|  |
| --- |
|  |
| Petits pains |
| Application de rapport pour les CRA |
| *Spécification technique* |

|  |
| --- |
| Benoit Masson-Bedeau  24/10/2017 |

Table des matières

[1 Introduction 7](#_Toc497323885)

[2 Dossier Assets 7](#_Toc497323886)

[3 Dossier Command 7](#_Toc497323887)

[3.1 Fichier CommandHandler.cs 7](#_Toc497323888)

[3.2 Fichier CommandHandlerT.cs 7](#_Toc497323889)

[4 Dossier Converter 7](#_Toc497323890)

[4.1 Fichier BoolToVisibilityConverter.cs 8](#_Toc497323891)

[4.2 Fichier NullToBoolConverter.cs 8](#_Toc497323892)

[4.3 Fichier NullToVisibilityConverter.cs 8](#_Toc497323893)

[4.4 Fichier StringNullOrEmptyToBoolConverter.cs 8](#_Toc497323894)

[4.5 Fichier StringNullOrEmptyToVisibilityConverter.cs 8](#_Toc497323895)

[5 Dossier Data 9](#_Toc497323896)

[5.1 Fichier PetitsPainsStore.cs 9](#_Toc497323897)

[5.1.1 Propriétés 9](#_Toc497323898)

[5.1.2 Méthodes ReadConfig/WriteConfig 9](#_Toc497323899)

[5.1.3 Méthodes WriteCroissantsLines/ReadCroissantsLines 9](#_Toc497323900)

[5.1.4 Méthode ReadPersons 10](#_Toc497323901)

[5.1.5 Méthode GetDefaultPeople 10](#_Toc497323902)

[6 Dossier Resources 10](#_Toc497323903)

[6.1 Fichier EmailTemplate.tt 10](#_Toc497323904)

[6.1.1 Configuration 10](#_Toc497323905)

[6.2 Fichier EmailTemplate.cs 10](#_Toc497323906)

[6.3 Fichier EmailTemplateExtended.cs 10](#_Toc497323907)

[6.3.1 Champs 11](#_Toc497323908)

[6.3.2 Constructeurs 11](#_Toc497323909)

[7 Dossier Utils 11](#_Toc497323910)

[7.1 Fichier Enums.cs 11](#_Toc497323911)

[7.1.1 SortDirection 11](#_Toc497323912)

[7.2 Fichier Extensions.cs 11](#_Toc497323913)

[7.2.1 Méthode BubbleSort<T>(this ObservableCollection<T> collection) where T : IComparable<T> 11](#_Toc497323914)

[7.2.2 Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T> 12](#_Toc497323915)

[7.3 Fichier ItemsChangeObservableCollection.cs 12](#_Toc497323916)

[8 Dossier Model 12](#_Toc497323917)

[8.1 Fichier Person.cs 12](#_Toc497323918)

[8.1.1 Interfaces 12](#_Toc497323919)

[8.1.2 Propriétés 12](#_Toc497323920)

[8.1.3 Évènements 13](#_Toc497323921)

[8.1.4 Constructeur par défaut 13](#_Toc497323922)

[8.1.5 Constructeur Person(string firstName, string lastName) 13](#_Toc497323923)

[8.1.6 Constructeur Person(string firstName, string lastName, string email) 13](#_Toc497323924)

[8.1.7 Constructeur Person(string firstName, string lastName, string persoId, string email) 13](#_Toc497323925)

[8.1.8 Constructeur Person(SerializationInfo info, StreamingContext context) 13](#_Toc497323926)

[8.1.9 Méthode ToString 13](#_Toc497323927)

[8.1.10 Méthode GetObjectData(SerializationInfo info, StreamingContext context) 13](#_Toc497323928)

[8.1.11 Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context) 14](#_Toc497323929)

[8.1.12 Méthode CompareTo(Person other) 14](#_Toc497323930)

[8.1.13 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 14](#_Toc497323931)

[8.1.14 Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null) 14](#_Toc497323932)

[8.2 Fichier Croissant.cs 14](#_Toc497323933)

[8.2.1 Interfaces 14](#_Toc497323934)

[8.2.2 Enum CroissantState 15](#_Toc497323935)

[8.2.3 Propriétés 15](#_Toc497323936)

[8.2.4 Évènements 15](#_Toc497323937)

[8.2.5 Constructeur par défaut 15](#_Toc497323938)

[8.2.6 Constructeur Croissant(DateTime? date) 15](#_Toc497323939)

[8.2.7 Constructeur Croissant(SerializationInfo info, StreamingContext context) 15](#_Toc497323940)

[8.2.8 Méthode ToString 16](#_Toc497323941)

[8.2.9 Méthode GetObjectData(SerializationInfo info, StreamingContext context) 16](#_Toc497323942)

[8.2.10 Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context) 16](#_Toc497323943)

[8.2.11 Méthode CompareTo(Croissant other) 16](#_Toc497323944)

[8.2.12 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 16](#_Toc497323945)

[8.2.13 Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null) 17](#_Toc497323946)

[8.3 Fichier Line 17](#_Toc497323947)

[8.3.1 Interfaces 17](#_Toc497323948)

[8.3.2 Propriétés 17](#_Toc497323949)

[8.3.3 Évènements et delegates 18](#_Toc497323950)

[8.3.4 Constructeur par défaut 18](#_Toc497323951)

[8.3.5 Constructeur Line(int croissantsSlots) 18](#_Toc497323952)

[8.3.6 Constructeur Line(SerializationInfo info, StreamingContext context) 18](#_Toc497323953)

[8.3.7 Méthode GetPenaltiesCount 18](#_Toc497323954)

[8.3.8 Méthode GetPenaltiesLimit 19](#_Toc497323955)

[8.3.9 Méthode GetNextIndexToDeactivate 19](#_Toc497323956)

[8.3.10 Méthode GetLastDeactivatedIndex 19](#_Toc497323957)

[8.3.11 Méthode AddPenalty(DateTime date) 19](#_Toc497323958)

[8.3.12 Méthode RemovePenalty(DateTime date) 19](#_Toc497323959)

[8.3.13 Méthode ProcessPenaltyAdding(DateTime date) 20](#_Toc497323960)

[8.3.14 Méthode PenaltiesToAdd(DateTime date) 20](#_Toc497323961)

[8.3.15 Méthode CheckFoutageDeGueule(DateTime date) 20](#_Toc497323962)

[8.3.16 Méthode FirstDayOfProcessedWeek(DateTime date) 21](#_Toc497323963)

[8.3.17 Méthode ReactivatedCroissant 21](#_Toc497323964)

[8.3.18 Méthode GetIndexOfCroissantToReactivate 21](#_Toc497323965)

[8.3.19 Méthode GetObjectData(SerializationInfo info, StreamingContext context) 21](#_Toc497323966)

[8.3.20 Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context) 22](#_Toc497323967)

[8.3.21 Méthode HandlePersonChanged(object sender, EventArgs e) 22](#_Toc497323968)

[8.3.22 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 22](#_Toc497323969)

[8.3.23 Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null) 22](#_Toc497323970)

[9 Dossier ViewModel 22](#_Toc497323971)

[9.1 Fichier IClosable 22](#_Toc497323972)

[9.1.1 Méthode Close 22](#_Toc497323973)

[9.2 Fichier IDialog 23](#_Toc497323974)

[9.2.1 Interface 23](#_Toc497323975)

[9.2.2 Propriété 23](#_Toc497323976)

[9.3 Fichier ViewModelBase 23](#_Toc497323977)

[9.3.1 Évènements et delegates 23](#_Toc497323978)

[9.3.2 Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null) 23](#_Toc497323979)

[9.3.3 Méthode SetProperty<T>( Action checkCommands, ref T storage, T value, [CallerMemberName]string propertyName = null) 23](#_Toc497323980)

[9.4 Fichier AddPersonViewModel 23](#_Toc497323981)

[9.4.1 Classes héritées 24](#_Toc497323982)

[9.4.2 Propriétés 24](#_Toc497323983)

[9.4.3 Commands 24](#_Toc497323984)

[9.4.4 Constructeurs AddPersonViewModel(ItemsChangeObservableCollection<Line> lines) 24](#_Toc497323985)

[9.4.5 Méthode AddPerson(IDialog window) 24](#_Toc497323986)

[9.4.6 Méthode AddPersonCanExecute 25](#_Toc497323987)

[9.4.7 Méthode Cancel(IDialog window) 25](#_Toc497323988)

[9.4.8 Méthode CheckCommands 25](#_Toc497323989)

[9.5 Fichier UpdatePersonViewModel 25](#_Toc497323990)

[9.5.1 Classes héritées 25](#_Toc497323991)

[9.5.2 Propriétés 25](#_Toc497323992)

[9.5.3 Champs 25](#_Toc497323993)

[9.5.4 Commands 26](#_Toc497323994)

[9.5.5 Constructeur UpdatePersonViewModel(Line line) 26](#_Toc497323995)

[9.5.6 Méthode ValidateUpdatePerson(IDialog window) 26](#_Toc497323996)

[9.5.7 Méthode CanValidateUpdatePerson 26](#_Toc497323997)

[9.5.8 Méthode Cancel(IDialog window) 26](#_Toc497323998)

[9.5.9 Méthode PersonChanged 26](#_Toc497323999)

[9.5.10 Méthode HandlePersonPropertyChanged(object sender, EventArgs e) 27](#_Toc497324000)

[9.5.11 Méthode CheckCommands 27](#_Toc497324001)

[9.6 Fichier MainWindowViewModel 27](#_Toc497324002)

[9.6.1 Classes héritées 27](#_Toc497324003)

[9.6.2 Propriétés 27](#_Toc497324004)

[9.6.3 Commands 28](#_Toc497324005)

[9.6.4 Constructeur par défaut 28](#_Toc497324006)

[9.6.5 Méthode HandlePenaltyAlreadyExistsAtThisDate(object sender, EventArgs e) 29](#_Toc497324007)

[9.6.6 Méthode HandleLinePropertyChanded(object sender, EventArgs e) 30](#_Toc497324008)

[9.6.7 Méthode HandleLinesCollectionChanged(object sender, EventArgs e) 30](#_Toc497324009)

[9.6.8 Méthode SelectRootPathFolder 30](#_Toc497324010)

[9.6.9 Méthode LoadAfterCheck 30](#_Toc497324011)

[9.6.10 Méthode UpdatePath 30](#_Toc497324012)

[9.6.11 Méthode ManageFilesOnPathChanged 30](#_Toc497324013)

[9.6.12 Méthode Load 31](#_Toc497324014)

[9.6.13 Méthode AddPenalty(Line line) 31](#_Toc497324015)

[9.6.14 Méthode RemovePenalty 31](#_Toc497324016)

[9.6.15 Méthode CanRemovePenalty 31](#_Toc497324017)

[9.6.16 Méthode ActivateCroissant 31](#_Toc497324018)

[9.6.17 Méthode CanActivateCroissant 31](#_Toc497324019)

[9.6.18 Méthode RemoveLine(Line line) 32](#_Toc497324020)

[9.6.19 Méthode UpdateLine(Line line) 32](#_Toc497324021)

[9.6.20 Méthode AddLine 32](#_Toc497324022)

[9.6.21 Méthode EmailSituation 32](#_Toc497324023)

[9.6.22 Méthode CheckChosenFolder 33](#_Toc497324024)

[9.6.23 Méthode Save 33](#_Toc497324025)

[9.6.24 Méthode CheckCommands 33](#_Toc497324026)

[10 Dossier View 33](#_Toc497324027)

[10.1 Window AddPersonWindow 33](#_Toc497324028)

[10.1.1 AddPersonWindow.xaml.cs 34](#_Toc497324029)

[10.1.2 AddPersonWindow.xaml 34](#_Toc497324030)

[10.2 Window UpdatePersonWindow 34](#_Toc497324031)

[10.2.1 UpdatePersonWindow.xaml.cs 34](#_Toc497324032)

[10.2.2 UpdatePersonWindow.xaml 35](#_Toc497324033)

[10.3 MainWindow 35](#_Toc497324034)

[10.3.1 MainWindow.xaml.cs 35](#_Toc497324035)

[10.3.2 MainWindow.xaml 35](#_Toc497324036)

Historique

|  |  |  |
| --- | --- | --- |
| Date | Version | Commentaire |
| 24/10/2017 | 1.0 | Création du document |

# Introduction

L’application *Petits Pains* consiste en une application permettant de saisir le rapport pour les CRA des personnes d’une équipe puis d’envoyer un email à ces derniers pour les informer de la synthèse.

Le tout, dans la joie et la bonne humeur : on n’est pas la Stasi 😊 !

# Dossier Assets

Dossier des ressources à utiliser dans le projet. Il contient :

* le fichier *credit.txt* est le fichier contenant un lien pointant vers l’auteur de l’image *croissant\_empty* ;
* les fichiers images sont utilisés pour représenter les statuts des emplacements.

# Dossier Command

Dossier dans lequel se trouve les classes de commande. Les classes de commande permettent de réaliser des *bind* sur des éléments de l’interface graphique, comme des boutons.

## Fichier CommandHandler.cs

*Command* qui implémente l’interface *ICommand*. Elle permet d’exécuter une fonction qui ne prend pas de paramètre en fonction d’une autre fonction qui détermine si la première fonction peut s’exécuter.

Le seul constructeur prend donc 2 paramètres :

* *methodToExecute* de type *Action* : méthode à exécuter ;
* *methodToDetermineCanExecute* de type *Func<bool>*: méthode permettant de déterminer si *methodToExecute* peut être exécutée.

## Fichier CommandHandlerT.cs

Même principe que **Fichier CommandHandler**. La seule différence est que *methodToExecute* prend un paramètre en entrée.

Le constructeur prend donc ces 2 paramètres en entrée :

* *methodToExecute* de type *Action<T>* : méthode à exécuter ;
* *methodToDetermineCanExecute* de type *Func<bool>*: méthode permettant de déterminer si *methodToExecute* peut être exécutée.

# Dossier Converter

Dossier contenant les *Converter*, permettant de convertir des objets d’un type particulier vers un type manipulable par l’interface graphique (et vice-versa).

## Fichier BoolToVisibilityConverter.cs

Converti un booléen vers un objet de type *Visibility* :

* booléen vaut *true* : *Visibility* en sortie vaut *Visible* ;
* sinon : *Visibility* en sortie vaut *Collapsed*.

Ceci permet de piloter la visibilité d’éléments de l’interface graphique grâce à un booléen.

## Fichier NullToBoolConverter.cs

Converti un objet vers un booléen.

Dans cette classe, un enum *NullToBoolDirection* est présent, et permet de déterminer le sens de conversion :

* *NullIsTrue* : un objet *null* doit être converti vers une valeur *true* ;
* *NullIsFalse* : un objet *null* doit être converti vers une valeur *false*.

La valeur de l’enum doit être passée dans le paramètre *Parameter*. La sortie est donc conforme à ce paramètre.

Si le paramètre est *null*, on considère que *NullIsFalse*.

## Fichier NullToVisibilityConverter.cs

Converti un objet vers un objet de type *Visibility*.

Dans cette classe, un enum *NullToVisibilityDirection* est présent, et permet de déterminer le sens de conversion :

* *NullIsVisible* : un objet *null* doit être converti vers une valeur *Visible* ;
* *NullIsCollapsed* : un objet *null* doit être converti vers une valeur *Collapsed*.

La valeur de l’enum doit être passée dans le paramètre *Parameter*. La sortie est donc conforme à ce paramètre.

Si le paramètre est *null*, on considère que *NullIsCollapsed*.

## Fichier StringNullOrEmptyToBoolConverter.cs

Converti une chaine de caractère *null* ou vide vers un booléen :

* la chaine est *null* ou vide : la valeur en sortie est *false* ;
* sinon : la valeur en sortie est *true*.

## Fichier StringNullOrEmptyToVisibilityConverter.cs

Converti une chaine de caractère *null* ou vide vers un objet de type *Visibility*:

* la chaine est *null* ou vide : la valeur en sortie vaut *Collapsed* ;
* sinon : la valeur en sortie vaut *Visible*.

# Dossier Data

Dossier contenant les classes permettant la manipulation des données.

## Fichier PetitsPainsStore.cs

Fichier permettant la manipulation des données de l’application.

Il s’agit d’une classe *static* dans laquelle se trouvent les méthodes permettant de :

* stocker les données ;
* lire les données.

### Propriétés

* *RootPath* :
  + disponible en lecture/écriture ;
  + si vide, retourne le chemin de l’application.
* *ConfigFilePath* :
  + disponible uniquement en lecture ;
  + retourne le chemin complet vers le fichier de config :
    - *[chemin de l’application]/petitspains.config*.
* *DefaultPeopleFileName* :
  + disponible uniquement en lecture ;
  + retourne le nom du fichier contenant les personnes par défaut :
    - *ListePersonnesDefaut.json*.
* *CroissantLinesFileName* :
  + disponible uniquement en lecture ;
  + retourne le nom du fichier contenant la sauvegarde du rapport :
    - *ListeLignes.json*.

### Méthodes ReadConfig/WriteConfig

Méthodes *public*.

Ces méthodes permettent de lire et d’écrire le fichier de configuration.

Le fichier de configuration est un fichier XML : il faut donc que ces méthodes écrivent/lisent les paramètres concernés au format XML.

### Méthodes WriteCroissantsLines/ReadCroissantsLines

Méthodes *public*.

Ces méthodes permettent de lire et d’écrire la sauvegarde pour le rapport des personnes suivies.

Le fichier de sauvegarde du rapport est un fichier JSON : il faut donc que les méthodes sérialisent/désérialisent celui-ci.

Dans le cas où le fichier n’existe pas lors de l’appel de *ReadCroissantsLines*, les personnes par défaut sont chargées via la méthode **Méthode ReadPersons**.

### Méthode ReadPersons

Méthode *private*.

Cette méthode permet de lire le fichier des personnes par défaut au format JSON ; il faut donc désérialiser celui-ci.

Si le fichier n’existe pas, une liste vide de personne est retournée.

### Méthode GetDefaultPeople

Méthode permettant de récupérer la liste des lignes contenant les personnes par défaut à charger, lorsqu’un fichier de sauvegarde n’est pas présent par exemple.

Concrètement, cette méthode construit une liste de lignes contenant les personnes récupérées par **Méthode ReadPersons**.

# Dossier Resources

Dossier contenant les fichiers annexes nécessaires au fonctionnement de l’application.

## Fichier EmailTemplate.tt

Fichier permettant la génération d’un texte au format HTML.

Le fichier utilise les champs passés en paramètre du constructeur de **Fichier EmailTemplateExtended.cs** pour générer un email conforme à ce qui est décrit dans la SF.

### Configuration

* le HTML généré doit être du HTML5 ;
* le HTML doit être au format UTF-8 ;
* l’accès aux images dans un email se fait par l’attribut *cid:*. Exemple :

<img src="cid:croissantEmpty" />

## Fichier EmailTemplate.cs

Fichier autogénéré lors de l’enregistrement de **Fichier EmailTemplate.tt**.

Ce fichier ne doit donc pas être modifié manuellement.

## Fichier EmailTemplateExtended.cs

Fichier d’extension pour la classe autogénérée **Fichier EmailTemplate.cs**. Permet d’étendre cette classe *partial*.

### Champs

* *date*, de type *DateTime* :
  + *private* ;
  + date à laquelle le rapport est traité.
* lines, de type *ItemsChangeObservableCollection<Line>* :
  + *private* ;
  + liste des lignes à afficher dans le rapport.

### Constructeurs

Un seul constructeur à ajouter : le constructeur prend 2 paramètres :

* *date*, de type *DateTime* :
  + *private* ;
  + date à laquelle le rapport est traité.
* lines, de type *ItemsChangeObservableCollection<Line>* :
  + *private* ;
  + liste des lignes à afficher dans le rapport.

Le constructeur s’occupe juste de valoriser ses champs avec ce qui lui est transmis en paramètre. Ainsi, les paramètres deviennent accessibles dans **Fichier EmailTemplate.tt**.

# Dossier Utils

Dossier contenant les classes utilitaires.

## Fichier Enums.cs

Fichier contenant les types *enum* nécessaires à l’application.

### SortDirection

Enum décrivant le sens de tri pour une collection :

* *Ascending* : le tri se fait de manière croissante ;
* *Descending* : le tri se fait de manière décroissante.

## Fichier Extensions.cs

Fichier contenant les extensions nécessaires à l’application.

### Méthode BubbleSort<T>(this ObservableCollection<T> collection) where T : IComparable<T>

Méthode d’extension pour un type *ObservableCollection<T>*.

Appelle **Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T>** avec le paramètre *SortDirection.Ascending*.

### Méthode BubbleSort<T>(this ObservableCollection<T> collection, SortDirection? sortDirection) where T : IComparable<T>

Méthode d’extension pour un type *ObservableCollection<T>*.

Tri une collection de type *ObservableCollection<T>* selon l’ordre demandé par *sortDirection*. Si *sortDirection* est null, le tri se fait de manière croissante.

## Fichier ItemsChangeObservableCollection.cs

Fichier permettant de notifier une collection lorsqu’un élément de celle-ci change. En effet, par défaut, lorsque ceci arrive, la collection n’est pas notifiée et l’interface graphique, par exemple, n’est pas mise à jour.

Son implémentation est reprise de ce projet :

* <https://www.codeproject.com/tips/694370/how-to-listen-to-property-chang>.

# Dossier Model

Dossier contenant les modèles.

## Fichier Person.cs

Fichier représentant une personne.

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *IComparable<Person>* ;
* *INotifyPropertyChanged*.

### Propriétés

* *FirstName*, de type *string* :
  + prénom d’une personne.
* *LastName*, de type *string* :
  + nom d’une personne.
* *PersoId*, de type *string* :
  + persoid d’une personne.
* *Email*, de type *string* :
  + email d’une personne.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque une personne change.

### Constructeur par défaut

Le constructeur par défaut est obligatoire pour assurer une sérialisation correcte. Ce constructeur est vide.

### Constructeur Person(string firstName, string lastName)

Valorisation des propriétés idoines.

### Constructeur Person(string firstName, string lastName, string email)

Valorisation des propriétés idoines.

### Constructeur Person(string firstName, string lastName, string persoId, string email)

Valorisation des propriétés idoines.

### Constructeur Person(SerializationInfo info, StreamingContext context)

Constructeur *protected*.

Désérialisation des propriétés ; cf. <https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.110).aspx> pour un exemple.

### Méthode ToString

Permet de générer une représentation d’une personne sous forme de string :

* *FirstName* + " " + *LastName*.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de sérialiser un objet de type *Person*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de sérialiser un objet de type *Person*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode CompareTo(Person other)

Implémentation de l’interface *IComparable<Person>*.

Permet de comparer une personne par rapport à une autre.

La comparaison doit se faire sur la propriété *FirstName*.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’une personne change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

## Fichier Croissant.cs

Fichier représentant un croissant (un emplacement sur une ligne).

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *IComparable<Person>* ;
* *INotifyPropertyChanged*.

### Enum CroissantState

Enum décrivant l’état d’un croissant :

* *IsAvailable* : le croissant n’a pas reçu de pénalité et peu en recevoir une ;
* *IsUsed* : le croissant possède une pénalité ;
* *IsDeactivated* : le croissant est désactivé : il n’est pas possible de positionner une pénalité dessus.

### Propriétés

* *State*, de type *CroissantState* :
  + état du croissant.
* *Date*, de type *DateTime?* :
  + date à laquelle une pénalité a été positionnée sur le croissant ;
  + lors de sa valorisation, si celle-ci est différente de *null*, la propriété *State* doit être positionnée à *State.IsUsed*. Sinon (si la valeur est *null*), si le croissant n’est pas désactivé, State doit être positionné à *State.IsAvailable*.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque un croissant change.

### Constructeur par défaut

Le constructeur par défaut est obligatoire pour assurer une sérialisation correcte. Ce constructeur se contente de positionner la propriété *State* à *CroissantState.IsAvailable*.

### Constructeur Croissant(DateTime? date)

Constructeur permettant de valoriser la date du croissant.

### Constructeur Croissant(SerializationInfo info, StreamingContext context)

Constructeur *protected*.

Désérialisation des propriétés ; cf. <https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.110).aspx> pour un exemple.

### Méthode ToString

Permet de générer une représentation d’une personne sous forme de string :

* Si le croissant a une date de positionnée :
  + « Croissant : date :  » + [date au format dd/MM/yyyy] + *State*.
* sinon :
  + « Croissant : pas de pénalité ;  » + *State*.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de sérialiser un objet de type *Croissant*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de sérialiser un objet de type *Croissant*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode CompareTo(Croissant other)

Implémentation de l’interface *IComparable<Croissant>*.

Permet de comparer une croissant par rapport à un autre :

* si la date du croissant est *null* et que la date de l’autre croissant ne l’est pas, retourne 1 (un croissant ne possédant pas de date vient après un croissant avec) ;
* sinon si la date du croissant est non *null* et que la date de l’autre croissant l’est, retourne -1 (un croissant possédant une date vient avant un croissant sans) ;
* sinon si les deux dates sont *null*, retourne 0 (les croissants sont égaux) ;
* sinon, on compare les 2 dates.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’un croissant change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

## Fichier Line

Fichier représentant une ligne. Une ligne possède une personne et les emplacements correspondants à cette personne.

### Interfaces

Cette classe implémentent les interfaces suivantes :

* *ISerializable* ;
* *INotifyPropertyChanged*.

### Propriétés

* *Person*, de type *Person* :
  + personne sur la ligne ;
  + lors du *set* de cette propriété, **Méthode HandlePersonChanged(object sender, EventArgs e)** doit souscrire à l’évènement *PropertyChanged* de *Person*.
* *Croissants*, de type *ItemsChangeObservableCollection<Croissant>* :
  + liste des croissants (emplacement) sur la ligne.
* *SelectedCroissant*, de type *Croissant* :
  + croissant sélectionné sur la ligne.
* *CroissantsSlots*, de type *int* :
  + en lecture seule ;
  + nombre de pénalités admissible pour la ligne.
* *PenaltiesAdded*, de type *ObservableCollection<DateTime*> :
  + liste des pénalités qu’une personne a eu pendant l’exécution courante de l’application.
* *HasToBringCroissants*, de type *bool* :
  + booléen indiquant si la personne doit ramener les croissants : cela signifie que la personne a dépassé le nombre de pénalités autorisé pendant l’exécution courante de l’application.
* *CroissantsSlotsDefault*, de type *int* :
  + propriété *static* ;
  + en lecture seule ;
  + retourne systématiquement 10 ;
  + nombre de pénalités par défaut pour une ligne.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé.

### Évènements et delegates

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque une ligne change.
* *delegate void PenaltyAlreadyExistsAtThisDateHandler(object sender, EventArgs e)* :
  + *handler* pour l’évènement *PenaltyAlreadyExistsAtThisDate*.
* *event PenaltyAlreadyExistsAtThisDateHandler PenaltyAlreadyExistsAtThisDate* :
  + évènement à déclencher lorsque le nombre de pénalités autorisées pour une personne est dépassée.

### Constructeur par défaut

Le constructeur par défaut est obligatoire pour assurer une sérialisation correcte. Ce constructeur ne réalise aucune opération : il se contente d’appeler **Constructeur Line(int croissantsSlots)** avec en paramètre *CroissantsSlotsDefault*.

### Constructeur Line(int croissantsSlots)

Constructeur permettant de construire une ligne en indiquant le nombre maximum de pénalités possibles :

* valorisation de *CroissantsSlots* avec le paramètre fourni ;
* instanciation de *Croissants* :
  + ajout d’autant de croissant qu’il y a de défini dans *CroissantsSlots*.
* instanciation de *PenaltiesAdded*.

### Constructeur Line(SerializationInfo info, StreamingContext context)

Constructeur *protected*.

Désérialisation des propriétés ; cf. <https://msdn.microsoft.com/en-us/library/system.runtime.serialization.iserializable.getobjectdata(v=vs.110).aspx> pour un exemple.

Seules les propriétés *Person* et *Croissants* nécessitent d’être désérialisées.

La propriété *PenaltiesAdded* doit être instanciée.

### Méthode GetPenaltiesCount

Méthode permettant de déterminer le nombre de pénalités d’une personne (sur la ligne).

* si *Croissants* différent de *null*, retourne le nombre de croissants dont *State* est à *CroissantState.IsUsed* ;
* sinon, retourne 0.

### Méthode GetPenaltiesLimit

Méthode permettant de déterminer le nombre de pénalités qu’une personne peut avoir avant de devoir ramener les croissants et d’avoir un emplacement de désactiver.

* si *Croissants* différent de *null*, retourne le nombre de croissants dont *State* est différent de *CroissantState.IsDeactivated*;
* sinon, retourne 0.

### Méthode GetNextIndexToDeactivate

Méthode permettant de déterminer quel est l’index de *Croissants* qui sera le prochain à être désactivé.

Les croissants sont désactivés dans l’ordre inverse de la liste (dernier croissant, avant-dernier croissant…).

### Méthode GetLastDeactivatedIndex

Méthode permettant de déterminer quel est l’index de *Croissants* qui a été désactivé en dernier.

Retourne simplement le résultat de **Méthode GetNextIndexToDeactivate** + 1.

### Méthode AddPenalty(DateTime date)

Ajoute une ou plusieurs pénalités à la date passée en paramètre.

* si aucun croissant n’a déjà de pénalité à cette date :
  + détermination du nombre de pénalités à ajouter via **Méthode PenaltiesToAdd(DateTime date)** ;
  + pour chaque pénalité à ajouter :
    - ajout via **Méthode ProcessPenaltyAdding(DateTime date)**.
  + ajout d’une pénalité à la date passée en paramètre dans la collection *PenaltiesAdded*.
* sinon, déclenchement de l’évènement *PenaltyAlreadyExistsAtThisDate* si des abonnés y souscrivent.

### Méthode RemovePenalty(DateTime date)

Supprime les pénalités à la date passée en paramètre.

* sélection des croissants possédant une pénalité à la date passée en paramètre ;
* positionnement à *null* de *Date* pour ces croissants ;
* tri des croissant via la méthode d’extension *BubbleSort*, en passant en paramètre *SortDirection.Ascending* ;
* sélection des pénalités dans *PenaltiesAdded* à la date passée en paramètre ;
* suppression de ces pénalité de *PenaltiesAdded*.

### Méthode ProcessPenaltyAdding(DateTime date)

Méthode permettant l’ajout effectif de pénalités à une ligne.

* si résultat de **Méthode GetPenaltiesCount** est inférieur à résultat de **Méthode GetPenaltiesLimit** :
  + détermination de l’index à partir duquel insérer la pénalité : les pénalités des croissants sont triées par date croissante ;
  + décalage à droite des dates supérieures à la date à insérer ;
  + insertion des nouvelles pénalités.
* sinon -> cela signifie que la personne a dépassé le seuil admissible ; elle doit donc ramener les croissants :
  + remise à *null* de la date pour tous les croissants de la ligne ;
  + désactive le prochain croissant désactivable (déterminé grâce à **Méthode GetNextIndexToDeactivate**) ;
  + positionnement de *HasToBringCroissants* à true.

### Méthode PenaltiesToAdd(DateTime date)

Méthode permettant de déterminer combien de pénalités à ajouter à la personne.

* par défaut, 1 pénalité est ajoutée ;
* si l’ajout se fait un lundi ou un vendredi, 2 pénalités sont ajoutées ;
* si la personne n’a pas soumis sont CRA de la semaine et que l’ajout se fait un vendredi, 1 pénalité est ajoutée. Le contrôle se fait via **Méthode CheckFoutageDeGueule(DateTime date)**.

### Méthode CheckFoutageDeGueule(DateTime date)

Méthode permettant de déterminer si la personne n’a pas saisie son CRA de la semaine et que l’ajout d’une pénalité se fait un vendredi.

La méthode renvoie true si c’est le cas, false sinon.

* création d’une variable *result*, initialisée à true ;
* détermination du premier jour de la semaine pour laquelle *date* appartient, via **Méthode FirstDayOfProcessedWeek(DateTime date)** ; placement dans une variable *firstDayOfProcessedWeek* ;
* création d’une variable *dayToCheck*, initialisée avec *firstDayOfProcessedWeek* ;
* boucle de 0 à 4 (exclus) : représente les jours de lundi à jeudi :
  + si, pour *dayToCheck*, au moins 1 croissant ne possède pas de pénalité :
    - positionnement de *result* à false ;
    - échappement de la boucle.
  + ajout de 1 jour à *dayToCheck*.
* la fonction retourne *result*.

### Méthode FirstDayOfProcessedWeek(DateTime date)

Méthode permettant de déterminer le premier jour de la semaine (lundi) en fonction de la date passée en paramètre.

* création d’une variable *offset* initialisée à 0 ;
* si *date* est un dimanche, offset vaut -6 ;
* si *date* est un lundi, offset vaut 0 ;
* si *date* est un mardi, offset vaut -1 ;
* etc.

La fonction retourne *date.AddDays(offset).Date* (juste la partie *Date*, sans les heures, secondes, etc.).

### Méthode ReactivatedCroissant

Méthode permettant de réactiver un croissant. Concrètement, on réactive systématiquement le croissant qui a été désactivé en dernier. Rappel : les croissants sont désactivés dans l’ordre inverse de la liste *Croissants*.

* récupération de l’index du croissant à réactiver, s’il en existe un, via **Méthode GetIndexOfCroissantToReactivate** ;
* si l’index est différent de *null* :
  + passage de la propriété *State* du croissant concerné à *CroissantState.IsAvailable* ;
  + passage de la propriété *HasToBringCroissants* à false.

### Méthode GetIndexOfCroissantToReactivate

Méthode permettant de déterminer l’index du croissant à réactiver.

* création d’une variable *index*, initialisée à *null* ;
* boucle de 0 à *Croissants.Count* (exclus) dans la collection *Croissants* :
  + si *Croissant[i]* est désactivé :
    - *index* = *i* ;
    - échappement de la boucle.
* la fonction retourne *index*.

### Méthode GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de sérialiser un objet de type *Line*.

Cette méthode doit être *protected*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

Seules les propriétés *Person* et *Croissants* nécessitent d’être sérialisées.

### Méthode ISerializable.GetObjectData(SerializationInfo info, StreamingContext context)

Méthode permettant de désérialiser un objet de type *Line*.

Cette méthode est l’implémentation explicite de l’interface *ISerializable*.

Cf. <https://docs.microsoft.com/en-us/visualstudio/code-quality/ca2240-implement-iserializable-correctly> pour l’implémenter correctement.

### Méthode HandlePersonChanged(object sender, EventArgs e)

Méthode permettant de gérer l’évènement *PropertyChanged* pour une personne.

Cette méthode est privée.

Cette méthode réalise une seule chose :

* si *PropertyChanged* n’est pas *null*, l’évènement est déclenché avec pour paramètre la ligne (*this*) et « Person » dans l’objet *PropertyChangedEventArgs*.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’un croissant change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>(ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon, appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**.

# Dossier ViewModel

Dossier contenant les ViewModel et classes associées.

## Fichier IClosable

Interface implémentant une méthode permettant à un objet de se fermer.

### Méthode Close

Méthode indiquant à l’objet de se fermer.

## Fichier IDialog

Interface implémentant une propriété indiquant quel a été le résultat d’une ouverture d’un élément graphique.

### Interface

Cette interface implémente l’interface suivante :

* *IClosable*.

### Propriété

* *DialogResult*, de type *bool?*.

## Fichier ViewModelBase

Fichier servant de classe de base pour tous les ViewModel de l’application ; essentiellement, cette classe gère l’évènement *PropertyChanged*.

### Évènements et delegates

* *event* *PropertyChangedEventHandler* *PropertyChanged* :
  + fait partie de l’implémentation de *INotifyPropertyChanged* ;
  + évènement à déclencher lorsque une ligne change.

### Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)

Méthode à invoquer lorsqu’un croissant change.

Cette méthode doit déclencher l’évènement *PropertyChanged* si des abonnés ont souscrit à celui-ci (c’est-à-dire que l’évènement n’est pas *null*).

### Méthode SetProperty<T>( Action checkCommands, ref T storage, T value, [CallerMemberName]string propertyName = null)

Méthode permettant de mettre à jour une propriété et d’exécuter une action (*checkCommands*) permettant de rafraichir les *Commands* d’un ViewModel :

* si la valeur de la propriété n’a pas changée, rien ne doit être fait ;
* sinon :
  + appel de **Méthode RaisedPropertyChanged([CallerMemberName]string propertyName = null)**;
  + exécution de *checkCommands*.

## Fichier AddPersonViewModel

Fichier permettant d’insérer une personne à la liste.

### Classes héritées

Cette classe hérite de la classe suivante :

* *ViewModelBase*.

### Propriétés

* *Lines*, de type *ItemsChangeObservableCollection<Line>* :
  + liste de lignes à laquelle ajouter la nouvelle.
* *FirstName*, de type *string* :
  + prénom de la nouvelle personne à insérer.
* *LastName*, de type *string* :
  + nom de la nouvelle personne à insérer.
* *PersoId*, de type *string* :
  + persoid de la nouvelle personne à insérer.
* *Email*, de type *string* :
  + email de la nouvelle personne à ajouter.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>( Action checkCommands, ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé. Le paramètre *checkCommands* est valorisé avec **Méthode CheckCommands**.

### Commands

* *AddPersonCommand*, de type *CommandHandler<IDialog>* :
  + *command* qui permet de réaliser l’ajout de la ligne à la liste.
* *CancelCommand*, de type *CommandHandler<IDialog>* :
  + *command* qui annule la saisie et qui laisse la liste intacte.

### Constructeurs AddPersonViewModel(ItemsChangeObservableCollection<Line> lines)

Seul constructeur de la classe.

* initialisation de *AddPersonCommand* avec en paramètres **Méthode AddPerson(IDialog window)** et **Méthode AddPersonCanExecute** ;
* initialisation de *CancelCommand* avec en paramètre **Méthode Cancel(IDialog window)** ; le second paramètre est un *delegate* qui renvoi tout le temps true, car cette commande doit toujours être exécutable ;
* valorisation de la propriété *Lines* avec le paramètre *lines*.

### Méthode AddPerson(IDialog window)

Méthode ajoutant une nouvelle ligne à la liste et fermant l’élément d’ajout de personne.

* recherche d’une personne comportant le même nom et le même prénom que la personne à ajouter dans *List* ;
* si aucun résultat, ajout d’une nouvelle ligne avec la personne à ajouter ;
* fermeture de l’élément.

### Méthode AddPersonCanExecute

Méthode déterminant si la command *AddPersonCommand* peut s’exécuter.

* retourne true si *FirstName* et *LastName* sont valorisés ;
* sinon, retourne false.

### Méthode Cancel(IDialog window)

Méthode fermant l’élément permettant d’ajouter une personne, sans toucher à la liste.

### Méthode CheckCommands

Méthode permettant de rafraichir les *commands*. Appel de *RaiseCanExecuteChanged* pour :

* *AddPersonCommand* ;
* *CancelCommand*.

## Fichier UpdatePersonViewModel

Fichier permettant de modifier une personne dans la liste.

### Classes héritées

Cette classe hérite de la classe suivante :

* *ViewModelBase*.

### Propriétés

* *PersonToUpdate*, de type *Person* :
  + personne à mettre à jour ;
  + au *set* de cette propriété, **Méthode HandlePersonPropertyChanged(object sender, EventArgs e)** souscrit à l’évènement *PropertyChanged* de *PersonToUpdate*.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>( Action checkCommands, ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé. Le paramètre *checkCommands* est valorisé avec **Méthode CheckCommands**.

### Champs

* *oldFirstName*, de type *string* :
  + prénom avant modification de la personne à modifier.
* *oldLastName*, de type *string* :
  + nom avant modification de la personne à modifier.
* *oldPersoId*, de type *string* :
  + persoid avant modification de la personne à modifier.
* *oldEmail*, de type *string* :
  + email avant modification de la personne à modifier.

### Commands

* *ValidateUpdatePersonCommand*, de type *CommandHandler<IDialog>* :
  + *command* qui permet de valider la modification de la personne.
* *CancelCommand*, de type *CommandHandler<IDialog>* :
  + *command* qui annule la saisie et qui laisse la personne intacte.

### Constructeur UpdatePersonViewModel(Line line)

Seul constructeur de la classe.

* initialisation de *ValidateUpdatePersonCommand* avec en paramètres **Méthode**  et **Méthode CanValidateUpdatePerson** ;
* initialisation de CancelCommand avec en paramètre **Méthode Cancel(IDialog window)** ; le second paramètre est un *delegate* qui renvoi tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *oldFirstName*, *oldLastName*, *oldPersoId* et *oldEmail* avec les propriétés idoines de *line.Person*;
* initialisation de *PersonToUpdate* avec *line.Person*.

### Méthode ValidateUpdatePerson(IDialog window)

Valide la modification d’une personne.

Comme les modifications sont faite instantanément, on se contente de fermer la fenêtre et on indique à l’appelant que le retour est OK.

* positionnement de *window.DialogResult* à true ;
* fermeture de l’élément appelant via la méthode *Close* de *window*.

### Méthode CanValidateUpdatePerson

Permet de déterminer si la commande *ValidateUpdatePersonCommand* peut être exécutée.

Retourne simplement le résultat de **Méthode PersonChanged**.

### Méthode Cancel(IDialog window)

Annule la modification et ferme l’élément appelant.

* si la personne a été modifiée (résultat de **Méthode PersonChanged**) :
  + valorisation des champs de *PersonToUpdate* avec les champs *old\**.
* positionnement de *window.DialogResult* à false ;
* fermeture de l’élément appelant via la méthode *Close* de *window*.

### Méthode PersonChanged

Méthode retournant un booléen indiquant si la personne a réellement été modifiée : oui si c’est le cas, non sinon.

La personne n’a réellement été modifiée si et seulement si un des champs *old\** est différent de son champ correspondant dans *PersonToUpdate*.

### Méthode HandlePersonPropertyChanged(object sender, EventArgs e)

Méthode qui permet de gérer l’évènement *PropertyChanged* levé par une personne.

Lorsque la personne change, les changements sont instantanément visibles depuis l’écran de visualisation de la liste globale.

* appel de **Méthode CheckCommands**.

### Méthode CheckCommands

Méthode permettant de rafraichir les *commands*. Appel de *RaiseCanExecuteChanged* pour :

* *ValidateUpdatePersonCommand*;
* *CancelCommand*.

## Fichier MainWindowViewModel

Fichier gérant l’interface principale de l’application.

### Classes héritées

Cette classe hérite de la classe suivante :

* *ViewModelBase*.

### Propriétés

* *Lines*, de type *ItemsChangeObservableCollection<Line>* :
  + collection de lignes constituant le rapport ;
  + lors du set de cette propriété :
    - **Méthode HandleLinesCollectionChanged(object sender, EventArgs e)** souscrit à *Lines.CollectionChanged* ;
    - pour chaque *line* dans *Lines* :
      * **Méthode HandlePenaltyAlreadyExistsAtThisDate(object sender, EventArgs e)** souscrit à *line.PenaltyAlreadyExistsAtThisDate* ;
      * **Méthode HandleLinePropertyChanded(object sender, EventArgs e)** souscrit à *line.PropertyChanged*.
* *SelectedLine*, de type *Line* :
  + ligne sélectionnée par l’utilisateur.
* *RootPathOriginal*, de type *string* :
  + propriété *get* uniquement sur un champ en *readonly* ;
  + représente le chemin racine vers les fichiers de sauvegarde au chargement de l’application.
* *RootPath*, de type *string* :
  + représente le chemin racine vers les fichiers de sauvegarde.
* *InformationMessage*, de type *string* :
  + message d’information à afficher à l’utilisateur ;
  + si *value* est valorisé, *ErrorPathInvalid* est vidé.
* *ErrorPathInvalid*, de type *string* :
  + message d’erreur à afficher à l’utilisateur lorsque le chemin choisi est invalide ;
  + si *value* est valorisé, *InformationMessage* est vidé.
* *ProcessedDate*, de type *DateTime* :
  + date à laquelle le rapport est traité.
* *IsSendingEmail*, de type *bool* :
  + propriété permet de savoir si l’email de la situation est en cours d’envoi.

Lors de la mise à jour de chaque propriété, un appel à **Méthode SetProperty<T>( Action checkCommands, ref T storage, T value, [CallerMemberName]string propertyName = null)** doit être réalisé. Le paramètre *checkCommands* est valorisé avec **Méthode CheckCommands**.

### Commands

* *SelectRootPathFolderCommand*, de type *CommandHandler* :
  + *command* permettant de sélectionner le dossier de sauvegarde.
* *LoadFileCommand*, de type *CommandHandler* :
  + *command* permettant de charger le fichier se trouvant à l’emplacement de sauvegarde.
* *AddPenaltyCommand*, de type *CommandHandler<Line>* :
  + *command* permettant d’ajouter une pénalité à la ligne *SelectedLine*.
* *RemovePenaltyCommand*, de type *CommandHandler* :
  + *command* permettant de retirer une pénalité à la ligne *SelectedLine*.
* *ActivateCroissantCommand*, de type *CommandHandler* :
  + *command* permettant de réactiver un croissant sur la ligne *SelectedLine*.
* *SelectLineCommand*, de type *CommandHandler<Line>* :
  + *command* permettant d’attribuer une ligne à *SelectedLine*.
* *RemoveLineCommand*, de type *CommandHandler<Line>* :
  + *command* permettant de supprimer une ligne de *Lines*.
* *UpdateLineCommand*, de type *CommandHandler<Line>* :
  + *command* permettant de mettre à jour une ligne (plus précisément, une personne sur une ligne).
* *AddLineCommand*, de type *CommandHandler* :
  + *command* permettant d’ajouter une ligne à *Lines*.
* *EmailCommand*, de type *CommandHandler* :
  + *command* permettant la préparation et l’ouverture sous forme d’un email dans Outlook du rapport de la situation.
* *SaveCommand*, de type *CommandHandler* :
  + *command* permettant la sauvegarde des fichiers à l’endroit spécifié par *RootPath*.

### Constructeur par défaut

Unique constructeur du fichier.

* initialisation de *SelectRootPathFolderCommand* avec en paramètre **Méthode SelectRootPathFolder** ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *LoadFileCommand* avec en paramètre **Méthode LoadAfterCheck** ; le second paramètre renvoie true si RootPath est valorisé, false sinon ;
* initialisation de *AddPenaltyCommand*, avec en paramètre **Méthode AddPenalty(Line line)** ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *RemovePenaltyCommand* avec en paramètres **Méthode RemovePenalty** et **Méthode CanRemovePenalty**;
* initialisation de *ActivateCroissantCommand*, avec en paramètres **Méthode ActivateCroissant** et **Méthode CanActivateCroissant** ;
* initialisation de *SelectLineCommand*, avec en paramètre une ligne qui valorise SelectedLine avec elle-même ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *RemoveLineCommand*, avec en paramètre **Méthode RemoveLine(Line line)** ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *UpdateLineCommand*, avec en paramètre **Méthode UpdateLine(Line line)** ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *AddLineCommand*, avec en paramètre Méthode AddLine ; le second paramètre est un *delegate* qui renvoie tout le temps true, car cette commande doit toujours être exécutable ;
* initialisation de *EmailCommand*, avec en paramètre **Méthode EmailSituation** ; le second paramètre est un delegate qui renvoie true si *Lines* possède plus d’une ligne et que *IsSendingEmail* différent de true ; sinon, il renvoie false ;
* initialisation de SaveCommand avec en paramètre **Méthode Save** ; le second paramètre est un delegate qui renvoie true si RootPath est valorisé, et false sinon ;
* initialisation de *IsSendingEmail* à false ;
* initialisation de *ProcessedDate* à la date du jour ;
* chargement de la configuration : lancement de **Méthodes ReadConfig/WriteConfig** ;
* initialisation de *RootPathOriginal* avec *RootPath* déclaré dans *PetitsPainsStore* ;
* initialisation de *RootPath* avec *RootPath* déclaré dans *PetitsPainsStore* ;
* chargement du rapport via **Méthode Load**.

### Méthode HandlePenaltyAlreadyExistsAtThisDate(object sender, EventArgs e)

Méthode qui permet de gérer l’évènement PenaltyAlreadyExistsAtThisDate d’une ligne.

* Affichage d’un message d’information dans *InformationMessage* :
  + « [*Person*] a déjà une pénalité en date du [*ProcessedDate*]. ».

### Méthode HandleLinePropertyChanded(object sender, EventArgs e)

Méthode qui permet de gérer l’évènement *PropertyChanged* d’une ligne.

* Appel de **Méthode CheckCommands**.

### Méthode HandleLinesCollectionChanged(object sender, EventArgs e)

Méthode qui permet de gérer l’évènement *CollectionChanged* sur Lines.

* Appel de **Méthode CheckCommands**.

### Méthode SelectRootPathFolder

Affichage d’une fenêtre de sélection d’un dossier.

* tentative de positionnement à l’emplacement précisé par RootPath (avec gestion chemin absolu ou relatif) ;
* récupération du chemin choisi dans *RootPath* si l’utilisateur a choisi un chemin et à valider.

### Méthode LoadAfterCheck

Vérification de l’existence des fichiers dans l’emplacement précisé par *RootPath*, et chargement de ceux-ci s’ils sont présents.

* effacement du contenu de *ErrorPathInvalid* ;
* vérification de la présence du fichier de sauvegarde à l’emplacement indiqué ;
* mise à jour de l’emplacement via **Méthode UpdatePath** ;
* si une erreur se produit, valorisation de *ErrorPathInvalid* :
  + « Erreur : le chemin est invalide : [détails]. ».
* chargement du fichier via **Méthode Load** si le chemin existe ;
* sinon, valorisation de *ErrorPathInvalid* :
  + « Erreur : le chemin spécifié n’existe pas. ».

### Méthode UpdatePath

Mise à jour du chemin du fichier de sauvegarde dans la configuration et déplacement du fichier.

* mise à jour de *PetitsPainsStore.RootPath* avec *RootPath* ;
* déplacement des fichiers via **Méthode ManageFilesOnPathChanged**.

### Méthode ManageFilesOnPathChanged

Déplacement du fichier de sauvegarde depuis son emplacement original vers un emplacement cible.

* création d’une variable contenant le chemin original du fichier de sauvegarde ;
* création d’une variable contenant le chemin cible du même fichier ;
* si le fichier original existe et que le chemin cible est différent du chemin original, déplacement du fichier.

### Méthode Load

Méthode chargeant le résultat de la lecture du fichier de sauvegarde dans *Lines*.

### Méthode AddPenalty(Line line)

Ajout d’une pénalité sur une ligne.

* effacement du contenu de *InformationMessage* ;
* appel de **Méthode AddPenalty(DateTime date)**, avec en paramètre *ProcessedDate*;
* appel de **Méthode CheckCommands** pour rafraichir ces dernières.

### Méthode RemovePenalty

Suppression d’une pénalité sur la ligne sélectionnée.

* vérification de l’existence de *SelectedLine* et de *SelectedLine.SelectedCroissant*;
* si c’est le cas, appel de **Méthode RemovePenalty(DateTime date)**, avec en paramètre *SelectedLine.SelectedCroissant.Date.Value* ;
* sinon, valorisation de *InformationMessage* :
  + « Impossible de supprimer une pénalité inexistante. ».
* appel de **Méthode CheckCommands**.

### Méthode CanRemovePenalty

Méthode permettant de déterminer si *RemovePenaltyCommand* peut s’exécuter.

* vérification de l’existence de *SelectedLine* et de *SelectedLine.SelectedCroissant*;
* si c’est le cas, la méthode retourne vrai si *State* pour *SelectedCroissant* vaut *CroissantState.IsUsed*, et false sinon ;
* sinon, la méthode retourne false.

### Méthode ActivateCroissant

Méthode permettant de réactiver un croissant sur une ligne.

* vérification de l’existence de *SelectedLine* et de *SelectedLine.SelectedCroissant*;
* si c’est le cas, appel de **Méthode ReactivatedCroissant** pour *SelectedLine* ;
* sinon, valorisation de *InformationMessage* :
  + « Impossible de supprimer une pénalité inexistante. ».
* appel de **Méthode CheckCommands**.

### Méthode CanActivateCroissant

* vérification de l’existence de *SelectedLine* et de *SelectedLine.SelectedCroissant*;
* si c’est le cas, la méthode retourne vrai si *State* pour *SelectedCroissant* vaut *CroissantState.* *IsDeactivated*, et false sinon ;
* sinon, la méthode retourne false.

### Méthode RemoveLine(Line line)

Méthode permettant de supprimer une ligne du rapport.

* affichage d’une fenêtre de confirmation de suppression de ligne ;
* si l’utilisateur valide la suppression, *line* est supprimée de *Lines*.

### Méthode UpdateLine(Line line)

Méthode permettant de mettre à jour une ligne, et plus précisément la personne sur la ligne.

* instanciation de **Window UpdatePersonWindow**, en passant en paramètre *line* ;
* appel de la méthode *ShowDialog* sur cette instance.

### Méthode AddLine

Méthode permettant d’ajouter une ligne.

* instanciation de **Window AddPersonWindow**, en passant en paramètre *Lines* ;
* appel de la méthode *ShowDialog* sur cette instance ; récupération du résultat dans une variable ;
* si l’utilisateur a effectivement ajouté quelqu’un :
  + **Méthode HandlePenaltyAlreadyExistsAtThisDate(object sender, EventArgs e)** souscrit à l’évènement *PenaltyAlreadyExistsAtThisDate* de la dernière entrée de *Lines* ;
  + **Méthode HandleLinePropertyChanded(object sender, EventArgs e)** souscrit à l’évènement *PropertyChanged* de la dernière entrée de *Lines*.

### Méthode EmailSituation

Méthode *async* permettant de préparer un email au format HTML et de l’ouvrir dans un nouveau mail Outlook, prêt à être envoyé.

* effacement du contenu de *InformationMessage* ;
* instanciation d’un *EmailTemplate* (cf. **Fichier EmailTemplate.tt**), en passant en paramètres *ProcessedDate* et *Lines* ;
* récupération du contenant HTML dans une variable en appelant la méthode *TransformText* sur l’instance *d’EmailTemplate* ;
* positionnement de *IsSendingEmail* à true ;
* démarrage d’une tâche :
  + instanciation d’un nouveau mail Outlook ;
  + pour chaque ligne présente, ajouter l’email de la personne à la liste des destinataires de l’email ;
  + création du sujet du mail :
    - « Soumission des CRA : rapport du [date au format dd/MM/yyyy] ».
  + joindre les images nécessaires à la bonne génération de l’email ; ce sont les images qui sont référencée dans le fichier *.tt* ; pour les ajouter en tant que cid, cf. <https://social.msdn.microsoft.com/Forums/vstudio/en-US/6c063b27-7e8a-4963-ad5f-ce7e5ffb2c64/how-to-embed-image-in-html-body-in-c-into-outlook-mail> ;
  + valorisation du *HTMLBody* de l’email avec le contenu HTML généré ;
  + affichage de l’email à envoyer.
* si la tâche lance une exception :
  + valorisation de *InformationMessage* avec « Erreur lors de la préparation du message » ;
* positionnement de *IsSendingEmail* à false ;
* sauvegarde du rapport via **Méthode Save**.

### Méthode CheckChosenFolder

Vérifie que le dossier de sauvegarde existe. S’il n’existe pas, il est créé.

* tentative de création du dossier à l’emplacement *RootPath* ;
* si une erreur se produit, valorisation de *ErrorPathInvalid* :
  + « Erreur : le chemin est invalide : [détails]. ».

### Méthode Save

Sauvegarde le rapport à l’emplacement spécifié par *RootPath*.

* effacement du contenu de *InformationMessage* ;
* vérification de l’emplacement via **Méthode CheckChosenFolder** ;
* si *ErrorPathInvalid* est vide (pas d’erreur) :
  + appel **Méthode UpdatePath** ;
  + sauvegarde configuration via **Méthodes ReadConfig/WriteConfig** de *PetitsPainsStore* ;
  + sauvegarde du rapport via **Méthodes WriteCroissantsLines/ReadCroissantsLines** de *PetitsPainsStore*.

### Méthode CheckCommands

Méthode permettant de rafraichir les *commands*. Appel de *RaiseCanExecuteChanged* pour :

* *SelectRootPathFolderCommand*;
* *LoadFileCommand ;*
* *AddPenaltyCommand* ;
* *RemovePenaltyCommand* ;
* *ActivateCroissantCommand* ;
* *SelectLineCommand* ;
* *EmailCommand* ;
* *SaveCommand*.

# Dossier View

## Window AddPersonWindow

Vue gérant l’ajout d’une nouvelle ligne (via l’ajout d’une nouvelle personne).

### AddPersonWindow.xaml.cs

Fichier de *code behind* pour cette vue.

#### Interfaces

Cette classe implémente les interfaces suivantes :

* IClosable ;
* IDialog.

#### Propriétés

* *ViewModel* de type *AddPersonViewModel* (cf. **Fichier AddPersonViewModel**).

#### Constructeur AddPersonWindow(ItemsChangeObservableCollection<Line> lines)

Unique constructeur de la classe :

* initialisation de *ViewModel* avec en paramètre *lines* ;
* valorisation du *DataContext* de la page avec *ViewModel* ;
* appel de *InitializeComponent*.

### AddPersonWindow.xaml

Fichier de design pour cette vue.

* nommage de la vue : « AddPersonView » ;
* Positionnement sur une grille des éléments du *ViewModel*;
* bouton *Ajouter* :
  + *bind* sur *AddPersonCommand* ;
  + *CommandParameter* : *bind* sur le nom de la vue.
* bouton *Annuler* :
  + *bind* sur *CancelCommand* ;
  + *CommandParameter* : *bind* sur le nom de la vue.

## Window UpdatePersonWindow

Vue gérant la modification d’une personne sur une ligne existante.

### UpdatePersonWindow.xaml.cs

Fichier de *code behind* pour cette vue.

#### Interfaces

Cette classe implémente les interfaces suivantes :

* IClosable ;
* IDialog.

#### Propriétés

* *ViewModel* de type *UpdatePersonViewModel* (cf. **Fichier UpdatePersonViewModel**).

#### Constructeur UpdatePersonWindow(Line lineToUpdate)

Unique constructeur de la classe :

* initialisation de *ViewModel* avec en paramètre *lineToUpdate* ;
* valorisation du *DataContext* de la page avec *ViewModel* ;
* appel de *InitializeComponent*.

### UpdatePersonWindow.xaml

Fichier de design pour cette vue.

* nommage de la vue : « UpdatePersonView » ;
* Positionnement sur une grille des éléments du *ViewModel*;
* bouton *Valider* :
  + *bind* sur *ValidateUpdatePersonCommand* ;
  + *CommandParameter* : *bind* sur le nom de la vue.
* bouton *Annuler* :
  + *bind* sur *CancelCommand* ;
  + *CommandParameter* : *bind* sur le nom de la vue.

## MainWindow

Vue principale.

### MainWindow.xaml.cs

Fichier de *code behind* pour cette vue.

#### Propriétés

* *ViewModel* de type *MainWindowViewModel* (cf. **Fichier MainWindowViewModel**).

#### Constructeur par défaut

* valorisation du *DataContext* de la page avec l’instance de la page (*this*) ;
* réglage de la culture avec la culture de l’ordinateur sur lequel l’application s’exécute ;
* appel de InitializeComponent.

### MainWindow.xaml

Fichier de design pour cette vue.

* nommage de la vue : « PetitsPainsWindow » ;
* positionnement des élements principaux sur une grille.

#### Resources

* *nullToBoolConverter*, de type *NullToBoolConverter* (cf. **Fichier NullToBoolConverter.cs**) ;
* *stringNullOrEmptyToBoolConverter*, de type *StringNullOrEmptyToBoolConverter* (cf. **Fichier StringNullOrEmptyToBoolConverter.cs**).

#### Grille principale

Grille contenant tous les éléments au premier plan.

* définition de 5 lignes ;
* 4ème ligne (gestion des détails d’un croissant sélectionné) : mise en place de *triggers* :
  + *bind* sur *SelectedLine.SelectedCroissant*, à l’aide du *converter* *nullToBoolConverter* ;
  + si *SelectedCroissant* n’est pas *null*, *Height* pour la ligne vaut 45 ;
  + si *SelectedCroissant* est *null*, *Height* vaut 0 ;
  + par défaut, *Height* vaut 0.

#### Grille de gestion du chemin

Grille contenant le chemin de sauvegarde et les boutons associés.

* *TextBox* pour le chemin :
  + *bind* sur la propriété *RootPath* ;
  + *UpdateSourceTrigger* positionné sur *PropertyChanged*.
* *Button* pour le changement de chemin :
  + *bind* sur la *command* *SelectRootPathFolderCommand*.
* *Button* pour le chargement :
  + *bind* sur la *command* *LoadFileCommand*.
* *TextBlock* pour les messages :
  + mise en place de *Trigger* :
    - si propriété *InformationMessage* est valorisée, *bind* de la propriété *Text* du *TextBlock* avec *InformationMessage* (avec l’aide de *stringNullOrEmptyToBoolConverter*) ;
    - si propriété *ErrorPathInvalid* est valorisée, *bind* de la propriété *Text* du *TextBlock* avec *ErrorPathInvalid* (avec l’aide de *stringNullOrEmptyToBoolConverter*).

#### Gestion de la date

*DockPanel* contenant la date de traitement.

* *DatePicker* pour la date :
  + *bind* sur la propriété *ProcessedDate*.

#### Gestion de la liste des lignes

*ListView* contenant les lignes du rapport.

* *ItemsSource* de la *ListView* :
  + *bind* sur Lines.
* *SelectedItem* :
  + *bind* sur SelectedLine.
* mise en place d’un *ContextMenu* :
  + menu « Modifier la personne… » qui pointe sur la *command* *UpdateLineCommand*; *CommandParameter* avec un *bind* sur *SelectedLine*;
  + menu « Supprimer la personne… » qui pointe sur la *command* *RemoveLineCommand* ; *CommandParameter* avec un *bind* sur *SelectedLine.*
* *TextBlock* pour la personne :
  + *bind* sur la propriété *Person*.
* *ListView* pour les croissants :
  + *ItemsSource* :
    - *bind* sur *Croissants*.
  + *SelectedItem* :
    - *bind* sur *SelectedCroissant*.
  + mise en place d’un *trigger* lorsqu’un croissant prend le *focus* :
    - *bind* sur la *command* *SelectLineCommand*, avec en *CommandParameter* la ligne elle-même.
  + mise en place de *triggers* pour le changement d’image des croissants :
    - *bind* sur la propriété *State* :
      * si *CroissantState.IsAvailable* :
        + utilisation de *croissant\_empty.png*.
      * si *CroissantState.IsUsed*:
        + utilisation de *croissant\_filled.png*.
      * si *CroissantState.* *IsDeactivated*:
        + utilisation de *croissant\_greyed.png*.
      * par défaut :
        + utilisation de *croissant\_empty.png*.
  + *ToolTip* pour chaque croissant :
    - *bind* sur la propriété *Date*.
* bouton « + » pour chaque ligne :
  + *bind* sur la *command* *AddPenaltyCommand*, avec en *CommandParameter* la ligne elle-même.

#### Gestion des détails d’un croissant sélectionné

Grille contenant la gestion des informations d’un croissant sélectionné.

* *TextBlock* pour la personne :
  + *bind* sur *SelectedItem.Person*.
* *TextBlock* pour le croissant :
  + *bind* sur SelectedLine.SelectedCroissant.
  + mise en place de *triggers* pour le *Text*, basé sur la propriété *State* :
    - si *CroissantState.IsAvailable* :
      * « Cet emplacement est vide. ».
    - si *CroissantState.IsUsed* :
      * « Pénalité en date du [date au format dd/MM/yyyy]. ».
    - si *CroissantState.IsDeactivated* :
      * « Cet emplacement n’est pas disponible ».
* *Button* pour retirer la pénalité :
  + *bind* sur la *command* *RemovePenaltyCommand*.
* *Button* pour réactiver l’emplacement :
  + *bind* sur la *command* *ActivateCroissantCommand*.

#### Gestion des commandes globales

Partie gérant les actions globales sur le rapport.

* *Button* pour ajouter une personne :
  + *bind* sur la command *AddLineCommand*.
* *Button* pour envoyer le rapport par email :
  + *bind* sur *EmailCommand*.
* *Button* pour sauvegarder :
  + *bind* sur *SaveCommand*.