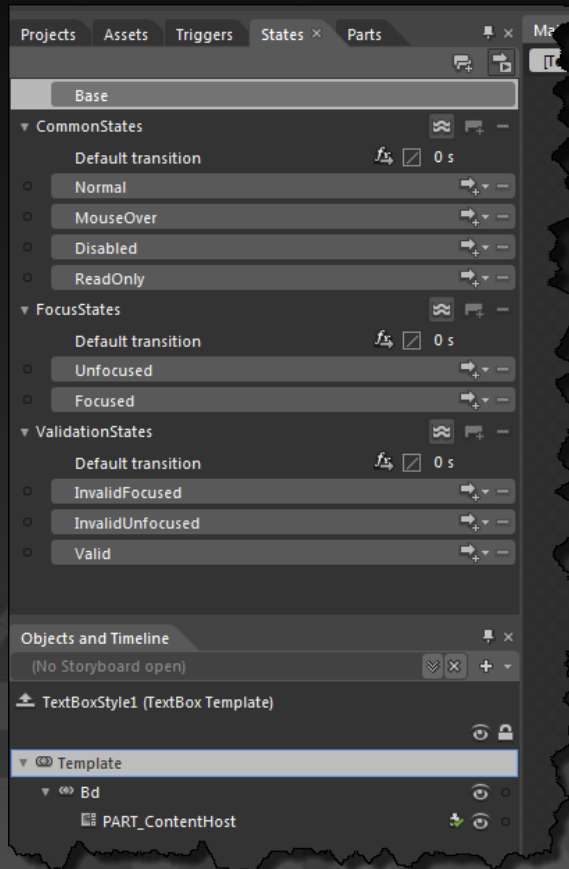
```
<Style BasedOn="{StaticResource styleButton}" x:Key="styleButtonHerited">
```

LES STYLES



Depuis WPF 4.0, les visuels states managers sont apparus. Ils proviennent à l'origine de silverlight.

Ils permettent de styliser un composant en fonction de son état.



2 façons de définir les états :

- En partant d'un model sous blend
- En le faisant soi-même

Pourquoi faire des états ?

- Sur un formulaire (champs en édition ou lecture)
- Mettre en valeur un composant selon une action
- Changer d'état un composant selon une logique métier



LES STYLES



Cas pratique, le formulaire en lecture / édition

L'ensemble des états sont définis dans des groupes

Les états contiennent les différents storyboard permettant de changer les apparences des composants

Ces groupes sont intégrés dans le VisualStateManager

Le but étant de mettre un champ dans un textblock puis de mettre le contenu dans un textbox. Cette modification peut être gérée par un bouton

A screenshot of a WPF application window titled 'MainWindow'. The window contains a form with two main sections. The top section has a label 'Name :' followed by a text box containing the text 'arnaud'. The bottom section has a button labeled 'Update'. Red arrows point from the text box and the 'Update' button to descriptive text on the right.

Zone qui sera en modification

Bouton manipulant l'état

LES STYLES



```
<TextBlock Text="Name :"/>
<TextBlock x:Name="textBlock" Grid.Column="1" Text="{Binding ElementName=textBox, Path=Text}" />
<TextBox x:Name="textBox" Grid.Column="1" Text="arnaud"/>
<ToggleButton x:Name="toggleButton" Grid.Row="1" Content="Update"/>
```

Contenu de la grille

Pour l'exemple on se bind sur le textbox, dans un exemple concret on effectuerait le bind sur la propriété d'un objet

LES STYLES



Nom du groupe

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup x:Name="Mode">
    <VisualState x:Name="Read">
      <Storyboard>
        <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)"
                                       Storyboard.TargetName="textBox">
          <DiscreteObjectKeyFrame KeyTime="0" Value="{x:Static Visibility.Collapsed}"/>
        </ObjectAnimationUsingKeyFrames>
      </Storyboard>
    </VisualState>
    <VisualState x:Name="Write">
      <Storyboard...>
    </VisualState>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Animation sur les
éléments de
l'interface

Nom des états

LES STYLES



Pour changer de state on peut utiliser directement en code behind :

```
VisualStateManager.GoToState(nameTextblock, "Read", true);
```

Les problèmes de cette méthode réside dans les faits suivants :

- Nous devons disposer de la référence sur l'élément sur lequel modifier l'état
- Le code doit être dans le fichier associé à la vue

On peut cependant respecter le modèle MVVM en utilisant les **DataStateBehavior**

LES STYLES



Rajout des namespaces, penser également à rajouter les DLL :

- System.Windows.Interactivity.dll
- Microsoft.Expression.Interactions.dll

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:Microsoft
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
x:Class="WpfRx.MainWindow"
Title="MainWindow" Height="350" Width="525" FontSize="36">
  <Grid>
    <VisualStateManager.VisualStateGroups>
      <VisualStateGroup x:Name="Mod" />
    </VisualStateManager.VisualStateGroups>
    </Grid>
    <i:Interaction.Behaviors>
      <ei:DataStateBehavior Binding="{Binding IsChecked, ElementName=toggleButton}" Value="True" TrueState="Write" FalseState="Read"/>
    </i:Interaction.Behaviors>
    <TextBox x:Name="textBox" Grid.Column="1" />
    <ToggleButton x:Name="toggleButton" Grid.Row="1" Content="Update"/>
  </Grid>
</Window>
```

Binding vers l'élément où l'on souhaite récupérer la valeur

La valeur de la condition

Les états



2D ET IMAGES



2D ET IMAGES



- En WPF tout est vectoriel !
- Aussi bien les images que ... les composants
- De ce fait, on peut effectuer plusieurs effets grâce par exemple aux pixels shaders.

2D ET IMAGES



Premier exemple, faire un radian, bien entendu ici on l'applique sur une grid mais comme vous l'avez compris on peut le faire sur tous les composants :

```
<Grid>
  <Grid.Background>
    <LinearGradientBrush>
      <GradientStop Color="Black" Offset="0"/>
      <GradientStop Color="White" Offset="1"/>
    </LinearGradientBrush>
  </Grid.Background>
</Grid>
```

Pour les offset 0 correspond au coin en haut à gauche
1 correspond au coin en bas à droite

Il existe également des gradients de type radial

PS : tout ce qu'on fait directement en code peut être fait sous blend en 2 clics mais on aborde cela dans un prochain cours.

2D ET IMAGES



On peut également personnaliser les composants grâce aux objets géométrie :

```
<Button Height="100" Width="100" Content="click">  
    <Button.Clip>  
        <EllipseGeometry Center="50,50" RadiusX="50"  
RadiusY="50"/>  
    </Button.Clip>  
</Button>
```

2D ET IMAGES



- Dans cette partie, nous aborderons rapidement les pixel shader.
- Un shader (anglais, du verbe to shade : ombrager ou estomper, nuancer) est un programme utilisé en image de synthèse pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel. Ils peuvent permettre de décrire l'absorption et la diffusion de la lumière, la texture à utiliser, les réflexions et réfractions, l'ombrage, le déplacement de primitives et des effets post-traitement.

2D ET IMAGES



- En WPF on peut réaliser ses shaders soi même ce qui est normal vu que WPF s'appuie sur la carte graphique. Cependant c'est assez long.
- Heureusement, certaines personnes nous offrent des bibliothèques de shaders toutes faites, comme ici :
- <http://www.codeplex.com/wpffx>

2D ET IMAGES



Pour utiliser la bibliothèque, inclure dans les références ShaderEffectLibrary.

Puis dans le code :

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    ShaderEffectLibrary.RippleEffect effect = new
    ShaderEffectLibrary.RippleEffect();

    effect.Amplitude = 10;
    button1.Effect = effect;
}
```

2D ET IMAGES



- Pour les transitions, c'est le même principe sauf que ces shaders ne s'appliquent qu'à des brush.



QUESTIONS ?