



Optez pour une transformation digitale intelligente

L'INTELLIGENCE ARTIFICIELLE AU SERVICE DE VOTRE BUSINESS.



WPF

SOMMAIRE

- 1 – Introduction : Qu'est ce que WPF
- 2 - Layout et Panels
- 3 - Content Controls et Items Controls
- 4 - Styles - Skins - Thèmes.
- 5 - Properties - Data Binding
- 6 - Templates - Triggers
- 7 - MVVM.

I - INTRODUCTION

Qu'est ce que WPF ?

Windows Presentation Foundation (WPF) est un framework d'interface utilisateur graphique (GUI) développé par Microsoft. Il a été introduit pour la première fois dans le .NET Framework 3.0 et est depuis devenu un élément essentiel du système d'exploitation Windows.

Qu'est ce que WPF ?

WPF est spécial pour plusieurs raisons :

1. **Interface utilisateur riche** : WPF fournit un ensemble complet de contrôles et d'outils pour créer des interfaces utilisateur visuellement attrayantes. Il prend en charge des fonctionnalités graphiques, d'animation et multimédias avancées qui peuvent être utilisées pour créer des applications attrayantes et interactives.
2. **XAML** : WPF utilise un langage de balisage basé sur XML appelé XAML (eXtensible Application Markup Language) pour définir l'interface utilisateur et le comportement d'une application. Cela permet de séparer facilement la conception d'une application de sa logique et permet aux concepteurs et aux développeurs de travailler ensemble plus efficacement.

Qu'est ce que WPF ?

WPF est spécial pour plusieurs raisons :

3. **Liaison de données** : WPF fournit de puissantes fonctionnalités de liaison de données qui permettent aux développeurs de connecter facilement des données provenant de différentes sources à l'interface utilisateur. Il est ainsi plus facile de créer des applications dynamiques et axées sur les données.
4. **Évolutivité** : WPF est conçu pour être évolutif, il peut donc être utilisé pour créer des applications qui fonctionnent bien sur différents appareils et tailles d'écran. Cela en fait un bon choix pour le développement d'applications de bureau, mobiles et Web.

Pourquoi WPF?

Windows Forms

1. Rendu basé sur GDI et GDI+
2. Absence de prise en charge de l'accélération matérielle
3. Difficile à utiliser pour l'animation matérielle
4. Incapable d'exploiter la puissance des cartes graphiques modernes

WPF

1. Intégré
2. Indépendance de la résolution
3. Séparation des préoccupations relatives à la programmation et à la conception
4. Des possibilités infinies de personnalisation
5. Simplicité de style

Pourquoi WPF?

Windows Forms

Utilise la technologie WinForms, qui est basée sur les API Windows natives. Cela signifie qu'elle est fortement liée au modèle de fenêtrage Windows classique.

Offre une interface utilisateur basée sur les contrôles classiques de Windows. Les graphismes sont souvent moins flexibles et les contrôles ont un aspect plus traditionnel.

WPF

Utilise la technologie DirectX pour le rendu graphique et repose sur XAML (eXtensible Application Markup Language) pour la définition de l'interface utilisateur. Cela offre une plus grande flexibilité en termes de conception et de graphismes.

Technologie Sous-jacente

Graphismes et Interface Utilisateur

Permet des interfaces utilisateur graphiquement riches avec des fonctionnalités telles que les graphiques vectoriels, les animations et les styles. Les possibilités de personnalisation sont bien plus étendues.

Pourquoi WPF?

Windows Forms

L'interface utilisateur est définie en utilisant le code source du langage de programmation (comme C# ou VB.NET) sans avoir besoin d'un fichier de balisage spécifique.

Le binding de données est moins puissant et plus verbeux, nécessitant souvent des mises à jour manuelles de l'interface utilisateur en fonction des changements de données.

Langage de Définition d'Interface Utilisateur

Binding de Données

WPF

Utilise XAML (eXtensible Application Markup Language), qui est un langage de balisage déclaratif pour définir l'interface utilisateur. Cela permet une séparation claire entre le design et le code.

Offre un système de binding de données robuste et flexible qui permet de lier dynamiquement les éléments de l'interface aux sources de données. Les mises à jour de l'interface sont automatiquement gérées en fonction des changements de données.

Pourquoi WPF?

Windows Forms

La personnalisation des contrôles est limitée et souvent requiert une personnalisation manuelle.

Permet une intégration aisée avec les anciens composants Win32.

Personnalisation et Styles:

Intéropérabilité

WPF

Permet une personnalisation avancée grâce à l'utilisation de styles et de templates. Cela offre une grande flexibilité pour modifier l'apparence des contrôles.

Offre des fonctionnalités d'interopérabilité avec les anciens composants Win32 et WinForms.

Pourquoi WPF?

- Il était une fois dans la technologie ...
 - Les interfaces des programmes étaient basées sur des caractères
 - Des boutons, des listes déroulantes, etc. fournissaient les aspects graphiques de l'interaction de l'utilisateur
 - Les écrans étaient des écrans d'ordinateur de bureau ou des ordinateurs portables
 - Tous avec les mêmes caractéristiques générales
 - Choix limité de résolutions

Pourquoi WPF?

Windows Forms a fourni l'environnement graphique pour les applications

- Les applications avaient toutes des caractéristiques visuelles très similaires
- L'interaction de l'utilisateur était utilitaire
- La fonctionnalité de l'application était l'objectif principal
- L'expérience utilisateur était limitée par le paradigme de programmation et le matériel de l'époque

Pourquoi WPF?

- ❖ Windows Forms (WinForms) est une bibliothèque de classes graphique
 - Utilise le moteur de rendu GDI/GDI+ basé sur des rasters
 - L'interface utilisateur était étroitement liée au code de l'application
- ❖ L'intégration d'éléments et de technologies d'interface utilisateur plus complexes a été possible
 - Extrêmement difficile

Pourquoi WPF?

Au milieu des années 2000, le monde changeait... encore

- ❑ Le matériel a continué de s'améliorer
- ❑ Les technologies Internet se sont améliorées et sont devenues une partie intégrante de la culture
- ❑ L'expérience utilisateur est devenue aussi importante que la fonctionnalité de l'application
 - Augmentation de la productivité
 - Moins de formation

Pourquoi WPF?

Windows Presentation Foundation a fourni une toute nouvelle façon de créer des interfaces utilisateur

- ❑ Basé sur DirectX
- ❑ Les interfaces utilisateur sont conçues dans un langage de balisage basé sur XML
 - XAML
- ❑ Les interfaces peuvent être beaucoup plus riches qu'auparavant

WPF?

- WPF rend possible des interfaces plus intuitives et immersives
 - Les images, le texte, les graphiques 2D et 3D, la vidéo, les annotations, etc. sont maintenant facilement intégrés dans l'interface
 - La conception de l'interface et de l'application est séparée
- WPF change la façon dont les interfaces utilisateur sont créées, gérées et expérimentées

EVOLUTION DE WPF

Windows Presentation Foundation (WPF) existe depuis étonnamment longtemps

- Introduit en 2006
- Une technologie mature
- Largement mis en œuvre
- Stable

EVOLUTION DE WPF

WPF Version 3.5

- Introduit en novembre 2007
- .NET Framework 3.5. Visual Studio 2008
- Amélioration de la liaison des données
- De meilleurs effets spéciaux, une mise en œuvre et des fonctionnalités plus efficaces
- Améliorations significatives des performances
- 3.5 SP1 est sorti en novembre 2008

EVOLUTION DE WPF

WPF Version 4.0

- Introduit en avril 2010
- .NET Framework 4.0. Visual Studio 2010
- Introduced the Calendar, DataGrid, and DatePicker controls
- Added Multi Touch and Manipulation in support of Windows 7 touchscreen capabilities

EVOLUTION DE WPF

WPF Version 4.5

- Introduit en août 2012
- .NET Framework 4.5 / Visual Studio 2012
- Ajout d'un nouveau contrôle du ruban (barre d'outils d'accès rapide, menu d'application et onglets)
- Deux versions mineures sans améliorations majeures
 - La version 4.5.1 est apparue en octobre 2013 / Visual Studio 2013
 - 4.5.2 est apparu en mai 2014

EVOLUTION DE WPF

WPF Version 4.6

- Introduit en juillet 2015
- .NET Framework 4.6 / Visual Studio 2015
- Prise en charge de la fenêtre enfant transparente
- Améliorations HOPI (High Dots Per Inch) et Touch

WPF ET WINDOWS FORMS

La plupart des développeurs jettent un coup d'œil à WPF et posent trois questions courantes

- WPF remplace-t-il les applications Windows Forms ?
- Windows Forms est-il le bon choix ?
- Quand choisirais-je l'un plutôt que l'autre ?

WPF ET WINDOWS FORMS

WPF remplace-t-il les applications Windows Forms ?

- WPF n'a pas été conçu pour remplacer immédiatement ou à court terme Windows Forms
- Windows Forms est la base de nombreuses applications d'entreprise clés et fournit toutes les fonctionnalités requises
- WPF est simplement un autre outil que les développeurs d'applications de bureau Windows peuvent utiliser

WPF ET WINDOWS FORMS

Windows Forms est-il le bon choix ?

- Il existe au moins deux situations où WPF n'est pas la meilleure option
 - L'application en cours de développement ne bénéficiera pas des fonctionnalités offertes par WPF
- La base d'utilisateurs ou l'industrie est réticente à changer à moins que le changement n'apporte des améliorations ou des gains d'efficacité significatifs

WPF ET WINDOWS FORMS

Quand choisirais-je l'un plutôt que l'autre ?

- WPF est le choix idéal pour les applications qui impliquent différents types de médias dans l'interface
 - Vidéo, documents, 3D contenus, transitions animées, etc.
- WPF est également un excellent choix lorsque l'application doit être skinnée (visuels facilement et systématiquement modifiés)

WPF ET WINDOWS FORMS

Quand choisirais-je l'un plutôt que l'autre ?

- WPF est également un excellent choix lors de la liaison à des données XML, du chargement dynamique de parties d'une interface utilisateur à partir de services Web, etc.
- Windows Forms dispose de plus de contrôles tiers, d'un support en ligne et de communautés d'utilisateurs plus nombreux, et il est plus facile de localiser et d'embaucher des développeurs Win Forms expérimentés

PROGRAMMATION AVEC WPF

La programmation avec WPF introduit quelques nouveaux paradigmes et fonctionnalités dans le développement Windows

- La conception de l'interface continue de faire partie intégrante du processus de développement
 - Il fournit une partie des fonctionnalités de création/gestion d'objets
 - Les langages de programmation tels que C#, VB.NET, etc. sont utilisés pour définir la logique dans le code derrière

PROGRAMMATION AVEC WPF

- WPF est en fait une collection de classes et d'assemblies dans Microsoft .NET Framework
 - Introduit dans .NET Framework 3.0
- Les classes WPF modifient fondamentalement de nombreux aspects de l'exécution et du fonctionnement des applications Windows

PROGRAMMATION AVEC WPF

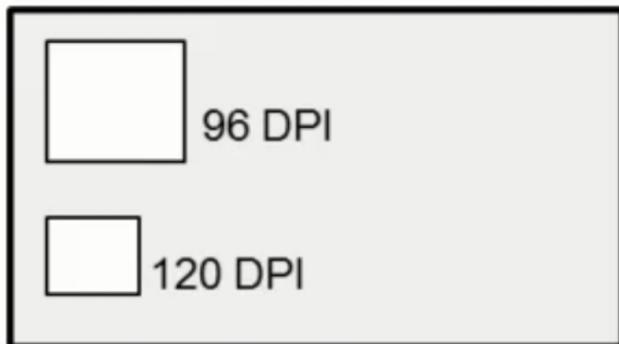
WPF introduit des paramètres DPI indépendants de l'appareil pour les applications créées avec lui

- Les applications sont désormais systématiquement ouvertes sur différentes tailles et résolutions d'affichage
 - Les différents paramètres DPI/tailles d'écran peuvent produire des interfaces bizarres dans les applications Windows Forms

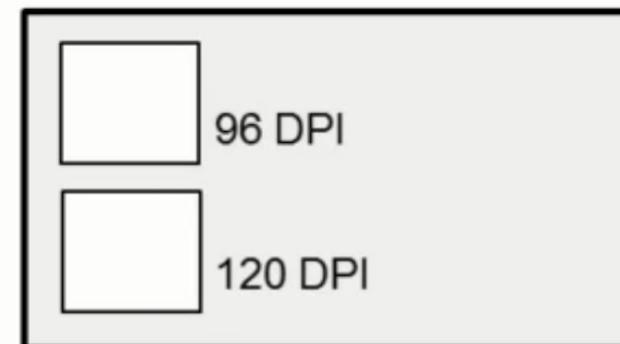
PROGRAMMATION AVEC WPF

Les applications Windows Forms utilisent une approche basée sur les pixels

- WPF effectue le rendu des objets indépendamment des paramètres DPI de l'appareil



Windows Forms



WPF Application

PROGRAMMATION AVEC WPF

Cette fonctionnalité est initialement transparente pour le développeur

- Les résultats de cette nouvelle fonctionnalité doivent être pris en compte lors de la conception de la mise en page de l'application
- La possibilité de gérer chaque aspect visuel de l'interface en tant qu'objet individuel changera l'approche du développeur
 - Et nécessitent un peu d'expérience

BALISAGE ET CODE-BEHIND

Les applications WPF sont constituées de balisage (XAML) et de code-behind (code managé)

- Le balisage définit la mise en page et l'apparence de l'application
- Le code-behind définit le comportement
 - L'éditeur de code dans Visual Studio est utilisé pour modifier le code derrière

BALISAGE ET CODE-BEHIND

Il existe quatre façons d'accéder à l'éditeur de code

- À partir de l'éditeur de vue de conception
 - Double-cliquez sur un objet
 - Cliquez avec le bouton droit de la souris dans l'éditeur de mode de conception et choisissez Afficher le code

BALISAGE ET CODE-BEHIND

Il existe quatre façons d'accéder à l'éditeur de code

- À partir de l'Explorateur de solutions
 - Cliquez avec le bouton droit sur un fichier avec l'extension .xaml et choisissez Afficher le code
 - Double-cliquez sur un fichier de code dans l'Explorateur de solutions (pour C#, il s'agit d'un fichier .cs)

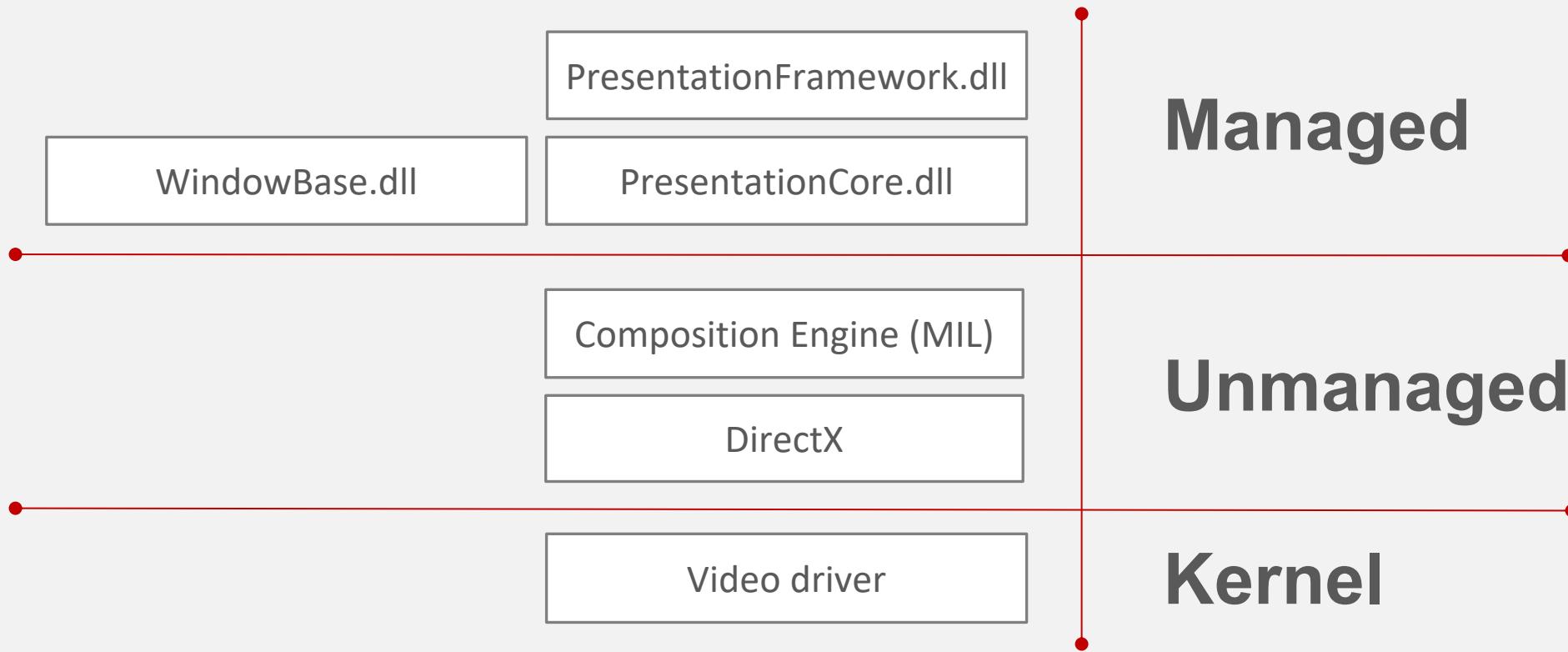
Le code peut également être modifié dans Blend

- Nous verrons l'utilisation de Blend plus tard dans le cours

ATTRIBUT DE CLASSE XAML

- XAML est mappé à des classes dans le .NET Framework
- L'élément racine d'un XAML a souvent un attribut x :Class
 - Spécifie le code-behind associé à la classe définie dans le balisage

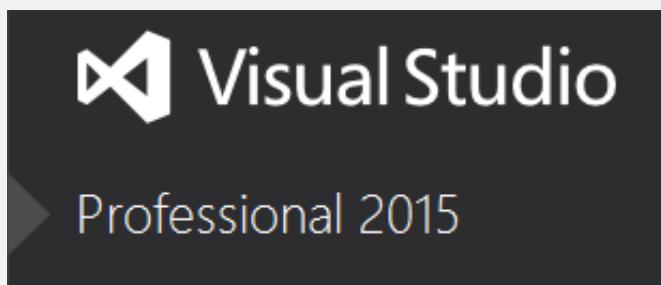
WPF Architecture



WPF Tools

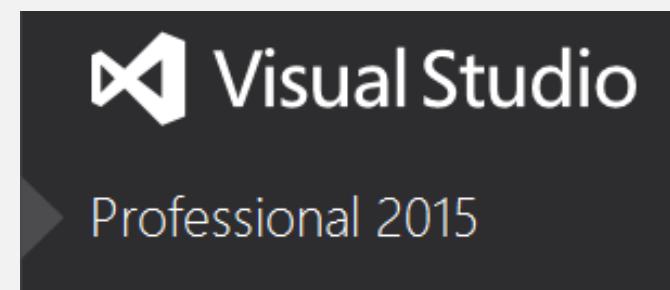


Outil interactif qui offre une expérience de portée de la création de conception et de mises en page sophistiquées.



Fournit une expérience de codage et de débogage de portée.

WPF Tools Recent Improvements



*Apparence et convivialité cohérentes avec Visual Studio
Powerful Solution Explorer
Powerful IntelliSense
Amélioration de la navigation dans le code XAML
Plusieurs améliorations réduisent le nombre de
basculements entre Blend et Visual Studio*

*Nouvel outil de performance - Chronologie
Amélioration du débogage des liaisons et des
arborescences visuelles*

Résumé

- WPF est une technologie préférable pour la création d'applications d'interface utilisateur Windows
- WPF a de l'avenir en tant que partie de la Plate-forme .NET
- Blend et Visual Studio sont les excellents outils qui prennent en charge WPF

II : Comprendre le XAML

Qu'est ce que le XAML ?

- **XAML** (eXtensible Application Markup Language) est un langage de balisage déclaratif utilisé principalement dans les technologies Microsoft telles que **WPF** (Windows Presentation Foundation), UWP (Universal Windows Platform) et Xamarin.Forms.
- Il permet de définir **l'interface utilisateur** (UI) d'une application de manière déclarative, séparant ainsi la conception visuelle du code logique.

Qu'est ce que le XAML ?

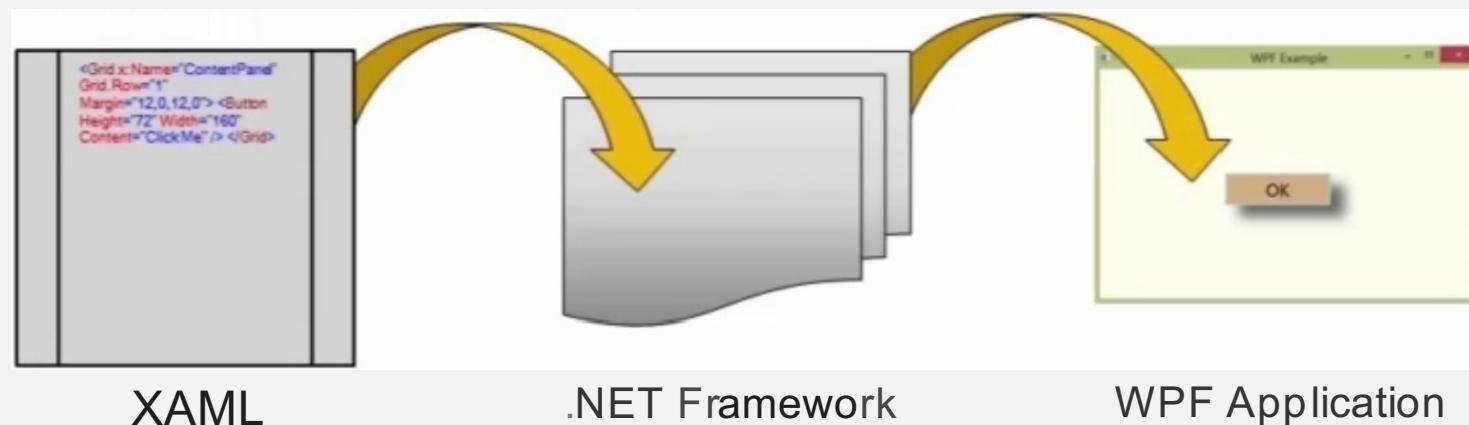
La réponse la plus simple

- XAML est une forme de XML
 - XML (eXtensible Markup Language)
 - Lisible par l'homme et par la machine
 - Permet de décrire, de stocker et de transporter des données dans un format standardisé

Qu'est ce que le XAML ?

Réponse complète : XAML est en fait deux choses

- Langage de balisage basé sur XML
- Langage de programmation déclaratif mappé à Microsoft .NET Framework



Qu'est ce que le XAML ?

Que signifie exactement la partie « application » pour XAML ?

- XAML est une forme de XML...
 - Avec un ensemble de règles sur ses éléments et attributs
 - Comment ces éléments et attributs sont mappés aux objets, leurs propriétés et les valeurs de ces propriétés
 - Bien plus qu'un simple balisage

Déclarative vs Procédurale

XAML offre deux avantages principaux aux développeurs d'applications

- Sépare la conception de l'interface de l'application du code de l'application
- Exécuté dans un environnement de programmation procédurale

Déclarative vs Procédurale

- Quel est le problème de la programmation procédurale ?
- Il existe un certain nombre de méthodologies de programmation différentes
 - Nous comparerons les éléments déclaratifs et procéduraux

Déclarative vs Procédurale

Programmation procédurale

- Vous spécifiez les étapes exactes qui doivent être suivies pour produire le résultat souhaité

Programmation déclarative

- Vous demandez les résultats que vous souhaitez sans décrire les étapes nécessaires pour l'obtenir

Déclarative vs Procédurale

L'aspect procédural de XAML fournit une grande partie de sa puissance et de sa relative simplicité

- Les concepteurs d'interface décrivent ce qui est nécessaire
- Le mappage et l'interaction de XAML avec le .NET Framework fournissent le résultat de manière découpée

Déclarative vs Procédurale

L'aspect procédural de XAML fournit une grande partie de sa puissance et de sa relative simplicité

- Vous pouvez faire bien plus que d'afficher une interface utilisateur statique avec XAML
- Créez des animations, intégrez des vidéos et liez-les à des données à l'aide d'un simple balisage

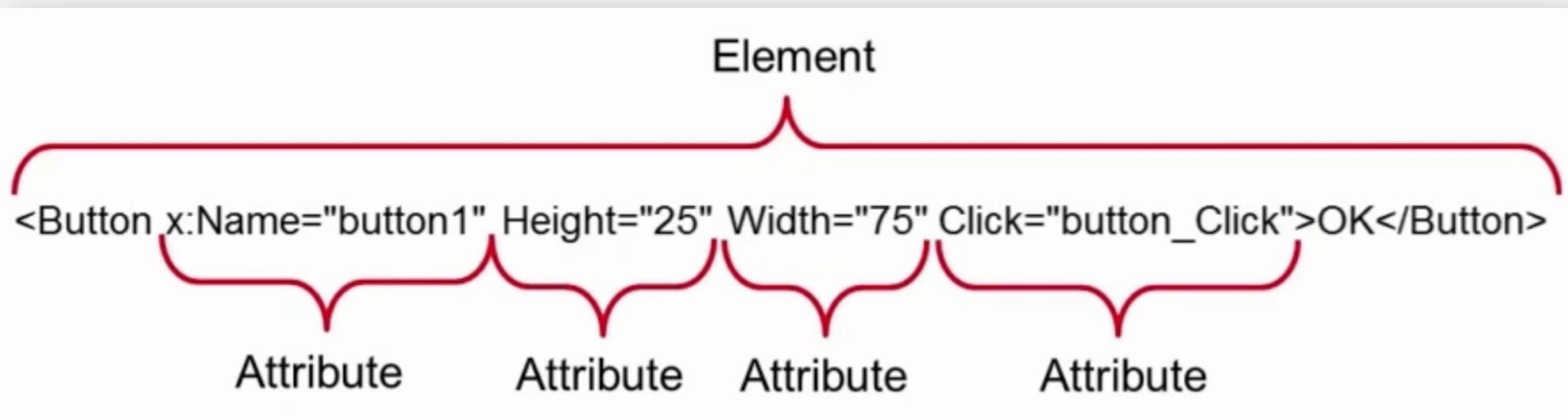
Eléments et Attribut

La spécification XAML définit des règles relatives au mappage des classes .NET Framework au balisage XAML

- Comprendre les termes rend la compréhension des relations beaucoup plus facile

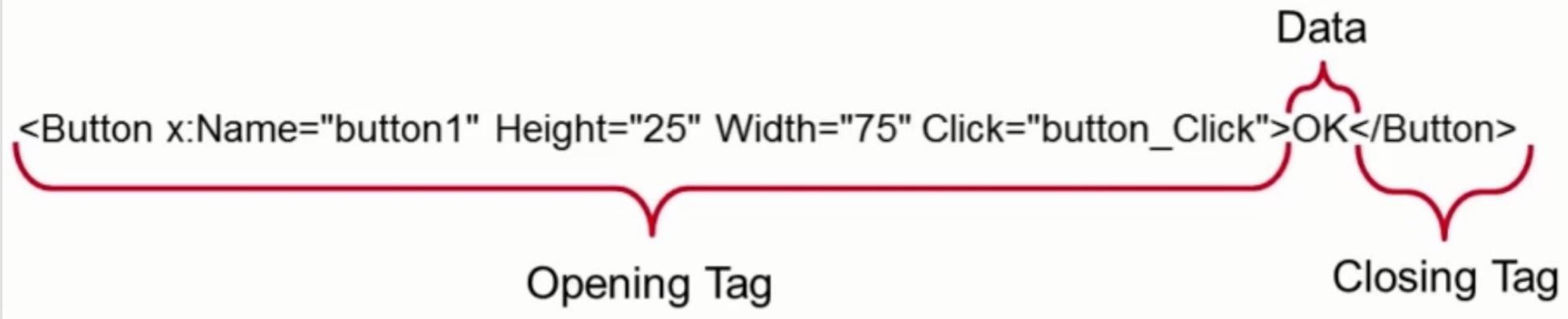
Eléments et Attribut

Examen rapide des éléments et attributs XML



Eléments et Attribut

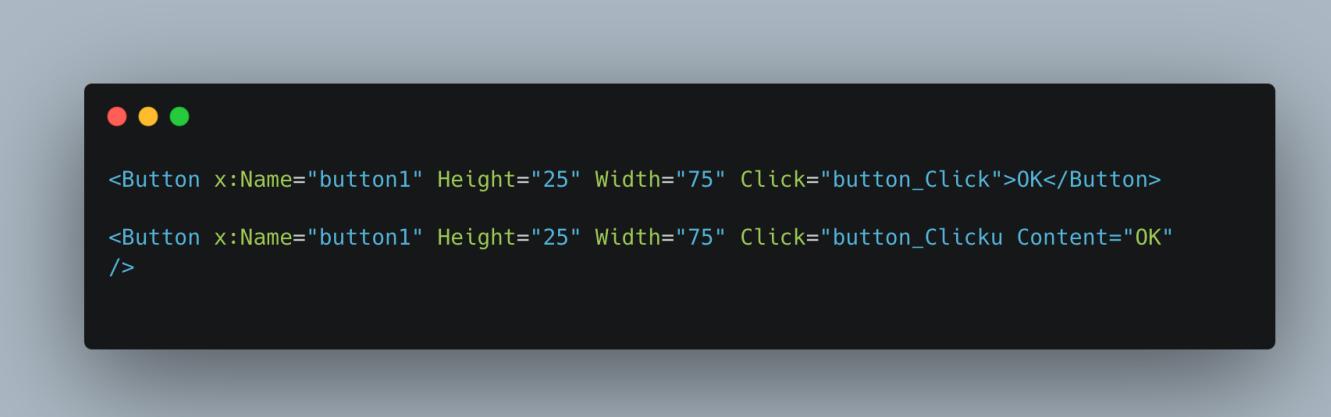
Examen rapide des éléments et attributs XML



Eléments et Attribut

Examen rapide des éléments et attributs XML

- Cet élément peut également être écrit sous la forme d'une balise à fermeture automatique
 - Toutes les données sont représentées sous forme d'attributs



```
<Button x:Name="button1" Height="25" Width="75" Click="button_Click">OK</Button>
<Button x:Name="button1" Height="25" Width="75" Click="button_Click" Content="OK"
/>
```

Eléments et Attribut

XAML est construit à l'aide d'espaces de noms, d'éléments et d'attributs

- Ceux-ci sont mappés aux espaces de noms, aux types, aux propriétés et aux événements du .NET Framework

XAML		.NET Framework
Namespace	↔	Namespace
Element	↔	Class
Attribute	↔	Property/Event

Eléments et Attribut

- Ce code XAML crée un bouton

```
● ● ●  
<Button xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation Content="OK" Click="button_Click" />
```

- Il s'agit du code C# équivalent

```
● ● ●  
System.Windows.Controls.Button b = new System.Windows.Controls.Button();  
b.Click += new System.Windows.RoutedEventHandler(button_Click);  
b.Content="OK";
```

Namespaces

Un aspect de XAML qui intimide et déroute les nouveaux utilisateurs les namespaces

- On dirait qu'il y en a partout.
- Elles sont longues et complexes.
- Comment devez-vous vous souvenir de les ajouter à votre code ?

Namespaces



```
<Window x:Class="MonApplication.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:MonApplication" <!-- Définition du namespace local -->
    Title="Ma Fenêtre" Height="350" Width="525">
    <Grid>
        <!-- Contenu de l'interface utilisateur -->
    </Grid>
</Window>
```

1. xmlns : La première ligne définit le namespace par défaut pour les éléments XAML. Il pointe vers l'espace de noms par défaut de WPF.

2. xmlns:x : Cette ligne ajoute un autre namespace (<http://schemas.microsoft.com/winfx/2006/xaml>) avec le préfixe x. Il est souvent utilisé pour déclarer des extensions XAML, comme les Storyboards.

3. xmlns:local : Cette ligne crée un namespace personnalisé appelé local qui fait référence à l'espace de noms de l'application actuelle. Cela permet d'utiliser des classes et des ressources définies dans cet espace de noms dans le XAML.

4. Title, Height, Width : Ce sont des propriétés de la fenêtre définies en XAML.

Namespaces

Tout le mappage d'espace de noms est codé en dur dans le framework

- Les namespaces sont utilisés uniquement en tant qu'identificateurs uniques
- Ils ne correspondent pas à une ressource Web

Namespaces

L’élément d’objet racine d’un fichier XAML doit spécifier au moins un espace de noms XML

- Le namespace est ajouté pour vous par les outils de développement XAML dans Visual Studio
- Cet espace de noms qualifie l’élément racine et tous les éléments enfants
 - Il n’est pas requis en tant qu’attribut dans chaque élément individuel (comme dans nos exemples précédents)

Namespaces

Habituellement, vous verrez cette déclaration d'espace de noms racine :

- xmlns=<http://schemas.microsoft.com/winfx/2006/xaml/presentation>

Namespaces

Vous verrez aussi généralement un espace de noms supplémentaire dans l'élément racine :

- `xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml`
- Cela correspond à l'espace de noms du langage XAML et définit des directives spéciales pour l'analyseur XAML
 - Cet espace de noms est appliqué à un élément en préfixant le nom de l'élément par « x : »
 - Souvent utilisé avec des attributs

Property Element (éléments de propriétés)

De nombreux développeurs ont la même réaction initiale à XAML

- Cela ressemble à beaucoup plus de code...

Les éléments de propriété leur permettent généralement de se sentir un peu plus à l'aise lorsqu'ils travaillent avec XAML

- La meilleure façon de comprendre les éléments de propriété est de commencer par un exemple simple de bouton

Property Element

Nous voulons créer un bouton sur lequel on clique pour arrêter un fichier vidéo ou audio

- <Button Width="95" Height="40" Content="Stop" />
- This will create a button [stop]
- A cooler solution would be to create a button with a graphic instead of the word 'stop' [•]

Property Element

Nous pourrions le faire en code C#...



```
System.Windows.Controls.Button b = new System.Windows.Controls.Button() ;  
b.Width = 96;  
b.Height = 38;  
  
System.Windows.Shapes.Rectangle r = new System.Windows.Shapes.Rectangle();  
r.Width = 10 ;  
r.Height = 10 ;  
  
r.Fill = new System.Windows.Media.Brushes.Black ; b.Content = r;
```

Property Element

Ou nous pourrions utiliser des éléments de propriété en XAML

- Au lieu d'utiliser ce code XAML pour créer le bouton...

```
<Button Width="96" Height="38" Content="Stop">
```

Property Element

Nous pouvons utiliser ce XAML ...

- La propriété content est désormais un élément au lieu d'un attribut



```
<Button Width="96" Height="38">
    <Button.Content>
        <Rectangle Width="10" Height="10" Fill="Black"/>
    </Button.Content>
</Button>
```

Property Element

Nous pouvons utiliser ce XAML ...

- La propriété content est désormais un élément au lieu d'un attribut



```
<Button Width="96" Height="38">
    <Button.Content>
        <Rectangle Width="10" Height="10" Fill="Black"/>
    </Button.Content>
</Button>
```

MARKUP EXTENSIONS

Étendez les capacités expressives et la simplicité du balisage XAML

- *Les "Markup Extensions" sont des expressions spéciales utilisées dans les fichiers XAML pour étendre la fonctionnalité de création d'objets et de déclaration de propriétés.*
- *Ils permettent de fournir des valeurs dynamiques ou de réaliser des opérations complexes au moment de l'analyse du XAML.*

MARKUP EXTENSIONS



```
<Button Content="{Binding Path=ButtonText}" />
```

Dans cet exemple, **{Binding Path=ButtonText}** est une Markup Extension appelée **Binding**. Elle permet de lier la propriété **Content du bouton** à une source de données externe, probablement définie dans le code-behind.

MARKUP EXTENSIONS



```
<TextBlock Text="{x:Static System:Environment.UserName}" />
```

Ici, **x:Static** permet d'accéder à la propriété statique **UserName** de la classe **System.Environment**.

Il existe d'autres Markup Extensions telles que **x:Null** (pour définir une valeur nulle), **x:Array** (pour créer des tableaux), **x>Type** (pour spécifier un type), etc.

MARKUP EXTENSIONS

- Les **Markup Extensions** sont particulièrement utiles dans les scénarios de liaison de données, de templates et de styles, car elles permettent de définir des comportements dynamiques ou des valeurs qui dépendent du contexte d'exécution.
- En résumé, les **Markup Extensions** sont des expressions spéciales utilisées dans le langage XAML pour étendre la manière dont les objets sont créés et configurés. Elles sont largement utilisées dans WPF pour des fonctionnalités comme la liaison de données, la définition de templates et plus encore.

X:NAME

Les éléments XAML doivent avoir un nom unique pour être référencés dans les fonctionnalités code-behind :

- <Button Content="OK' Name="button1" Click="button_Click" />

Vous pouvez également voir l'attribut x : utilisé avec le nom

- <Button Content="OK" x:Name="button2" Click="button_Click" />

Quelle est la différence ?

X:NAME

Ils sont fondamentalement interchangeables

- La plupart des classes .NET Framework définissent une propriété name ou héritent d'une classe qui la définit
 - Ils peuvent utiliser name, sans le x :
- Pour les éléments/objets qui n'ont pas la propriété name, ils peuvent utiliser la propriété name de l'espace de noms XAML
 - `xmlns:x="http://schemas.microsoft.com/wi nfx/2006/xaml"`

X:NAME

Le préfixe x : définit l'attribut name à partir d'un espace de noms séparé

- Permet d'accéder à la spécification du langage XAML de l'attribut name

COMMON XAML COMPLAINTS

WPF et XAML apportent un tout nouveau paradigme de développement/codage au développement Windows

- Il y a une courbe d'apprentissage
- Il y a deux plaintes courantes

COMMON XAML COMPLAINTS

Verbosité

- XAML est basé sur XML qui est trop verbeux pour être tapé
 - De nombreuses frappes dupliquées, des mots, etc. grâce aux balises d'ouverture et de fermeture, etc.
- Cette plainte est valable

COMMON XAML COMPLAINTS

Verbosité

- Microsoft continue d'améliorer les outils de développement afin de réduire considérablement la quantité d'entrées XAML manuelles
 - Le développement de l'outil a pris du retard sur certains aspects de la mise en œuvre (30, chemins, formes, etc.)

COMMON XAML COMPLAINTS

Les systèmes basés sur XML souffrent de performances médiocres

- XML est conçu pour assurer l'interopérabilité avant tout
 - L'efficacité de la représentation et du développement n'est pas une priorité
 - Les données XML sont des données textuelles qui doivent être analysées
- WPF compile XAML en BAML (XAML binaire)
 - Optimisé pour l'efficacité à l'exécution
- Les performances en pâtissent encore au moment du développement

Layout

La gestion des tailles

La gestion des tailles dans WPF avec XAML se fait principalement à l'aide de différentes propriétés et mécanismes :

- **Width et Height**
- **MinWidth et MinHeight,**
- **MaxWidth et MaxHeight**

La gestion des tailles

1. Width et Height :

- Vous pouvez définir explicitement la largeur (Width) et la hauteur (Height) d'un élément en utilisant ces propriétés. Par exemple :



```
<TextBlock Text="{x:Static System:Environment.UserName}" />
```

La gestion des tailles

2. MinWidth et MinHeight, MaxWidth et MaxHeight :

- Vous pouvez définir des limites minimales (MinWidth, MinHeight) et maximales (MaxWidth, MaxHeight) pour les dimensions d'un élément. Cela permet de contrôler la taille minimale et maximale que l'élément peut avoir.



```
<Button MinWidth="50" MinHeight="30" MaxWidth="150" MaxHeight="100" Content="Cliquez-moi" />
```

La gestion des tailles

2. MinWidth et MinHeight, MaxWidth et MaxHeight :

- Vous pouvez définir des limites minimales (MinWidth, MinHeight) et maximales (MaxWidth, MaxHeight) pour les dimensions d'un élément. Cela permet de contrôler la taille minimale et maximale que l'élément peut avoir.



```
<Button MinWidth="50" MinHeight="30" MaxWidth="150" MaxHeight="100" Content="Cliquez-moi" />
```

La gestion des tailles

- Dans WPF, les propriétés **DesiredSize**, **RenderSize**, **ActualHeight**, et **ActualWidth** sont en lecture seule. Cela signifie que vous ne pouvez pas les définir directement.
- Elles sont automatiquement calculées par le système de disposition de WPF en fonction de divers facteurs tels que la taille du contenu, les contraintes de disposition, et les propriétés parentes.

La gestion des tailles

- **DesiredSize** : En lecture seule. Représente la taille souhaitée de l'élément, calculée par le système de disposition en fonction du contenu et des propriétés de l'élément.
- **RenderSize** : En lecture seule. Représente la taille réelle de l'élément après le rendu, calculée par le système de rendu de WPF.
- **ActualHeight** et **ActualWidth** : En lecture seule. Représentent respectivement la hauteur et la largeur réelles de l'élément après la mise en page et le rendu.

Autres concepts Layout

1. Alignement (HorizontalAlignment, VerticalAlignment)
2. Le Margin
3. Le Padding
4. Etc.

Les panels

PLAN

- Layout Core Types
- Layout Process
- Sizing Elements
- Positioning Elements
- ViewBox
- Panels: StackPanel, WrapPanel, DockPanel, Grid

Layout Core Types

- Les "**Layout Core Types**" font référence à un ensemble de classes au sein de **Windows Presentation Foundation** (WPF) qui sont essentielles pour la gestion des dispositions (layouts) des éléments visuels.
- Ces classes fournissent les fonctionnalités de base pour organiser et positionner les éléments au sein de l'interface utilisateur.

Layout Core Types

1. **UIElement** :

- **UIElement** est une classe de base pour tous les éléments visuels dans WPF.
- Elle définit des propriétés et des méthodes communes à tous les éléments, telles que la taille, la position, la visibilité, etc.
- Les éléments dérivés de **UIElement** peuvent participer à la disposition et à l'agencement de l'interface utilisateur.

Layout Core Types

2. FrameworkElement :

- FrameworkElement hérite de UIElement et ajoute des fonctionnalités supplémentaires spécifiques à la disposition, telles que les propriétés Margin, HorizontalAlignment, VerticalAlignment, etc.
- Elle fournit également des événements liés à la disposition, tels que SizeChanged.

Layout Core Types

FrameworkElement:

- VerticalAlignment  Top, Center, Bottom, Stretch
- HorizontalAlignment  Left, Right, Center, Stretch

Control:

- VerticalContentAlignment
- HorizontalContentAlignment

Les alignements n'ont d'impact sur la mise en page que lorsque la taille souhaitée d'un contrôle est inférieure à la taille disponible pour son rendu.

Layout Core Types

3. Layout Process (Processus de Mise en Page) :

- Le processus de mise en page (layout process) est le flux d'exécution qui se produit lorsqu'un élément visuel est ajouté, modifié ou rendu dans l'interface utilisateur.
- Il comprend deux phases principales : **la mesure (measure)** et **l'agencement (arrange)**.
- Pendant la phase de mesure, les éléments parent demandent à leurs enfants de spécifier leur taille souhaitée.
- Ensuite, pendant la phase d'agencement, les éléments parent placent effectivement leurs enfants dans l'espace disponible.

Layout Process

Le processus de mise en page se compose de deux étapes:

- **Measure**
- **Arrange**
- **Step 1.** Chaque élément de l'interface utilisateur détermine sa taille souhaitée
Chaque élément demande à ses enfants (en appelant la méthode Measure sur chaque enfant) combien d'espace ils veulent englober.
- **Step 2. Attribuer des tailles et des positions**

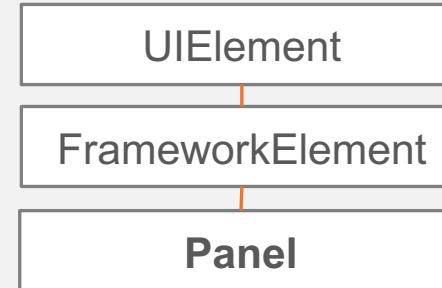
Les éléments appellent récursivement sur leurs enfants la méthode Arrange en passant la taille allouée à chaque enfant et la position à partir de laquelle l'enfant peut s'organiser.

Layout Core Types

4. Panel :

- Panel est une classe abstraite qui définit la base pour tous les conteneurs de disposition (tels que Grid, StackPanel, Canvas, etc.).
- Les panels sont responsables de l'agencement et de la position des éléments enfants.

Layout Core Types



Les panneaux composent ensemble les éléments de l'interface utilisateur.

Le panneau expose:

- **UIElementCollection Children**

Panels: Canvas, StackPanel, WrapPanel, DockPanel and Grid

Canvas

Canvas (Toile) :

Le Canvas permet un positionnement absolu des éléments en utilisant les coordonnées X et Y. Il est utile lorsque vous avez besoin de contrôler précisément la position des éléments.

Canvas

Canvas permet d'utiliser le positionnement absolu de ses éléments d'interface utilisateur enfants.

```
<Canvas>
    <Rectangle Canvas.Left="12" Canvas.Top="12"
               Fill="Red"/>
    <Rectangle Canvas.Left="12" Canvas.Bottom="12"
               Fill="Black"/>
    <Rectangle Canvas.Right="12" Canvas.Top="12"
               Fill="Blue"/>
    <Rectangle Canvas.Right="12" Canvas.Bottom="12"
               Panel.ZIndex="1" Fill="BlueViolet"/>
    <Rectangle Canvas.Right="64" Canvas.Bottom="120"
               Panel.ZIndex="2" Fill="Chartreuse"/>
</Canvas>
```

WrapPanel

WrapPanel (Panneau Enveloppant) :

Le WrapPanel organise les éléments dans des lignes et des colonnes, mais si l'espace est insuffisant, il les enveloppe dans une nouvelle ligne.

WrapPanel

WrapPanel permet d'organiser les éléments de l'interface utilisateur dans des piles en les enveloppant en cas de débordement.

Expose:

- Orientation  default
Horizontal, Vertical
- ItemWidth  Uniform sizes
- ItemHeight
- *Vertical WrapPanel ajoute une colonne lorsque les enfants débordent de la hauteur du panneau*
- *Horizontal WrapPanel ajoute une ligne lorsque les enfants dépassent la largeur du panneau*

DockPanel

DockPanel (Panneau d'Ancre) :

Le DockPanel permet d'ancrer les éléments sur les bords de son conteneur parent.

DockPanel

DockPanel permet d'organiser les éléments de l'interface utilisateur en les ancrant aux bordures.

Expose:

- Dock  Left, Right, Bottom, Top
 - LastChildFill  True\False
 - ↑
default
-
- *Le dernier enfant remplit l'espace restant, sauf si LastChildFill est défini sur false*
 - *Si LastChildFill est défini sur true, la valeur Dock du dernier enfant est ignorée*

Grid

Grid (Grille) :

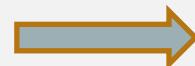
- Le **Grid** organise les éléments dans une structure de lignes et de colonnes.
- Il est utile lorsque vous devez créer une disposition de tableau.

Grid

Grid permet d'organiser les éléments de l'interface utilisateur en lignes et en colonnes.

Expose:

- RowDefinitions
- ColumnDefinitions



Utilisé pour définir des lignes et des colonnes

Pour positionner les enfants à l'intérieur d'une grille:

- Grid.Row, Grid.Column
- Grid.RowSpan, Grid.ColumnSpan

Grid

RowDefinition et ColumnDefinition:

- GridLength Height, Width
- double MinHeight, MinWidth

GridLength peut être défini sur :

Points

```
<RowDefinition  
Height="100"/>
```

La hauteur est fixe

Auto

```
<RowDefinition  
Height="Auto"/>
```

La taille est
déterminée par
l'enfant le plus haut

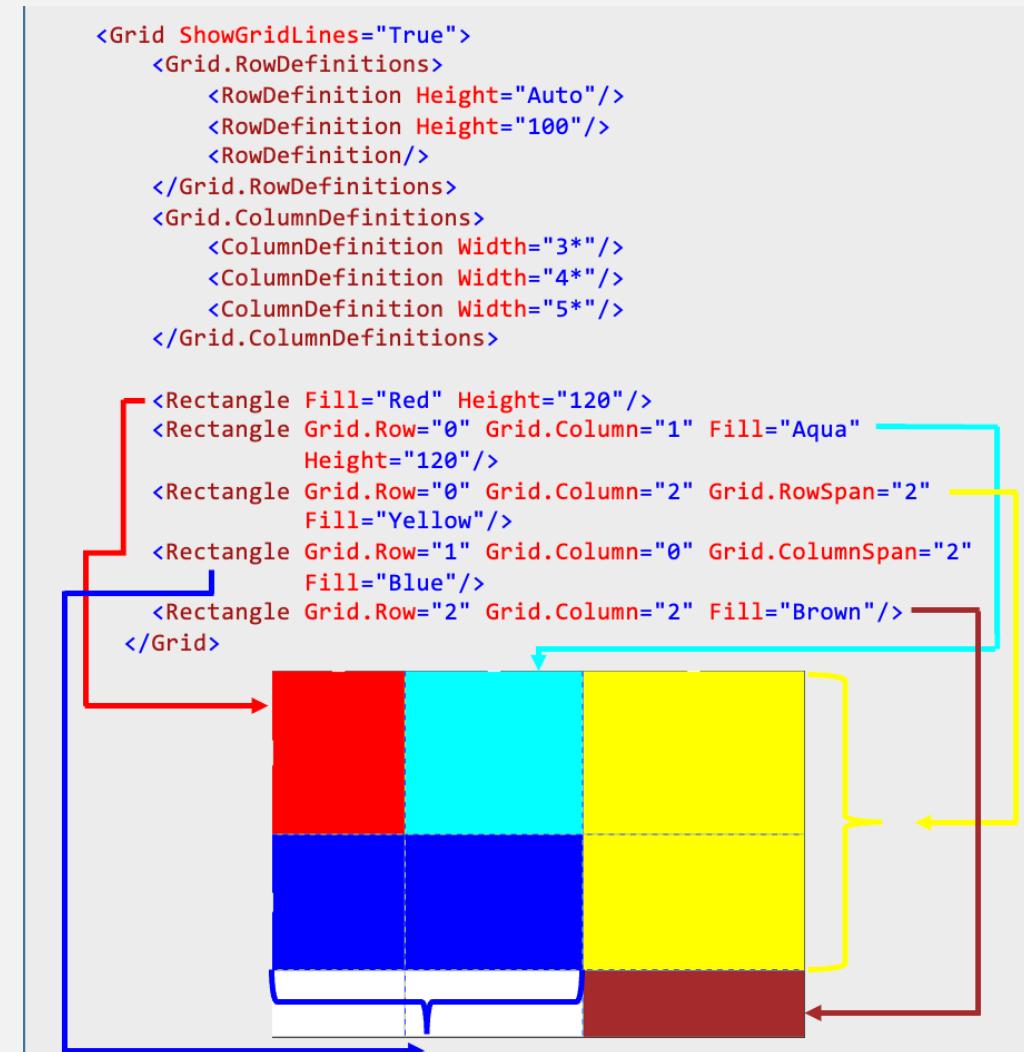
Star

```
<ColumnDefinition  
Width="3*"/>
```

La valeur de largeur
est relative

```
<Grid ShowGridLines="True">  
  <Grid.RowDefinitions>  
    <RowDefinition Height="Auto"/>  
    <RowDefinition Height="100"/>  
    <RowDefinition/>  
  </Grid.RowDefinitions>  
  <Grid.ColumnDefinitions>  
    <ColumnDefinition Width="3*"/>  
    <ColumnDefinition Width="4*"/>  
    <ColumnDefinition Width="5*"/>  
  </Grid.ColumnDefinitions>
```

```
  <Rectangle Fill="Red" Height="120"/>  
  <Rectangle Grid.Row="0" Grid.Column="1" Fill="Aqua" Height="120"/>  
  <Rectangle Grid.Row="0" Grid.Column="2" Grid.RowSpan="2" Fill="Yellow"/>  
  <Rectangle Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2" Fill="Blue"/>  
  <Rectangle Grid.Row="2" Grid.Column="2" Fill="Brown"/>  
</Grid>
```



Résumé

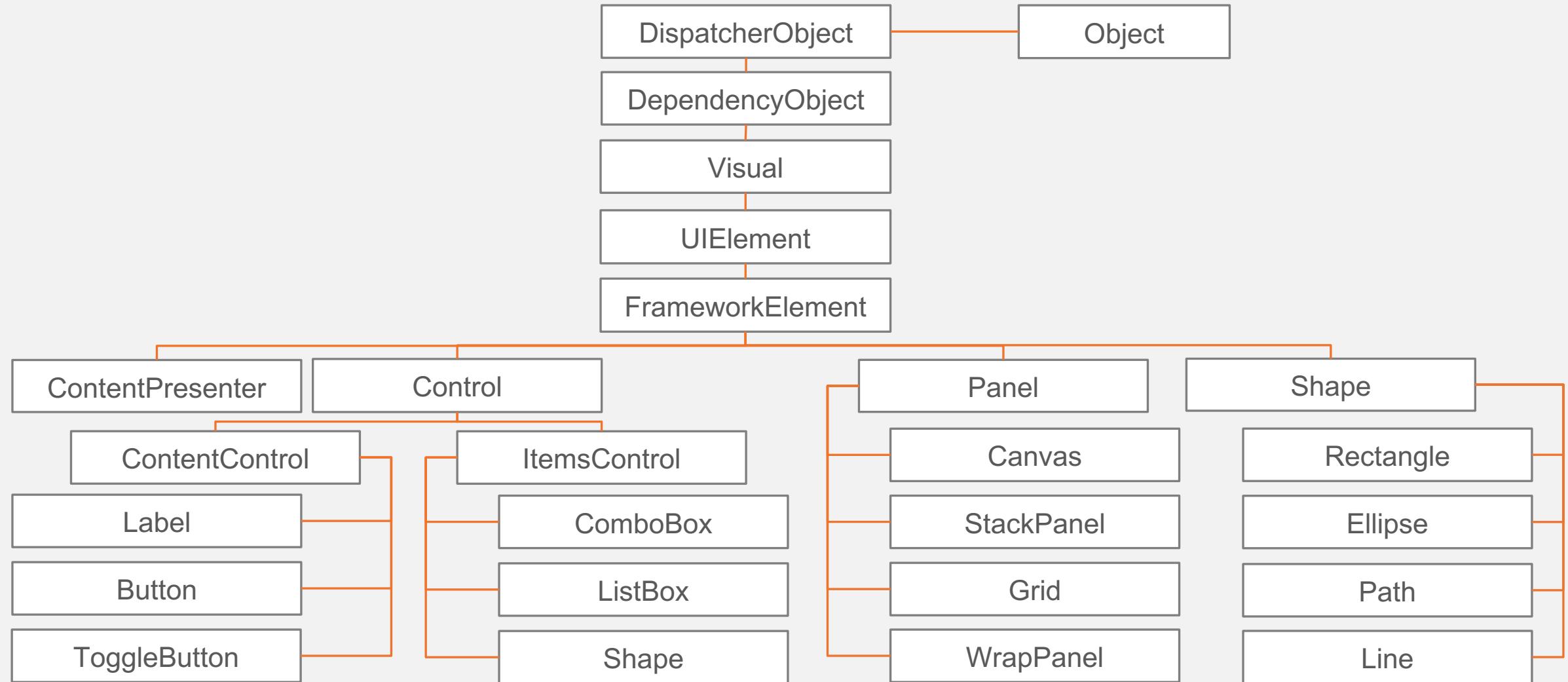
- **Types importants : UIElement, FrameworkElement, Panel**
- **Layout Process:** Mesurez et organisez
- **Sizing:** Height, Width, MinHeight, MinWidth, MaxHeight, MaxWidth, Margin, Padding, Visibility
- **Positioning:** VerticalAlignment, HorizontalAlignment, VerticalContentAlignment, HorizontalContentAlignment
- **Panels:** Canvas, StackPanel, WrapPanel, DockPanel, Grid

Les contrôles

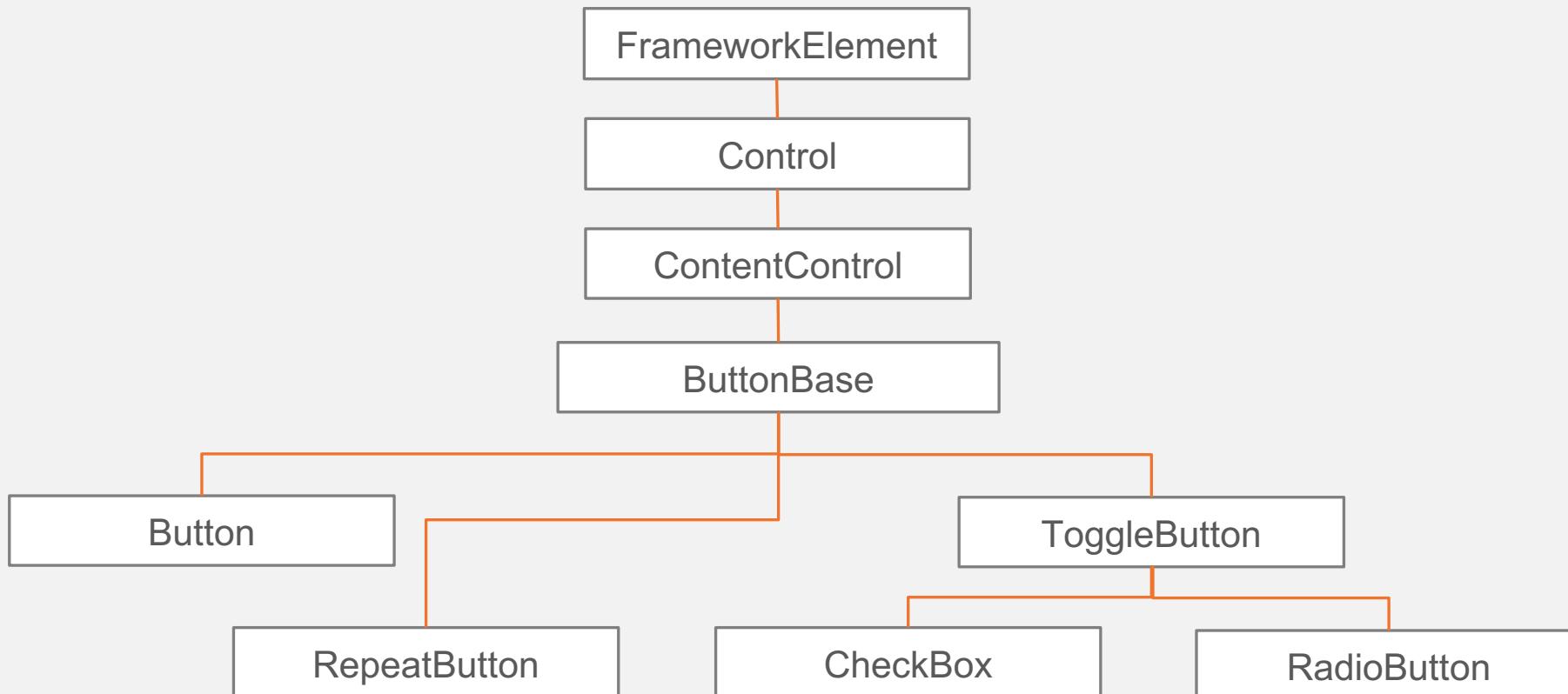
PLAN

- Buttons
- Containers avec Headers
- Items Controls
- Text Input

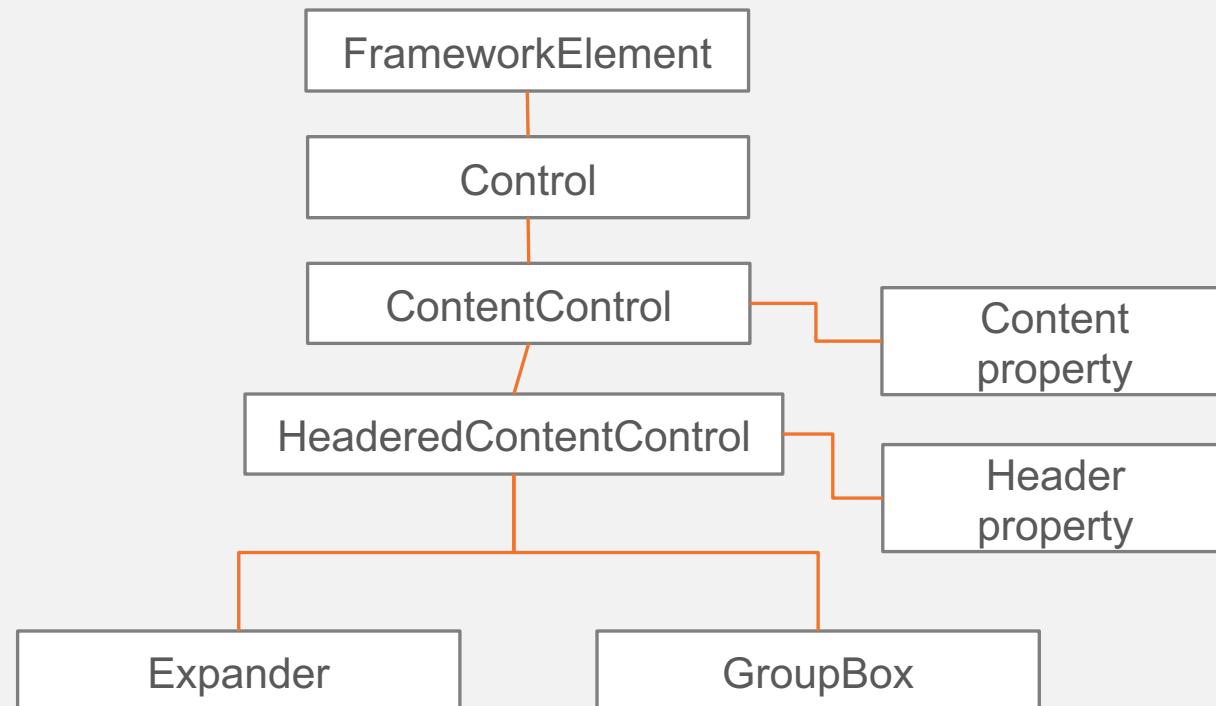
Class Hierarchy



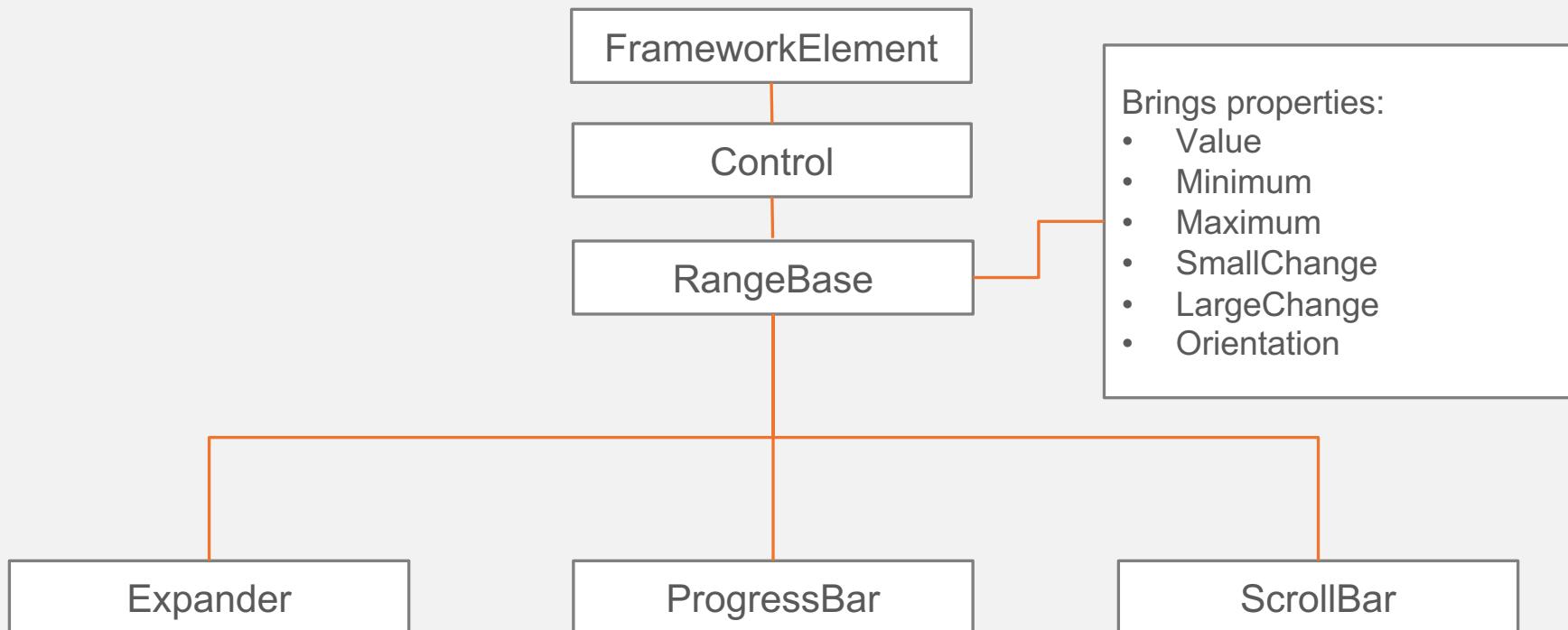
Buttons



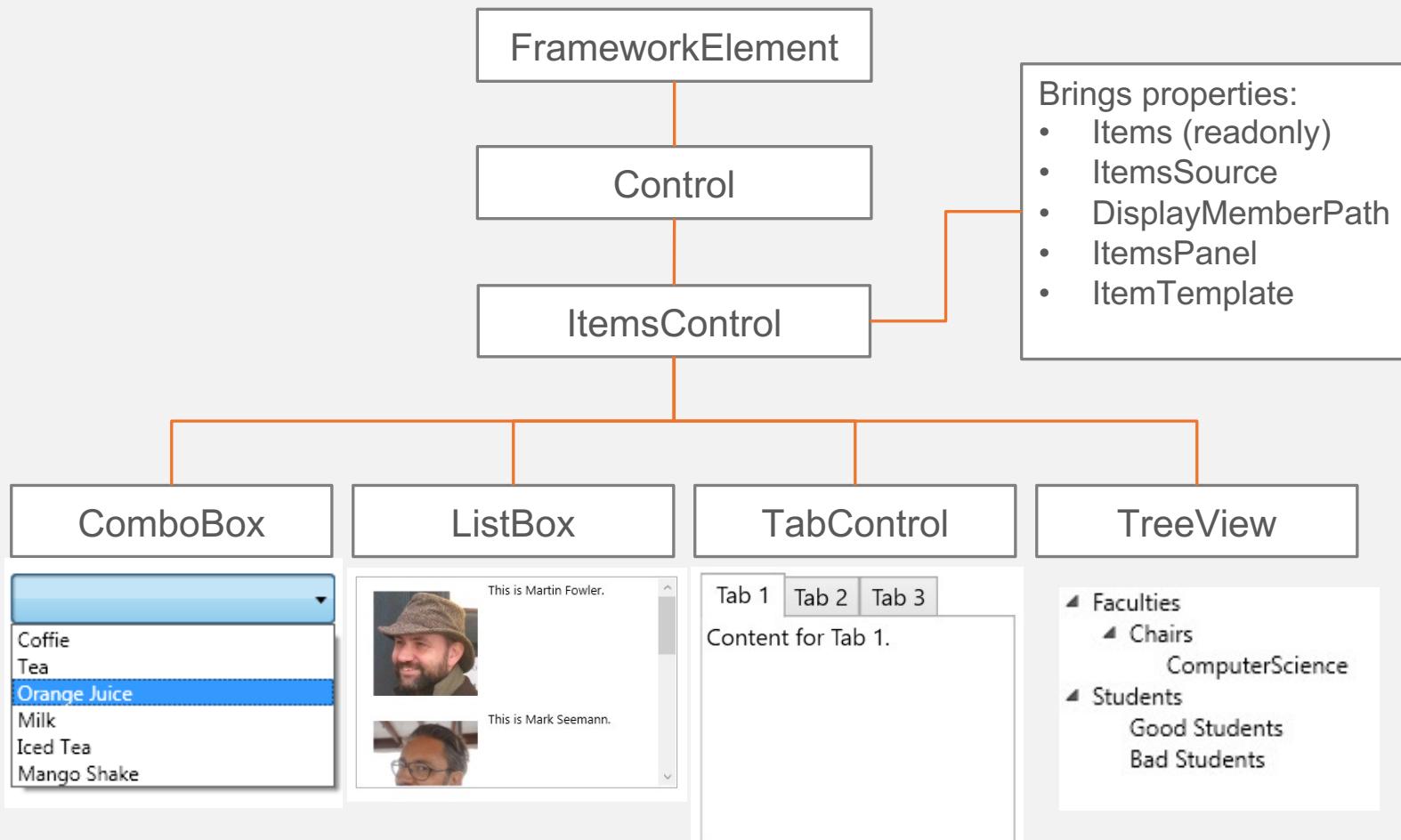
Headered Content Controls



Range Controls



Items Controls



Items Controls

Les sélecteurs permettent de sélectionner des éléments indexés.

ItemsControl

Selector

Selector exposes:

- SelectedItem
- SelectedIndex
- SelectedValue
- SelectedValuePath

Le sélecteur expose l'événement SelectionChanged de la signature:

```
void SelectionChangedEventHandler(object sender,  
                                SelectionChangedEventArgs e);
```

SelectionChangedEventArgs exposes:

- **IList** AddedItems
- **IList** RemovedItems

Image

Image est un contrôle permettant d'héberger des images.

Selector expose:

- Source default
 - ImageSource
 - Stretch  None, Fill, Uniform, UniformToFill
 - StretchDirection  Both, DownOnly, UpOnly
- default

Rendering quality est contrôlé par l'intermédiaire de la fonction
RenderingOptions.BitmapScalingMode:
Fant, HighQuality, Linear, LowQuality, NearestNeighbor

IV – Liaison de données

PLAN

1. Comment fonctionnent le Binding ?
2. Comment lier les éléments de l'interface utilisateur les uns aux autres ?
3. Quels sont les modes de liaison (binding) pris en charge ?
4. Data Context
5. Data Templates
6. Binding une Collection
7. DataGrid
8. Converters

Principes de base du Data Binding

- Le binding (liaison de données) est une caractéristique clé de Windows Presentation Foundation (WPF), un framework de développement d'interfaces utilisateur pour les applications de bureau sous Windows.
- Il permet de synchroniser automatiquement les données entre la source (généralement un objet de modèle de données) et l'interface utilisateur, sans nécessiter de code de mise à jour manuel.
- Voici comment fonctionne le binding dans WPF :
 1. **Source de données (DataContext) :**
 2. **Propriétés de dépendance :**
 3. **Modes de liaison :**
 4. **Chemins de liaison :**
 5. **Conversion de données**
 6. **Notifications de changement :**

Principes de base du Data Binding

- **Source** – Définit un objet source
- **Path** – définit un chemin d'accès à une propriété source
- **ElementName** – définit un objet source dans l'arborescence des éléments visuels
- **Mode** – OneWay, TwoWay, OneWayToSource

Principes de base du Data Binding

- **Source** – Définit un objet source
- **Path** – définit un chemin d'accès à une propriété source
- **ElementName** – définit un objet source dans l'arborescence des éléments visuels
- **Mode** – OneWay, TwoWay, OneWayToSource

Source Object

- **ElementName** – définit un objet source dans l’arborescence des éléments visuels
- **RelativeSource** – similaire à ElementName, mais fournit des fonctionnalités spéciales pour la recherche d’éléments d’interface utilisateur selon un critère
- **DataContext** – définit une source comme contexte pour de nombreux éléments
- **Source** – définit un objet source en le référençant via l’extension StaticResource

DataGrid

- Le contrôle DataGrid ressemble beaucoup à un ListView, lorsqu'il est utilisé avec d'un GridView, mais il offre de nombreuses fonctionnalités supplémentaires.
- Par exemple, le DataGrid peut générer automatiquement des colonnes en fonction des données que vous lui transmettez.
- Le DataGrid est également modifiable par défaut, permettant au end-user de modifier les valeurs des données sources sous-jacente.

DataGrid

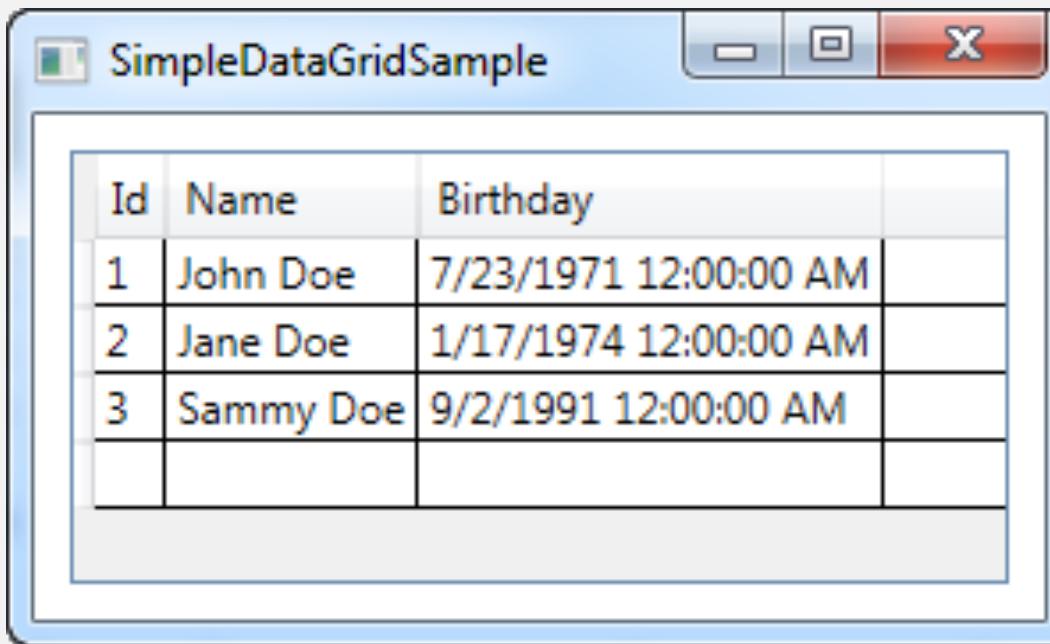
vue

```
<Window  
x:Class="WpfTutorialSamples.DataGrid_control.SimpleDataGridSample"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
Title="SimpleDataGridSample" Height="180" Width="300">  
    <Grid Margin="10">  
        <DataGrid Name="dgSimple"></DataGrid>  
    </Grid>  
</Window>
```

Code Behind

```
using System;  
using System.Collections.Generic;  
using System.Windows;  
  
namespace WpfTutorialSamples.DataGrid_control  
{  
    public partial class SimpleDataGridSample : Window  
    {  
        public SimpleDataGridSample()  
        {  
            InitializeComponent();  
  
            List<User> users = new List<User>();  
            users.Add(new User() { Id = 1, Name = "John Doe", Birthday = new DateTime(1971, 7, 23) });  
            users.Add(new User() { Id = 2, Name = "Jane Doe", Birthday = new DateTime(1974, 1, 17) });  
            users.Add(new User() { Id = 3, Name = "Sammy Doe", Birthday = new DateTime(1991, 9, 2) });  
  
            dgSimple.ItemsSource = users;  
        }  
  
        public class User  
        {  
            public int Id { get; set; }  
  
            public string Name { get; set; }  
  
            public DateTime Birthday { get; set; }  
        }  
    }  
}
```

DataGrid



A screenshot of a Windows application window titled "SimpleDataGridSample". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there is a DataGrid control displaying a table with three rows of data. The table has three columns: "Id", "Name", and "Birthday". The data is as follows:

Id	Name	Birthday
1	John Doe	7/23/1971 12:00:00 AM
2	Jane Doe	1/17/1974 12:00:00 AM
3	Sammy Doe	9/2/1991 12:00:00 AM

Converters

Le convertisseur est un pont entre des types de cibles et de sources incompatibles.

IValueConverter:

public object Convert(object value, Type targetType, object parameter)

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

Le convertisseur peut être injecté via la propriété Binding.Converter.

INotifyPropertyChanged

- **INotifyPropertyChanged** est une interface de base dans le framework .NET qui est couramment utilisée pour implémenter la notification des modifications de propriétés dans les classes.
- Cette interface est souvent utilisée en conjonction avec le binding de données, notamment dans les applications WPF (Windows Presentation Foundation) et Xamarin, pour informer l'interface utilisateur lorsque les valeurs des propriétés d'un objet changent.
- Elle fait partie de l'espace de noms System.ComponentModel.

INotifyPropertyChanged

- L'interface `INotifyPropertyChanged` définit un seul événement : `PropertyChanged`, qui est déclenché chaque fois qu'une propriété de l'objet qui implémente cette interface est modifiée.
- Cet événement permet aux composants de l'interface utilisateur, tels que les contrôles d'interface graphique, de s'abonner aux modifications de propriétés et de réagir en conséquence.

INotifyPropertyChanged

```
● ● ●

using System.ComponentModel;

public class MaClasse : INotifyPropertyChanged
{
    private string _nom;

    public string Nom
    {
        get { return _nom; }
        set
        {
            if (_nom != value)
            {
                _nom = value;
                OnPropertyChanged(nameof(Nom));
            }
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Trigger

- En WPF (Windows Presentation Foundation), les déclencheurs (Triggers) sont des fonctionnalités qui permettent de définir des actions à effectuer en réponse à un certain événement ou à une condition sur un élément de l'interface utilisateur.
- Les déclencheurs sont couramment utilisés pour effectuer des modifications visuelles ou de comportement en réponse à des interactions utilisateur ou à des changements d'état.
- Il existe deux types principaux de déclencheurs en WPF : les déclencheurs de propriété (Property Triggers) et les déclencheurs d'événement (Event Triggers).

Trigger

1. Déclencheurs de propriété (Property Triggers) :

- Les déclencheurs de propriété réagissent aux changements de propriétés sur un élément de l'interface utilisateur. Ils permettent de définir des actions à exécuter lorsque certaines conditions sont remplies. Par exemple, vous pouvez utiliser un déclencheur de propriété pour changer la couleur d'arrière-plan d'un contrôle lorsque sa propriété IsMouseOver (qui indique si la souris est positionnée sur le contrôle) change.

Trigger

Exemple de déclencheur de propriété pour changer la couleur d'arrière-plan d'un bouton lorsqu'il est survolé par la souris :

```
<Button Content="Survolez-moi">
    <Button.Style>
        <Style TargetType="Button">
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="Yellow"/>
                </Trigger>
            </Style.Triggers>
        </Style>
    </Button.Style>
</Button>
```

Trigger

2. Déclencheurs d'événement (Event Triggers) :

- Les déclencheurs d'événement réagissent à des événements déclenchés par un élément de l'interface utilisateur, tels que les clics de souris, les pressions de touches, etc. Vous pouvez utiliser un déclencheur d'événement pour définir des actions à exécuter lorsque l'événement se produit.

Trigger

Exemple de déclencheur d'événement pour changer la couleur d'arrière-plan d'un bouton lorsqu'il est cliqué :

```
<Button Content="Cliquez-moi">
    <Button.Triggers>
        <EventTrigger RoutedEvent="Click">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard>
                        <ColorAnimation Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                            To="Red" Duration="0:0:1"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

V - Le MVVM

Le MVVM

- Le **MVVM**, ou Modèle-Vue-VueModèle, est un modèle d'architecture logicielle couramment utilisé dans le développement d'applications informatiques, en particulier dans le développement d'applications avec des interfaces utilisateur graphiques (GUI) telles que les applications mobiles et les applications de bureau.
- Le **MVVM** est principalement associé à des technologies telles que WPF (Windows Presentation Foundation), Xamarin, et d'autres frameworks de développement d'interfaces utilisateur.

Le MVVM

Le MVVM se compose de trois composants principaux :

- **Modèle (Model)** : Le modèle représente la couche de données de l'application. Il contient les données, la logique métier et les opérations qui sont effectuées sur les données. Le modèle ne connaît pas l'interface utilisateur et n'interagit pas directement avec elle.
- **Vue (View)** : La vue représente l'interface utilisateur de l'application. C'est la partie de l'application que l'utilisateur voit et avec laquelle il interagit. La vue est responsable de l'affichage des données et de la réception des actions de l'utilisateur, telles que les clics de souris ou les touches pressées.
- **VueModèle (ViewModel)** : Le ViewModel agit comme un intermédiaire entre le modèle et la vue. Il expose les données du modèle à la vue sous une forme adaptée à l'interface utilisateur et gère les interactions utilisateur. Le ViewModel est souvent conçu pour être lié (binding) à la vue, ce qui signifie que les modifications apportées au ViewModel se reflètent automatiquement dans la vue et vice versa.

Le MVVM

- L'un des avantages clés du MVVM est la séparation stricte des préoccupations.
- Le modèle gère les données et la logique métier, la vue gère uniquement l'interface utilisateur, et le ViewModel agit comme un pont entre les deux.
- Cela permet une plus grande facilité de testabilité, de maintenance et de réutilisation du code.
- Le binding de données est une caractéristique essentielle du MVVM, car il permet de synchroniser automatiquement les données entre le ViewModel et la vue, sans nécessiter de manipulation manuelle de l'interface utilisateur.
- Cela rend l'architecture MVVM particulièrement adaptée aux applications où la mise à jour dynamique de l'interface utilisateur en réponse aux modifications des données est importante.

Le MVVM

- En résumé, le MVVM est un modèle d'architecture logicielle qui favorise la séparation des préoccupations et l'utilisation de liaisons de données pour simplifier le développement d'applications avec des interfaces utilisateur graphiques.
- Il est largement utilisé dans le développement d'applications modernes, en particulier sur les plates-formes Microsoft et les applications multiplateformes.

Le MVVM

