Editions ENI

WPF

Développez des applications structurées (MVVM, XAML...)

(Nouvelle édition)

Collection
Expert IT

Extrait

Chapitre 5
DataBinding

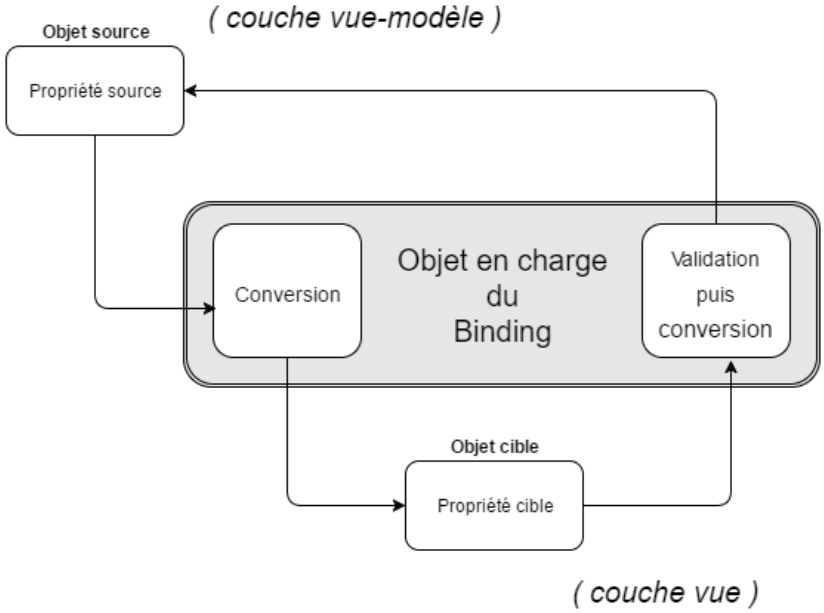
1. Introduction

Le *binding* ou le *binding de données*, que l'on pourrait traduire littéralement par « liaison de données », est un mécanisme absolument central en WPF. Ce mécanisme est d'ailleurs directement hérité de l'implémentation de MVVM par cette technologie.

S'il fallait le définir de façon très générale, le binding peut s'envisager comme le mécanisme qui lie deux sources de données et qui en assure la synchronisation mutuelle.

Ce mécanisme est au cœur de ce qui permet une réelle séparation entre vue et vue-modèle et à ce titre, est réellement le bras armé de MVVM dans le cadre d'un projet codé en WPF.

Ci-dessous, un schéma résumant le mécanisme de binding ainsi que les principaux objets impliqués.



En aucun cas le schéma précédent ne propose un cycle. Il s'agit bien là de mettre en exergue la synchronisation d'une propriété de la vue-modèle vers la vue et réciproquement.

Le schéma précédent utilise plusieurs dénominations qu'il s'agit ici de préciser :

– L'objet source situé dans la couche vue-modèle contient une propriété source qui fait l'objet d'une synchronisation *via* le binding.
– Côté vue, un objet, pas nécessairement visuel, contient une propriété cible qui elle aussi peut faire l'objet d'une synchronisation *via* le binding.
– L'objet Binding à même de réaliser le binding dans les deux sens.

1.1 Binding côté vue exclusivement

Côté vue, dans le code XAML, l'objet ou la propriété bindé utilise une expression « `Binding` » pour réaliser cette synchronisation.

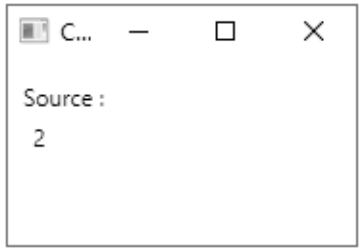
L'objet `Binding` utilise trois propriétés, `Source`, `RelativeSource` ou `ElementName` pour se lier avec l'objet source. À ce stade, il faut préciser que le binding ne se réalise pas nécessairement avec la vue-modèle. Classiquement, la vue-modèle va fréquemment être affectée au `DataContext` de la vue et le binding s'effectue alors préférentiellement avec la vue-modèle. Mais le binding peut également s'appliquer sans lien aucun avec la vue-modèle. Ce type de binding statique utilise l'une des trois propriétés explicitées ci-dessus.

1.1.1 Propriété Source

Cette propriété permet de spécifier un binding dont le chemin (`Path`) est totalement connu et ne dépend pas du `DataContext`. Par exemple ci-dessous, on affiche le mois courant (celui de la date du jour) dans un `TextBlock`.

```
<Window x:Class="CH5_BINDING1.MainWindow"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/
markup-compatibility/2006"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:local="clr-namespace:CH5_BINDING1"
        mc:Ignorable="d"
        Title="CH5_BINDING1 (Source)" Height="200" Width="200">

        <StackPanel Margin="10" Background="White">

            <TextBlock Text="Source :" />
            <TextBlock Margin="5"
            Text="{Binding Source={x:Static sys:DateTime.Today},
Path=Month}" />

    </StackPanel>
</Window>
```

On exécute ce programme au mois de février (deuxième mois de l'année) ce qui donne l'affichage suivant :



1.1.2 Propriété RelativeSource

Cette propriété est relative contrairement à la propriété précédente, c'est-à-dire qu'elle envisage comme contexte l'arbre visuel défini par le code XAML dans lequel le contrôle courant s'insère.

Dans cet arbre visuel, on peut rechercher des informations sur le contrôle courant (grâce au mot-clé `Self`) ou sur un ancêtre, une balise située plus haut dans l'arbre en quelque sorte, ceci grâce au mot-clé `FindAncestor`. Une dernière utilisation possible est de recourir au mot-clé `TemplatedParent` qui permet d'appliquer un binding aux éléments soumis à un template donné, par exemple définis *via* une balise `DataTemplate`.

L'exemple suivant reprend chacune des valeurs possibles de `RelativeSource` :

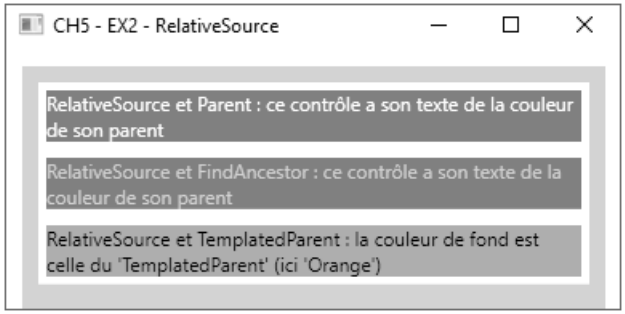
- `Self` : une première écriture voit sa couleur de texte identique à la couleur de fond de son parent.
- `FindAncestor` : une seconde écriture voit sa couleur de texte identique à la couleur de fond d'un ancêtre d'un type donné (`StackPanel`).
- `TemplatedParent` : une troisième écriture voit sa couleur de fond issue du contrôle qui accueille son *Template*.

Voici le code :

```
<Window x:Class="CH5_EX2_RELATIVESOURCE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/
markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX2_RELATIVESOURCE"
        mc:Ignorable="d"
        Title="CH5 - EX2 - RelativeSource" Height="200" Width="400">
    <StackPanel Margin="10,10,10,0"  Background="LightGray" Height="249"
VerticalAlignment="Top">

        <StackPanel Margin="10" Background="White">

            <TextBlock Margin="5"
            Background="Gray"
            Text="RelativeSource et Parent :
ce contrôle a son texte de la couleur de son parent"
            TextWrapping="Wrap"
            Foreground="{Binding RelativeSource={RelativeSource Self},
Path=Parent.Background}"/>

            <TextBlock Margin="5"
            Background="Gray"
            Text="RelativeSource et FindAncestor :
ce contrôle a son texte de la couleur de son parent"
            TextWrapping="Wrap"
            Foreground="{Binding RelativeSource={RelativeSource
FindAncestor, AncestorType={x:Type StackPanel}, AncestorLevel=2},
Path=Background}"/>

            <ItemsControl Background="Orange" Margin="5">
                <TextBlock TextWrapping="Wrap">RelativeSource et
TemplatedParent : la couleur de fond est celle du 'TemplatedParent'
(ici 'Orange')</TextBlock>
                <ItemsControl.ItemTemplate>
                    <DataTemplate>
                        <StackPanel>
                            <TextBlock Background="{Binding
RelativeSource={RelativeSource TemplatedParent}, Path=Background}"/>
                        </StackPanel>
                    </DataTemplate>
                </ItemsControl.ItemTemplate>
            </ItemsControl>
```

```
        </StackPanel>
    </StackPanel>

</Window>
```

Cela donne la fenêtre suivante :



Remarque

Lors de l'utilisation de `FindAncestor`, il est possible de spécifier le niveau d'ascendance auquel se trouve le contrôle « ancêtre » grâce à la propriété `AncestorLevel`.

1.1.3 Propriété ElementName

`ElementName` permet de référencer un élément de la vue par son nom. Ainsi, dans l'exemple d'école suivant, une première `TextBlock` affiche le nom d'une seconde `TextBlock` et réciproquement. Le référencement de chaque contrôle se fait via le nom de chacune d'elles.

```
<Window x:Class="CH5_EX3_ElementName.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/
markup-compatibility/2006"
        xmlns:local="clr-namespace:CH5_EX3_ElementName"
        mc:Ignorable="d"
        Title="CH5 - EX3 - ElementName" Height="350" Width="525">
    <StackPanel Margin="10">
        <TextBlock Margin="10" Name="Premier_TextBlock"
Text="{Binding ElementName=Second_TextBlock, Path=Name}" />
```

Editions ENI

MVVM

Maîtrisez vos développements .NET (WPF, Silverlight, Windows Phone...)

Collection
Epsilon

Extrait

Chapitre 3
MVVM pas à pas

1. Introduction

Il est possible d'étudier le patron MVVM en le décomposant. Une fois le rôle et la responsabilité de ses parties bien définis, il est possible de mettre en place les interactions pour ensuite assembler le tout et former le patron MVVM au complet.

2. Vue globale

MVVM se rapproche fortement des patrons MVC et MVP. Ceux-ci se chargent de tout ce qui concerne la vue, l'interaction avec l'utilisateur et le comportement général de l'interface graphique. La différence se situe dans le lien entre le modèle-vue et la vue. Le binding représente une dimension supplémentaire offerte aux développeurs. La forte flexibilité donnée par le développement d'interfaces en XAML facilite de plus la séparation entre la vue et le reste du modèle.

3. Le modèle

Le modèle représente généralement le conteneur de données. Il est important par son contenu mais aussi par sa structure. Le modèle est généralement représenté par des classes .NET simples ou encore **POCO** (*Plain Old C# Object*). Il est possible que ce que l'on appelle modèle soit représenté par une hiérarchie complexe de classes liées entre elles par composition. Cette organisation de classes constitue la structure du modèle. Quand elle n'est pas dynamique, elle est définie par le développeur et seul son contenu évoluera au cours de l'exécution du programme.

Le modèle ne contient généralement aucune intelligence. Pour ses données, un traitement métier a pu être effectué lors de sa récupération (filtrage, tri), lors de sa création ou encore lors de son exploitation dans le modèle-vue.

3.1 Rôle du modèle

Le modèle est défini en amont, avant même la création de l'interface. Il a pour rôle de structurer le métier de l'application et de contenir les données. Son rôle est aussi de pouvoir transiter de l'endroit où il est créé vers l'endroit où il doit être exploité.

La structure du modèle guidera le modèle-vue associé. Selon cette structure, le modèle-vue doit plus ou moins adapter les données afin de les retranscrire et de les fournir à la vue. Il n'est parfois pas essentiel de remonter la totalité de la structure. Une extraction partielle peut être suffisante afin de satisfaire l'application.

Les données du modèle sont aussi importantes : en fonction de leur quantité et/ou de leur profondeur, le modèle-vue doit les retravailler, les synthétiser ou encore les répartir entre les différents modèles-vues de l'application. Là encore, il peut être possible de ne récupérer qu'une partie du contenu en fonction de ce qui est exploitable. L'exemple le plus probant est la pagination. Il n'est nécessaire de récupérer qu'une sous-partie du modèle selon la quantité qui doit être affichée et l'index de départ représenté par la page affichée. Un autre exemple peut être une vue **Master-Detail**. Cette vue charge une liste d'objets qui, après sélection, est affichée en détail dans un autre contrôle. La liste d'objets peut être une liste de modèles dit légers pour optimiser la récupération.

3.2 Origine du modèle

Le modèle peut avoir plusieurs origines. Il peut venir d'une ressource embarquée dans l'application. La ressource est alors compilée avec l'exécutable. La ressource se retrouve physiquement intégrée à l'application et il est alors possible de la relire lors de l'exécution afin de remplir une instance de modèle avant son utilisation dans les différents modèles-vues.

Le modèle peut aussi venir d'une ressource contenue sur la machine qui exécute l'application. Le programme est alors responsable du chargement des ressources et de leur intégration dans une instance du modèle. Il est par exemple concevable de fournir une application s'appuyant sur un fichier XML qui contient la totalité des données nécessaires au bon déroulement de l'application. Il ne faut néanmoins pas oublier de traiter les problèmes liés à l'exploitation des données externes tels que l'indisponibilité de celles-ci ou encore leur non-conformité en terme de format.

Sur des applications de moyenne ou de grande dimension et afin de satisfaire aux contraintes architecturales de développement, le modèle peut venir d'une autre source appelée DAL (*Data Access Layer*). Cette couche architecturale est en charge de la récupération de données stockées dans un conteneur qui peut être une base de données. Cette DAL permet, selon certains critères, de créer et de remplir le modèle. Elle se charge aussi de le fournir, à la demande, au reste de l'application. L'intérêt d'une telle couche est l'intégration d'algorithmes complexes comme l'optimisation de la récupération, la mise en place d'un cache simple ou d'un cache distribué. Ces algorithmes peuvent être utilisés dans cette couche ou dans la couche qui est présente derrière la DAL, mais ils n'impactent en rien le format de retour de cette récupération.

Dans une application N-tiers utilisant des services web, le modèle peut être généré selon le contrat respecté par le service. Le service créant et remplissant le modèle doit alors le transcrire avant de le transmettre. Après création, le modèle est **sérialisé**, c'est-à-dire que chacune de ses propriétés est lue et écrite dans un flux qui pourrait être transmis par le biais d'un réseau. Les services web développés pour Windows Communication Foundation et déployés sous IIS ont plusieurs canaux de transports comme HTTP, TCP/IP et les canaux nommés. Après transmission et lors de la réception du flux, le modèle est alors **désérialisé** afin de le reconstituer et de le passer au modèle-vue.

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/Clock">
    <m:GetClock>
      <m:ClockValue>00h33m06s</m:ClockValue>
    </m:GetClock>
  </soap:Body>

</soap:Envelope>
```

Cet exemple de code XML est la réponse à un service web sérialisée et retournée par le serveur. Ce service répond à une demande d'heure de la part du client. La réponse est formatée afin d'être transportée et interprétée par le client.

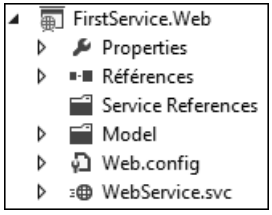
3.3 Exemple

Il est assez fréquent de trouver des applications utilisant des services WCF au sein d'un développement métier. Afin de démontrer la possibilité d'intégrer une couche de service dans la mise en place de MVVM, l'exemple suivant détaille la création et la consommation d'un service web basique dans une application WPF. Lors du référencement du service dans le client, un générateur produira une classe pour appeler le service et les classes correspondant aux données.

- Ouvrez Visual Studio et ajoutez deux projets. Le premier est un projet web vierge, il a pour rôle d'héberger le service web et le second est un projet WPF qui représente le client du service.
- Dans le projet web, ajoutez un dossier nommé **Model** et ajoutez une classe **Eleve** à l'image de la classe décrite précédemment.

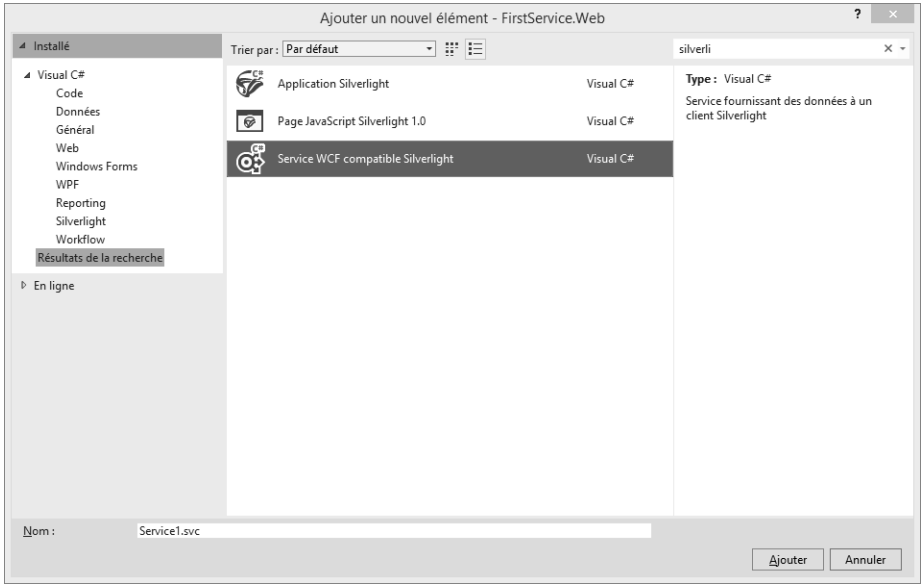
```
public class Eleve
{
    public string Nom { get; set; }
    public string Academie { get; set; }
}
```

La structure donnée en exemple est une classe **Eleve** exposant deux propriétés. Il est concevable d'avoir une structure bâtie sur une composition de classes pouvant même posséder des références circulaires. La classe en charge de la sérialisation des données gère les classes imbriquées et les références circulaires.



La prochaine étape consiste à créer un service qui retourne une liste d'élèves. La structure du modèle qui est utilisé dans la réponse ainsi que le descriptif de la méthode exposée sont tous les deux décrits dans les métadonnées du service.

- Dans le projet web, ajoutez un service web. Afin de faciliter la configuration de ce service et à des fins de démonstration, ajoutez un service web à destination des applications Silverlight.



- Alimentez votre service avec une méthode **GetEtudiants** retournant une liste d'élèves.

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed)]
public class WebService
{
    [OperationContract]
    public List<Eleve> GetEtudiants()
    {
        return new List<Eleve>()
            {
                new Eleve() {Nom = "Elève 1", Academie="Académie 1"},
                new Eleve() {Nom = "Elève 2", Academie="Académie 1"},
                new Eleve() {Nom = "Elève 3", Academie="Académie 1"}
            };
    }
}
```

- Comme décrit ci-dessus, le contenu du retour n'est pas essentiel pour cet exemple. Afin de valider qu'aucune erreur ne soit présente dans l'application web, compilez celle-ci. De cette manière vous rendez accessible le service auprès des potentiels projets clients en développement.

- Rendez-vous dans l'application WPF afin d'ajouter une référence de service. Cette option est accessible par le biais d'un clic droit sur l'application WPF.