



Éditeur de notes C# et XAML

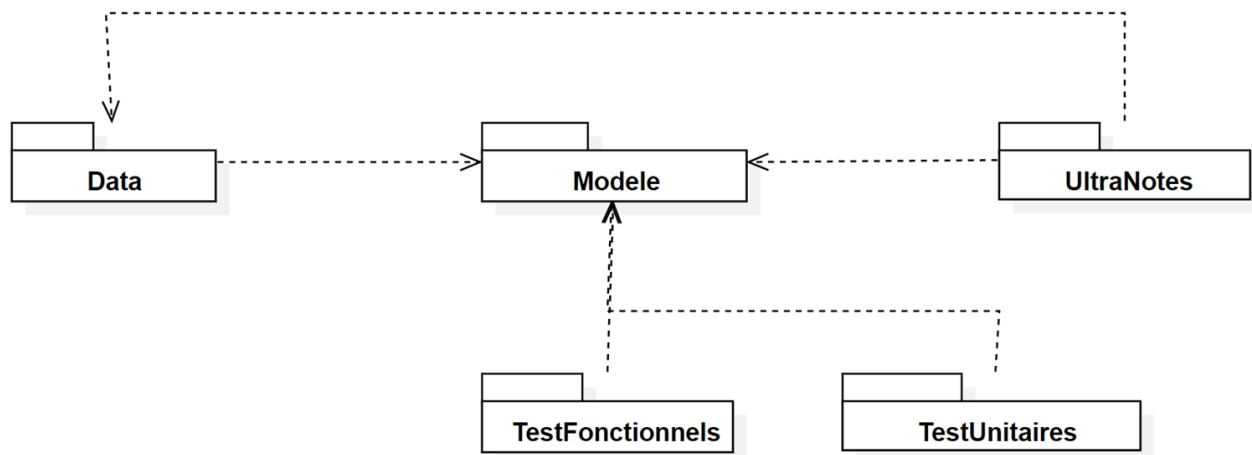
Thomas DEVIENNE 4H
BIZET Matthias 3H

26 avril 2021

Sommaire

...

Diagramme de paquetage



Le diagramme de paquetage est composé de plusieurs éléments. La première raison de séparer le code de l'application en plusieurs paquets est de pouvoir les rendre interchangeables. En effet, nous développons actuellement une application de bureau. Cependant, peut-être que par la suite nous souhaiterons porter l'application sur appareils mobiles, ou encore sur le web. Pour faire cela, inutile de refaire l'application de zéro : il suffit de remplacer le paquetage UltraNotes puis de réutiliser les autres paquets dans notre application. En effet, chaque paquetage est indépendant des autres et peut être remplacé par un autre. Encore, si nous souhaitons améliorer la partie persistance de notre application (sauvegarde des données), il nous suffit de remplacer le paquetage Data par un autre (exemple utilisation d'une base de données plutôt que des fichiers binaires). Notre application est composée des paquets ci-dessous.

Les **bibliothèques de classes** sont des « dossiers » contenant exclusivement des classes. Modele définit toutes les classes concernées uniquement par la logique de l'application. Ainsi, les collections de notes, les propriétés décrivant ces objets et la classe Manager sont définies dans ce paquetage. La classe Manager permet d'effectuer toutes les actions de notre application. Son but est de simplifier l'accès à tout notre sous-système de classes dans le Modele mais ne les interdit pas pour autant. En effet, nous pouvons toujours accéder à des méthodes de classes « derrière » Manager pour faire du Binding par exemple. Manager joue donc le rôle de façade car il fait le pont entre la vue et le Modele, entre l'application et son fonctionnement interne. Ainsi, elle se contente d'appeler les méthodes des autres classes dans ces méthodes. Cela permet une meilleure compréhension du programme et facilite aussi l'utilisation de notre application. En effet, il n'est pas nécessaire de connaître toutes les méthodes de toutes les classes du Modele pour utiliser notre application, seulement celles de Manager. Encore, Data est une seconde bibliothèque de classe qui est chargée de la sérialisation (sauvegarder un objet) de l'objet Bouquin. Bouquin est une classe qui regroupe une collection de Note et les gère. Il n'est pas nécessaire de sérialiser la classe Manager car elle ne fait que déléguer le travail à la classe Bouquin, qui contient toutes les données utiles à savoir les notes et les paramètres de l'application. C'est donc la classe Bouquin qu'il faut sérialiser. Toutes les bibliothèques de classes sont en .NET Standard 2.1.

Le paquetage TestFonctionnels est une **application console** qui sert à tester les fonctionnalités de l'application. C'est dans ce paquetage que nous testons les classes qui gèrent les fonctionnalités de l'application, décrites dans le diagramme de cas d'utilisation (créer une note, supprimer une note, etc.).

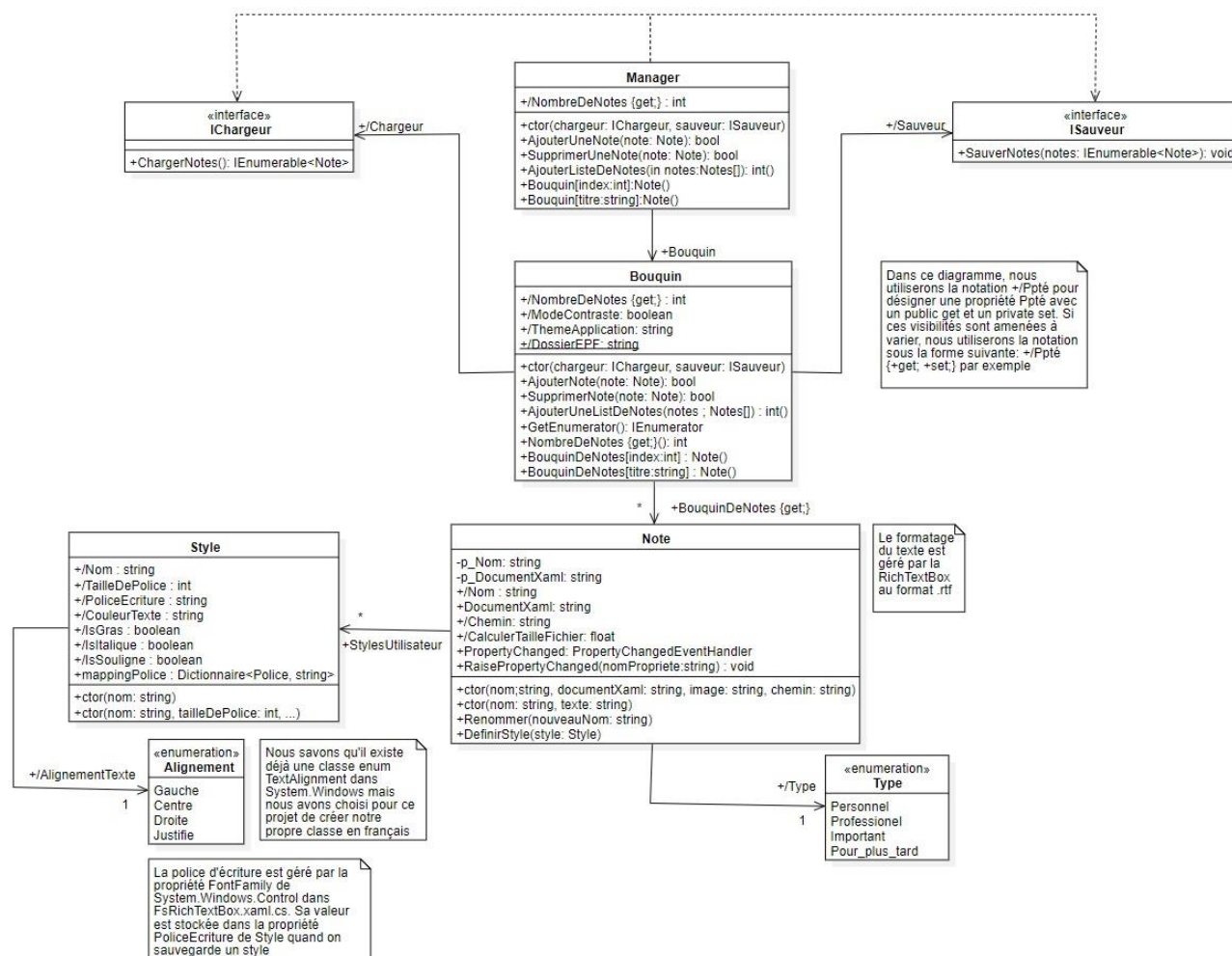
TestUnitaires est un paquetage de **tests Xunit**. Dans ce paquetage, nous testons les méthodes de nos classes avec des tests unitaires.

Enfin, UltraNotes est une **application WPF .NET Core 3.0**. Dans ce paquetage sont organisées en sous-dossiers les vues de l'application (la vue principale et les user control) ainsi que les classes qui y sont associés et une classe qui sert à convertir un objet de type FlowDocument en fichier de balisage XML.

On remarque d'ailleurs que tous les paquages dépendent de Modele.

Diagramme de classe

1. Diagramme de classe de Modèle



La classe **Manager** gère les fonctionnalités de l'application. Cette classe possède une instance de la classe **Bouquin** et la manipule en appelant ces méthodes. Cela permet à l'utilisateur de seulement instancier dans son code une instance de la classe **Manager** pour manipuler un **Bouquin**, sans avoir à connaître toutes les méthodes de cette dernière. La classe **Manager** possède une propriété calculée **NombreDeNotes** qui renvoie le nombre de notes dans le **Bouquin**. Les méthodes d'ajout et de suppression de notes permettent de fournir et gérer le **Bouquin**. Les indexeurs permettent d'accéder aux éléments du **Bouquin** depuis **Manager** en utilisant la notation `manager[index]`. Le constructeur de la classe nécessite une instance **IChargeur** et **ISaveur**. Ces deux interfaces servent à charger les données dans le **bouquin** et les sérialiser. Le **bouquin** de la classe **Manager** sera ainsi instancié avec ces objets. Ceci permet de pouvoir manipuler **manager** comme si c'était un **Bouquin**, mais en gardant les avantages d'une façade.

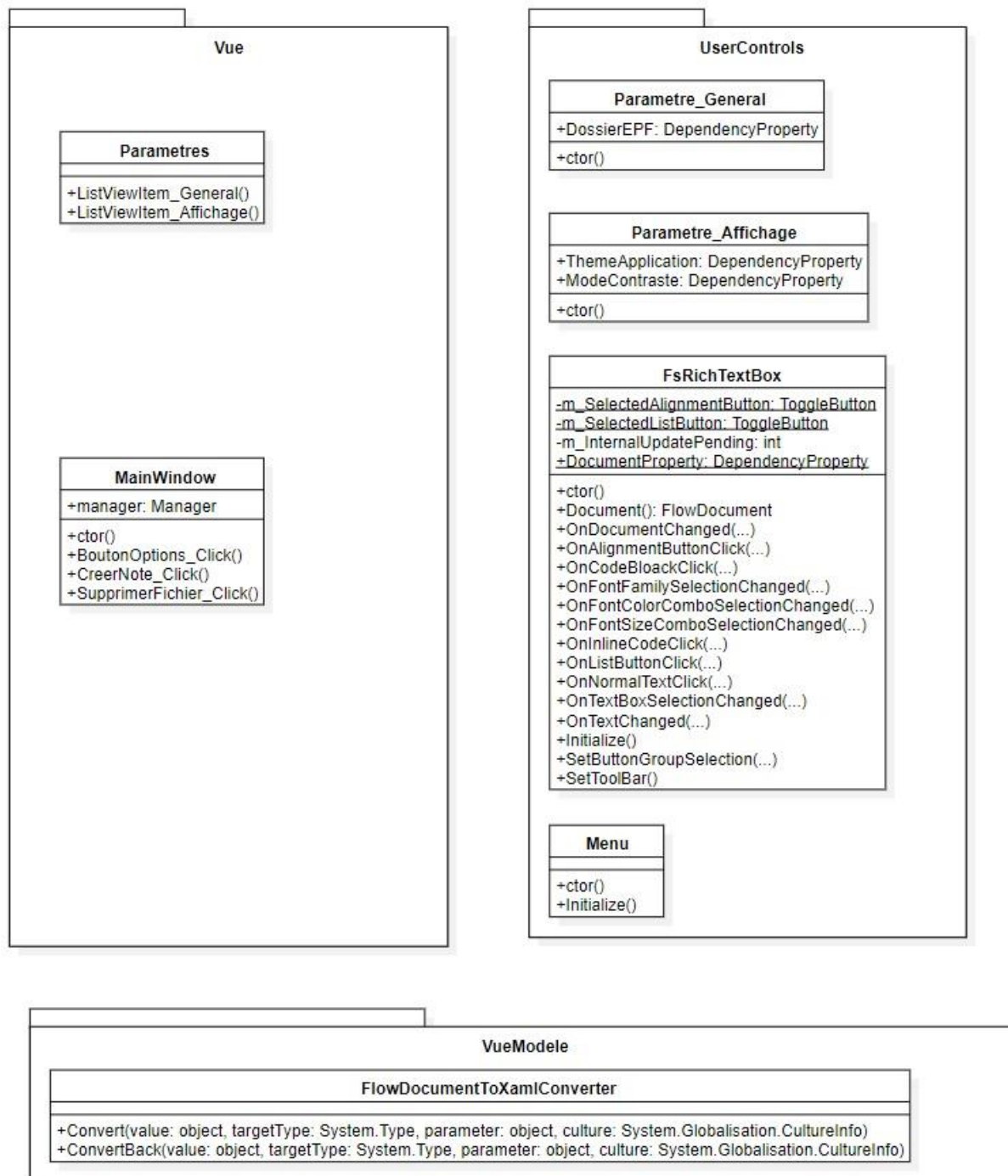
La classe **Bouquin** gère une collection de notes. Elle possède donc une propriété calculée qui renvoie le nombre de notes dans le **Bouquin** et de quelques paramètres utilisateurs à sérialiser. La propriété **ModeContraste** précise si le thème colorimétrique de l'application doit être accentué ou non. Cette option est utile pour les personnes peine à lire des textes écrit en blanc sur les fonds colorés par exemple. L'application est donc en noir et blanc (sauf dans la partie éditeur de texte). **ThemeApplication** définit la couleur dominante

de l'application, par défaut le bleu. DossierEPF (EPF pour Dossier d'Enregistrement Par Défaut) est le dossier dans lequel les notes vont être enregistrées par défaut si l'utilisateur ne précise pas de chemin. Ces paramètres sont stockés dans la classe Bouquin car ils font partis des réglages utilisateurs et sont donc propres à son « plan de travail ». Cette propriété est statique pour pouvoir y avoir accès sans forcément créer une instance de Bouquin. Ensuite, la classe Bouquin possède des méthodes qui permettent d'ajouter et de supprimer des notes pour fournir et gérer le Bouquin, ainsi que des indexeurs pour retrouver les informations rapidement dans la collection BouquinDeNotes qui contient donc la collection de Notes. La classe implémente l'interface IEnumerable pour pouvoir être parcourue. Elle redéfinit sa méthode GetEnumerator(). Enfin, le constructeur prend en paramètre deux objets : un de type IChargeur et l'autre de type ISauveur. Ces objets permettent aux méthodes ChargerBouquin() et SauverBouquin() de désérialiser et sérialiser l'objet. La méthode de sérialisation et désérialisation dépend du type d'instance des propriétés Chargeur et Sauveur, ce qui permet à un utilisateur de pouvoir changer très facilement de méthode de sauvegarde et récupération de données. En effet, il lui suffit de changer le type d'instance des paramètres lors de l'instanciation d'un Manager ou d'un Bouquin.

La classe **Note** définit donc une note. Une note possède un Nom donné par la propriété Nom, un contenu qui est la représentation de la note en XML (la note est un FlowDocument) donné par DocumentXaml et un chemin sur le disque donné par Chemin. Une propriété calculée permet de calculer la taille du fichier. Les champs p_Nom et p_DocumentXaml sont les variables associées aux propriétés du même nom. Ces deux propriétés envoient une notification lorsque leur contenu change pour notifier la vue de leur évolution. Une note possède également une liste de styles utilisateur qui lui est propre. Un Style est un ensemble de paramètres qui définissent la mise en forme d'un texte. Par exemple, un titre peut être défini comme écrit en gras, en police Cantarell en taille 15. La classe Note dispose également d'une propriété Type qui définit son type. Un Type définit quel est le type de document. La note peut avoir été rédigée pour le travail ou à titre personnel par exemple. Ces types sont cumulables. Les méthodes de Note permettent de la renommer et de définir un style. Note implémente l'interface INotifyPropertyChanged qui permet de notifier la vue que la valeur d'une propriété dans la classe a changé.

Enfin, la classe **Style** définit un style utilisateur. Les différentes propriétés de la classe représentent les différents paramètres de mise en page à savoir : la taille de police, la police d'écriture, la couleur du texte, savoir si le texte doit être mis en gras, en italique, et/ou en souligné ainsi que l'alignement du texte. Ce dernier paramètre est donné par une classe d'énumération qui définit qu'un texte peut être aligné à droite, au centre, à gauche ou en justifié (le texte est aligné à gauche et à droite). Les alignements ne sont pas cumulables. Le but de mettre tous ces paramètres dans des propriétés est d'une part de bénéficier du système de notifications de ces dernières (pour informer la vue que leur valeur a changé, pour le binding). Nous plaçons ces paramètres dans le Modele car un style doit être sérialisé.

2. Diagramme de classe de UltraNotes



Description à venir...