# Notes sur le module : *WPF.*

Armel Pitelet

5 février 2022

## Table des matières



## 1 Bases

Mot clef : **https ://fr.wikipedia.org/wiki/Windows_Presentation_Foundation**.

Permet de créer des application en mode fenêtre.Pour utiliser WPF, soit on passe par du .Net Framework (ancienne version) soit on passe sur du .Net Core (nouvelle).

Remarque : *XAML.*

Language balise associé au WPF. C'est via XAML que l'on définit le design. Voir les différents layout ci dessous

Remarque : *Stack Panel.*

Layout qui permet de stacker plusieurs éléments les uns à la suite des autres.

Remarque : *Wrap panel.*

Identique au stack panel mais passe à la ligne suivante lorsque la ligne en cours est pleine (dynamiquement en fonction de la taille de l'écran contrairement au stack panel).

Remarque : *Dock panel.*

Permet de fixer un élément à un endroit donné avec une taille donnée.

Remarque : *Grid.*

Permet de créer des lignes et des colonnes avec des tailles possiblement relative les unes aux autres.

Remarque : *Canvas*.
Permet un placement libre des éléments.

Note :
Pour une intro à wpf : https://wpf-tutorial.com/.

Remarque : *Textbox*.
Permet de créer une boite de texte éditatble.

Remarque : *Textblock*.
Même chose que textblox mais le texte n'est pas éditable.

Remarque : *Combobox*.
Permet de faire une liste déroulante à possibilité de choix fixes.

Remarque : *Listbox*.
Même chose que combobox mais sous forme de liste et pas d'un menu déroulant.

Remarque : *Menu*.
Permet de faire des menus, avec potentiellement des sous menu inclus.

Note :
Attention par défaut les images ne sont pas embarquées dans l'application WPF. On les voits dans le designer mais pas dans l'application une fois celle ci lancée. Il faut changer le statut de l'image en "ressource de l'application" (dans les paramètres Propriétés/Avancé). Cela permet d'embarquer l'image avec le dll. On peut aussi la mettre en "Contenue". A ce moment l'image sera copiée (à la compilation) avec la même arborescence que l'application lors de la création.

Note :
Pour faire des gitignore facilement https://www.toptal.com/developers/gitignore

Note :
Pour de la génération de classe automatique https://github.com/AutoFixture et https://github.com/bchavez/Bogus.

Remarque : *Utiliser les ToString*.
Mauvaise pratique pour faire de l'affichage en WPF

# 2 Event et Binding

Remarque : *Sur les events*.
Les events sont en fait des implémentation du pattern observateur qui permettent de déclencher les évènement sur un changement d'état. POur ce sur quoi c'est basé Nico pour son cour voir : https://docs.microsoft.com/fr-fr/dotnet/standard/events/.

Remarque : *Sur le remplacant de WCF*.
Il s'agit de gRPC : google remote protocole col (service gRPC ASP.Net Core). S'appuie sur http 2 (sérialisation binaire). Très proche sur du wcf (basé sur SOAP : sérialisation xml) mais avec de bien

meilleur performance.

Remarque : *Sur le binding.*
Il est possible d'utiliser le Binding pour relier n'importe quel élément. Attention on ne bind que les propriétés, pas les attributs.

Remarque : *DataContext.*
Définit la source des bindings. (peut etre un objet fenêtre ou un objet de type manager, ou autre...).

Remarque : *ObservableCollection.*
Permet de définir des collections observables : permet de s'abonner par la suite pour suivre les changements.

Remarque : *Sur IList et IEnumerable.*
Si l'on veut faire la modif autant passer par le IList mais si l'on veut just eliste autant avoir un IEnumerable pour garder la généricité la plus haute possible.

Remarque : *Sur le mapping.*
Il ne fuat pas faire trop de profile de mapping, mais les grouper par grand groupes d'objets (et pas par objets).

Remarque : *Sur la notion de Model.*
Un ObjectModel (dédié à la vue) sert pour l'affichage : c'est encore différent d'un Dto(transfert sérialisation), d'une Entity (classe de bdd) ou d'un objet métier. Il peut contenir de la validation de données supplémentaire.

Remarque : *Sur les Entity.*
Les Entities peuvent contenir des méthodes.

Remarque : *Element Name.*
après un binding = qqchose (comme SelectedItem), ElementName=UnAutre élément () permet de binder à autre chose que le context courant (sur un autre élément graphique). On peut également changer le `mode` de binding (voir one way, two way, ...). `UpdateSourceTrigger` permet de changer l'élément déclencheur (PropertyChanged,...)

Remarque : *Pour les changement sur les property soient appliqué sans passer par un SelectedItem.*
On peut utiliser INotifyPropertyChanged dans nos modèles. Cela impose de définir un évènement handler PropertyChangedEventHandler. L'implémentation de la méthode associée impose de changer les propriétés classiques en attribut + propriété set qui track le changement (voir le UserModel dans le code de nico.). En fait SelectedItem a en interne ces propriété qui font que les changement dessus sont suivis.

Remarque : *L'annotation [CallerMemberName].*
devant un argument dans une méthode permet de récupérer le nom de la propriété qui appel cette méthode.

Remarque : *ILSPie.*
permet de récupérer du code décompiler à partir d'un .dll. Le fichier pdb permet de faire le lien entre le .dll et le code .cs

Remarque : *Lors d'un Binding : Trois questions à se poser.*

- 1 - Quel est le DataContext ? Si il n'y en a pas on remonte à l'élément du dessus. (dans le xaml, jusqu'au niveau de la fenêtre.). Ici on a set le DataContext au niveau de la main window.
- 2 - Quelle est la propriété source (itemsource ou à droite du binding), propriété cible (selecteditem ou à gauche du binding),
- 3 - Quel mode de binding utilisé (dans un sens, dans l'autre, dans les deux sens, property Mode=... dans un binding).

Remarque : *Exemple implicite du binding.*
Le textBlock Text = "{Binding Name}" est un binding implicite dans le context définies par itemsource.

Remarque : *Pour un équivalent du apt get-install.*
Voir ChocolaTey (présent dans node.js) pour un équivalent sous windows.

Note :
Voir Bogus pour un autre genre de AutoFixture.

Remarque : *Sur les converter.*
Il faut hériter de IValueConverter et décorer avec ValueConversion(source, destination). Il faut ensuite implémenter Convert (du modèle à la vue) et ConvertBack(de la vue vers le modèle). Dans les namespace (xlmns en début de fichier xaml), puis les ressources (Env Windows ressource) il faut ensuite inclure les converter. On peut ensuite utiliser le converter. Ces converters peuvent servir pour les images par exemple (convertion d'image) ou bien faire de la vérifiction de droit.

Remarque : *Sur la vue.*
On appel vue la partie xaml du wpf. Les donnéès provenant de la vue proviennent de la fenêtre. Voir la v2 du code de nico.

# 3 Sur lien wpf et api

Remarque : *Si l'on oublie les chaines deconnection.*
Il existe un site connection string qui répertorie tt les synatxes possiblent en fonction du type de bdd : https://www.connectionstrings.com/.

Remarque : *Sur les migrations avec CLI.*
`dotnet ef migrations add InitialCreate --projet chemin_vers_le.csproj` pour initialiser la première migration (il faut lui indiquer le chemin du projet ou est le DbContext). On doit aussi pouvoir faire -- output projetcible.csproj pour déplacer la migration ailleur. Pour faire la migration il est nécessaire d'avoir un constructeur qui prend un DbContextOptions.

Remarque : *Pour utiliser la chaine de connexion du appsettings.json pour la migration.*
Parait compliqué. On peut au mieux lui passer la chaine via le programme de l'API).

Remarque : *Sur l'update de la bdd.*
Soit on passe par les ligne de commande (via update) ou bien le configurer dans le program.cs de l'api. (voir la partie `app.Services.CreateScope()`)

Remarque : *Sur le `using` en inline.*

Inline `using` (IDisposable) des truc permet de créer un scope locale dans lequel les `trucs` seront détruit à la fin du scope. IDisposable est une interface qui implémente la méthode Dispose (qui permet de *refermer* des ressources (les rendre inaccessible jusqua ce que le Garbage collector les désaloues)). Ex : un DbContext est un disposable et pourrait être utilisé de cette manière.

Remarque : *Sur le Set du DbContext.*

Lorsque l'on fait un Context.Set<Type> on crée une représentation de la bdd. Si l'on fait un Where dessus on récupère juste un IQueryable sur la représentation. La requête vers la bdd n'est faite qu'à partir du moment ou l'on fait un ToList() ou un AsEnumerable() dessus.

Remarque : *Sur les retry de requête.*

Il faut utiliser des bibliothèques externes comme Polly qui permet d'implémenter un retry avec condition et nombre d'essaie avec ou sans attente.

Remarque : *Dans le cli.*

on peut faire dans un terminal `dotnet watch run` pour faire de la compilaytion automatique à chaque sauvegarde. Il faut être dans le dossier contenant le proejt (ou la solution → à tester.)

# 4 Sur les thèmes en wpf

Remarque : *Pour styliser un peu les applications.*

Il faut passer par des framework externe : **matérial-design**, ou bien **mahapps-metro**.

Remarque : *Sur material design.*

Il faut commencer par installer le themes via les Nugets (avec les dépendances indiquées sur le site). Puis dans l'app.xaml il faut compléter la partie ressource Dictionnary (dans application.Resources). Voir le site de material design (notemment la **démo**) et l'exemple de nico.

Remarque : *Une fois le style importé et configuré.*

On a accès à des propriétés propres aux thèmes (notemment via le champ Style).

Remarque : *Sur le changement de fenêtre de démarage.*

En changeant startupUri dans le xaml de l'app, on peut changer la fenêtre de démrage.

Remarque : *Sur l'élement visual state.*

L'élement VisualState manager permet de déclarer des éléments changeant dans le temps. (en taille, couleurs, positions...).

Note :

Voir rating bar pour une nottation avec des étoiles animées!!!.

# 5 Sur les user control

Remarque : *Pour lancer un build automatique.*

On peut dans le CLI faire un `dotnet watch run` dans le dossier du projet qui nous interesse.

Remarque : *Sur le DataContext dans un UC.*
Si il n'y a pas de DataContext dans le UC, alors le programme remonte les windows parents pour en trouver un.

Remarque : *Sur le Binding spécifique dans un UC.*
On peut nommer le UC dans le xaml (x :Name"LeNom") et dans le Binding on peut choisir l'élément à choisir via un ElementName.

Note :
Les classes sont partielles car déclarée dans le .xaml et dans le .cs. Sans le `partial` la vue ne fonctionnerai tt simplement pas !

Remarque : *Pour passer un objet d'une fenêtre à l'autre.*
Il faut déja nommé le user control dans son appel (dans la fenêtre parent)

Remarque : *Sur les dependancy property.*
DependancyProperty permet d'enregistrer (via Register(nom de la prop, type de la propriété, type du propriétaire)) une propriété qui sait se notifier soit même.

Note :
TT les propriétés attaché de base à la vue sont des DependancyProperty (celle à gauche d'un binding).

Note :
On peut faire Binding . ou le point design this, la fenêtre en cours.

Note :
On peut s'éviter de faire de la dupplication sur le OnNotifyPropertyChange en définissant une classe de laquelle les autres UserControl dérivent (Au lieu de dériver directement de User control)

Remarque : *Sur cette manière de faire les user controls.*
Cela implique de pouvoir réutiliser des user control dans d'autres mais fait comme sa on réutilise et le code behind et le xaml associé.

Remarque : *Pour faire autrement.*
Il faut passer par un DataTemplate (que l'on doit nommer via x :key=) à l'intérieur des Ressources (UserControl.Resource ou Application.Ressource). Plus bas dans le xaml on peut ensuite déclarer un ContentControl avec l'attribut ContentTemplate = {StaticRessource nomDuTemplate} pour réutiliser cette partie de xaml.

Note :
Dans le template on peut également passer un DataType = {x :type TypeName} et le réutiliser dans un Datatemplate plus bas

Note :
Utiliser un DataTemplate permet de ne pas avoir à utiliser la logique, on ne partage que du xaml.

Note :
Pour qq exemple : https://www.softfluent.fr/blog/les-datatemplate-wpf-partie-1/.

# 6 La navigation

Le navigateur permet de centraliser les vue navigables et de gérer les transitions entre chaque. Il faut commencer par définir une interface pour le navigateur.

Remarque : *Sur l'utilisation.*
On instancie un navigateur et on utilise sa méthode RegisterView pour enregistrer les vue utilisable par le navigateur.

Note :
En WPF les Control sont tt ce qui est objet visuel (usercontrol, bouton, fenêtre).

Remarque : *Sur lélément stack.*
On passe par ce type de collection pour faciliter la navigation. Le stack est un Last In, Firt Out.

Remarque : *Dans la main window.*
On utilise le Navigator dans la main windows pour définir la première page via un Navigateto(typeof(MaPag)).

Note :
Il existe des frameworks (MVVM) qui implémentent nativement ce genre de navigateur.

# 7 Les bibliothèques de controles personnalisés WPF

C'est un type de projet que l'on peut créer lors de l'ouverture d'un nouveau projet WPF.
Permet de faire des composant partageable entre différentes applications.
Cela permet cette fois ci de partager une logique commune entre différent projet (ou bien component).

Note :
Dans le folder theme on a une dicitonnaire de ressource qui permet de définir les styles associées aux composants.

Remarque : *Sur l'importation.*
Il faut ensuite importer ce dictionnaire de ressource dans celui de l'application qui veut utiliser la ressource.

Note :
Ne surtout pas fiare de binding sur une `PassWordBox` (ne pas monter d'infos sensibles dans la mémoire).