WPF - Fondamentaux

Architecture et Maîtrise de l'Interface

Objectifs de la journée

- Comprendre l'architecture interne de WPF
- Maîtriser les **Dependency Properties**
- Utiliser les **Attached Properties** et **Routed Events**
- Créer des **styles** et **templates** personnalisés
- Implémenter un système de **thèmes**

Partie 1

Architecture et Fondamentaux WPF

Qu'est-ce que WPF ?

Windows Presentation Foundation - Framework de présentation moderne

- Rendu **vectoriel** basé sur DirectX
- Séparation **UI/logique** avec XAML
- **Data Binding** puissant et flexible
- **Styles et Templates** pour personnalisation complète
- Conçu pour le pattern **MVVM**

Architecture WPF

Couche	Rôle
PresentationFramework	Contrôles, Binding, Styles
PresentationCore	Visual, UIElement, ContentElement
WindowsBase	DependencyObject, Dispatcher
milcore.dll	Rendu DirectX natif

> La couche **milcore** est non-managée pour les performances

Arbre Visuel vs Arbre Logique

Deux représentations de l'interface :

Arbre Logique	Arbre Visuel
Structure **conceptuelle**	Structure de **rendu**
Éléments XAML explicites	Tous les éléments (+ templates)
Résolution des ressources	Hit-testing, transformations
Héritage des propriétés	Rendu graphique

Exemple concret

XAML simple :

```
<Button Content="Cliquez-moi"/>
```

Arbre Logique : Window → StackPanel → Button
Arbre Visuel : Window → Border → ... → Button → ButtonChrome → ContentPresenter → TextBlock

“  **Démo** : `Demo01-ArbreVisuel` ”

Navigation dans les arbres

Utilitaires WPF :

- `LogicalTreeHelper` - Parcours de l'arbre logique
- `VisualTreeHelper` - Parcours de l'arbre visuel

Scénario	Arbre à utiliser
Trouver tous les TextBox	**Visuel**
Résoudre une ressource	**Logique**
Hit-test (clic souris)	**Visuel**

Partie 2

Dependency Properties

Pourquoi les Dependency Properties ?

Les propriétés CLR classiques sont **limitées**

Dependency Properties offrent :

- ✅ Data Binding bidirectionnel
- ✅ Styles et Templates
- ✅ Animations
- ✅ Héritage de valeur
- ✅ Coercion et validation

Anatomie d'une Dependency Property

5 éléments clés :

1. **Champ statique** `readonly` (convention: `NomProperty`)
2. **Wrapper CLR** (get/set)
3. **PropertyChangedCallback** (optionnel)
4. **CoerceValueCallback** (optionnel)
5. **ValidateValueCallback** (optionnel)

Ordre de résolution des valeurs

Du plus prioritaire au moins prioritaire :

1. **Coercion** active
2. **Animation** active
3. **Valeur locale** (SetValue)
4. **Style Trigger**
5. **Template Trigger**
6. **Style Setter**
7. **Héritage**

FrameworkPropertyMetadataOptions

Option	Effet
`AffectsRender`	Déclenche un nouveau rendu
`AffectsMeasure`	Déclenche Measure()
`AffectsArrange`	Déclenche Arrange()
`Inherits`	Valeur héritée par enfants
`BindsTwoWayByDefault`	Binding TwoWay par défaut

Coercion vs Validation

Aspect	Validation	Coercion
Quand	Avant tout	Après validation
Retour	`bool` (valide/invalide)	Valeur corrigée
Si échec	Exception	Jamais d'échec
Usage	Rejeter NaN, null...	Clamp min/max

Partie 3

Attached Properties

Concept des Attached Properties

Dependency Properties applicables à **n'importe quel objet**

Exemples courants :

- `Grid.Row`, `Grid.Column`
- `Canvas.Left`, `Canvas.Top`
- `DockPanel.Dock`
- `Validation.HasError`

> Permettent d'étendre les capacités sans héritage

Cas d'usage

Usage	Exemple
Layout	`Grid.Row`, `Canvas.Left`
Comportements	Watermark, Focus, Drag&Drop
Validation	Erreurs, états
Accessibilité	`AutomationProperties.Name`

“  **Démo** : `Demo03-AttachedProperty` ”

Création d'une Attached Property

Différences avec Dependency Property :

- `RegisterAttached` au lieu de `Register`
- Méthodes statiques `GetXxx` et `SetXxx`
- Pas de wrapper CLR sur la classe

> Idéal pour ajouter des comportements réutilisables

Partie 4

Routed Events

Concept des Routed Events

Événements qui **se propagent** dans l'arbre visuel

3 stratégies :

Stratégie	Direction	Préfixe
Tunneling	Racine → Source	`Preview*`
Bubbling	Source → Racine	*(sans)*
Direct	Source uniquement	-

Tunneling et Bubbling

```
        Window          ← Bubbling (MouseDown)
          |
      StackPanel        ← ...
          |
        Button          ← ...
          |
      TextBlock         ← [CLIC] Source

Tunneling (PreviewMouseDown) →
```

“ `e.Handled = true` arrête la propagation ”

Propriétés de l'événement

Propriété	Description
`Source`	Élément qui a déclenché
`OriginalSource`	Élément le plus profond
`RoutedEvent`	Identifiant de l'événement
`Handled`	Arrêter la propagation

“ 📂 **Démo** : `Demo04-RoutedEvents` ”

Cas d'usage du Tunneling

Preview* permet d'**intercepter** avant le traitement :

- Validation de saisie (bloquer caractères)
- Raccourcis clavier globaux
- Logging/Audit
- Annulation d'actions

> Le tunneling arrive **avant** le bubbling

Points clés - Partie 1

- ✅ **Arbre Visuel** = rendu complet (avec templates)
- ✅ **Arbre Logique** = structure XAML conceptuelle
- ✅ **Dependency Properties** = binding, styles, animations
- ✅ **Ordre de résolution** = coercion > animation > local > style
- ✅ **Attached Properties** = étendre sans hériter
- ✅ **Routed Events** = propagation dans l'arbre

Questions ?

Pause avant la partie **Ressources et Styles**

Partie 5

Ressources et Styles

Système de ressources WPF

Ressource = objet réutilisable défini une fois

- Couleurs, brushes
- Styles, templates
- Converters
- Données statiques

“ Recherche **ascendante** dans l'arbre ”

StaticResource vs DynamicResource

Aspect	StaticResource	DynamicResource
Résolution	Une fois (chargement)	À chaque accès
Performance	✅ Plus rapide	❌ Plus lent
Mise à jour	❌ Non	✅ Oui
Forward ref	❌ Non	✅ Oui

> Préférer `StaticResource` sauf pour les thèmes

Organisation des ressources

Structure recommandée :

```
Resources/
├── Colors.xaml          # Couleurs et brushes
├── Typography.xaml      # Polices, styles texte
├── Controls/
│   ├── ButtonStyles.xaml
│   └── TextBoxStyles.xaml
└── Themes/
    ├── Light.xaml
    └── Dark.xaml
```

MergedDictionaries

Fusionner plusieurs dictionnaires :

- Ordre important : les derniers **écrasent** les premiers
- Ressources locales après les merged
- Permet la **modularité**

>  **Démo** : `Demo05-Styles-Triggers`

Styles : implicite vs explicite

Type	Déclaration	Application
Explicite	`x:Key="MonStyle"`	Manuelle
Implicite	`TargetType` seul	Automatique

BasedOn pour l'héritage de styles

> Un style implicite affecte **tous** les contrôles du type

Triggers

Type	Réagit à
Property Trigger	Propriété du contrôle
Data Trigger	Donnée bindée
Event Trigger	Événement (animations)
MultiTrigger	Plusieurs conditions (AND)

“ Les Triggers sont **réversibles** automatiquement ”

Partie 6

Templates

ControlTemplate

Redéfinit complètement l'apparence d'un contrôle

- Structure visuelle personnalisée
- Conserve le **comportement**
- `TemplateBinding` pour lier aux propriétés
- Respecter les `PART_*` pour fonctionnalités

>  **Démo** : `Demo06-Templates`

DataTemplate

Définit l'affichage d'un objet de données

Utilisation	Description
Implicite	`DataType="{x:Type local:Client}"`
Explicite	`x:Key="ClientTemplate"`
ItemTemplate	Dans les listes
ContentTemplate	Dans ContentControl

DataTemplateSelector

Choisir le template **dynamiquement** selon les données

- Hériter de `DataTemplateSelector`
- Override `SelectTemplate`
- Retourner le template approprié

> Utile pour affichage conditionnel (ex: messages chat)

ItemsControl et ItemsPanelTemplate

ItemsPanelTemplate définit le conteneur :

Panel	Disposition
`StackPanel`	Vertical/Horizontal
`WrapPanel`	Grille fluide
`UniformGrid`	Grille uniforme
`Canvas`	Position absolue
`VirtualizingStackPanel`	Optimisé grandes listes

Atelier Jour 1

Exercice 1 : Contrôle personnalisé

Créer un `RatingControl` (étoiles) avec template modifiable

Exercice 2 : Système de thèmes

Implémenter un switch thème clair/sombre dynamique

Durée : 1h30

Résumé Jour 1

Concept	À retenir
Arbres	Logique = XAML, Visuel = rendu
Dependency Properties	Binding, styles, coercion
Attached Properties	Étendre sans hériter
Routed Events	Tunneling puis Bubbling
Ressources	Static (perf) vs Dynamic (thèmes)
Templates	Control = apparence, Data = données

Fin du Jour 1

Demain : MVVM - Architecture et Implémentation

Pensez à explorer les démos !