Projet Programmation Système

Livrable 1

Documentation de la version 1.0

Groupe 6	Maxime JENN Loïc MATTER François PANSERA	24/11/2021

Table des matières

Contexte

Le groupe de projet vient d'intégrer l'équipe d'édition de l'éditeur de logiciels ProSoft. Sous la responsabilité du DSI, nous aurons la responsabilité de gérer la création du projet "EasySave" qui consiste à développer un logiciel de sauvegarde. Ce logiciel sera mis en vente à 200€ unité et sera un gestionnaire de sauvegarde. La version 1.0 (actuelle) doit être une application console codée sous C#/.NET Core 3.0 et doit être multiplateforme.

Présentation de l'équipe

Maxime JENN

François PANSERA

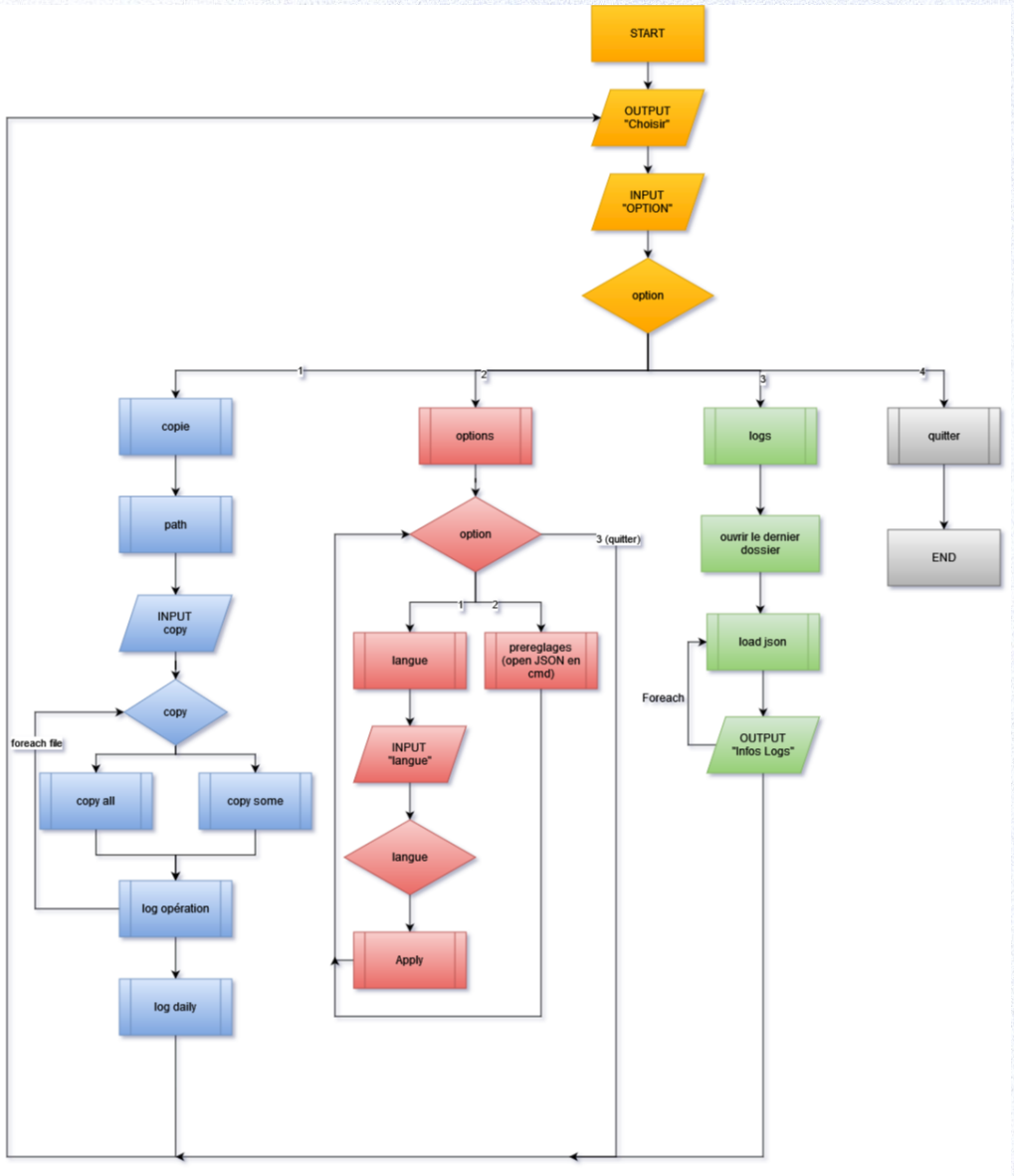
MATTER Loïc

Documentation fonctionnelle

Fonctionnement global de l'application

L'application est un logiciel de sauvegarde appelé « EasySave » permettant de cloner un répertoire de fichiers d'un emplacement vers un autre de façon optimisée. L'application inclut des logs (en temps réel et d'état), un changement de langue via les options, ainsi que la possibilité de sauvegarder 5 pré-configurations pour les réutiliser à de multiples reprises.

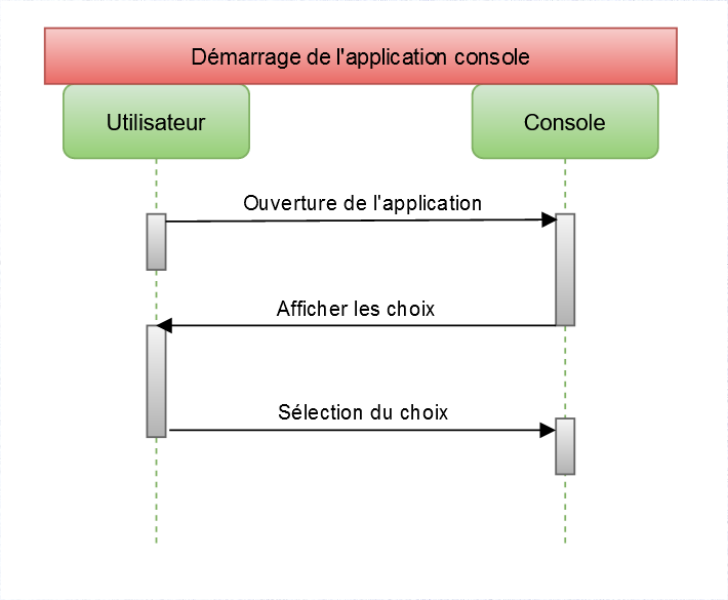
L'application démarre avec un menu, où l'on peut choisir entre « Copier », « Options », « Logs », et « Quitter ». Pour le moment, tout se fait sur la console.



Fonctionnalités détaillées

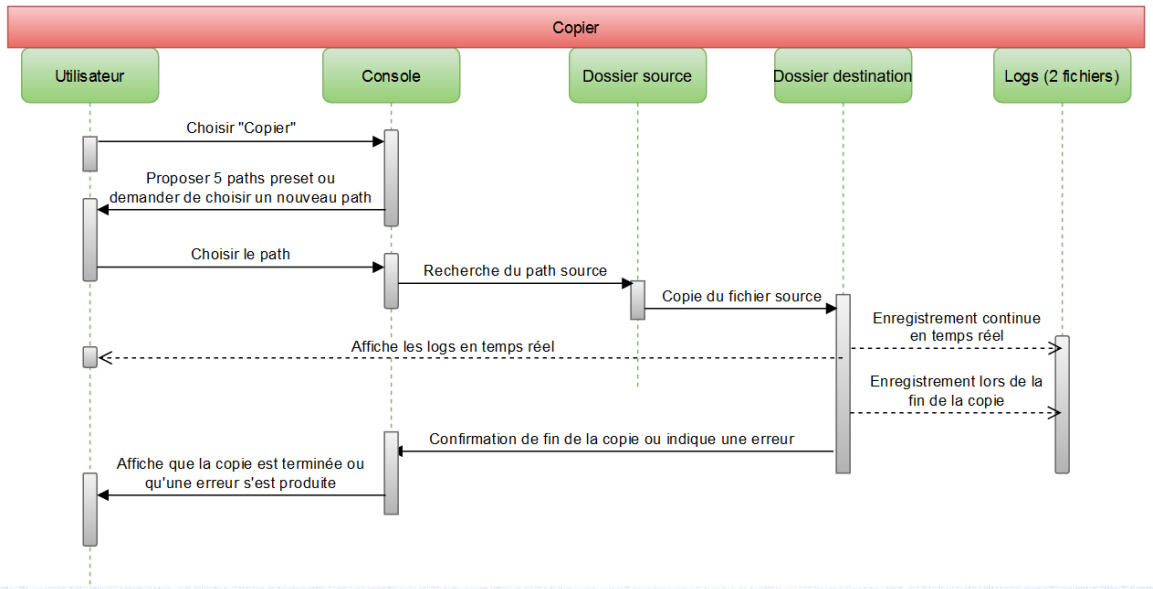
1. Menu principal

Lorsqu'on lance l'application, la console affiche le menu qui comprend « Copier », « Options », « Logs », et « Quitter ». L'utilisateur peut choisir entre ces 4 options.



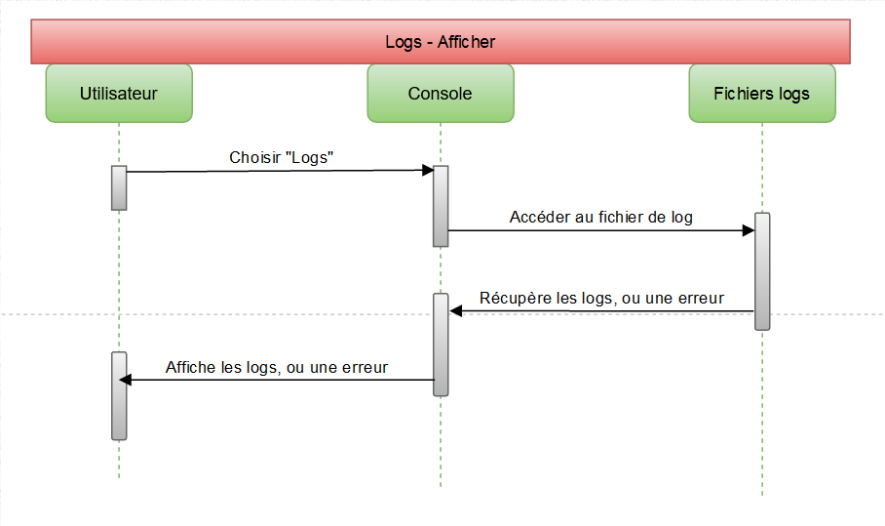
2. Copier un fichier

Lorsque l'utilisateur choisit « Copier » sur le menu, la console affiche les presets enregistrés dans le fichier preset.json et demande à l'utilisateur de choisir parmi l'un d'eux ou de définir un nouveau path, ainsi qu'un type de copie (complet ou différentiel). Le programme copie ensuite les fichiers à partir des chemins prédéfinis défini. Durant la copie, on enregistre les informations dans un fichier log en temps réel pour chaque fichier. Quand la copie est terminée, on enregistre l'état du travail de sauvegarde dans un fichier log d'état, puis on affiche dans la console que la copie a bien été effectuée, ou alors on renvoie une erreur.



3. Afficher les logs

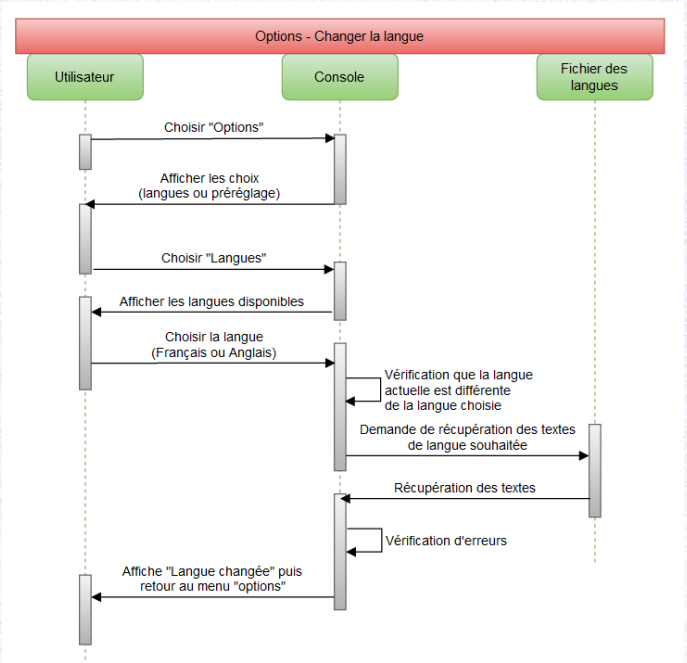
Pour afficher les logs, on choisit « Voir les logs » dans le menu, puis on accède aux fichiers de logs qu'on affiche dans la console de façon structurée et claire.



4. Options

a) Changer la langue

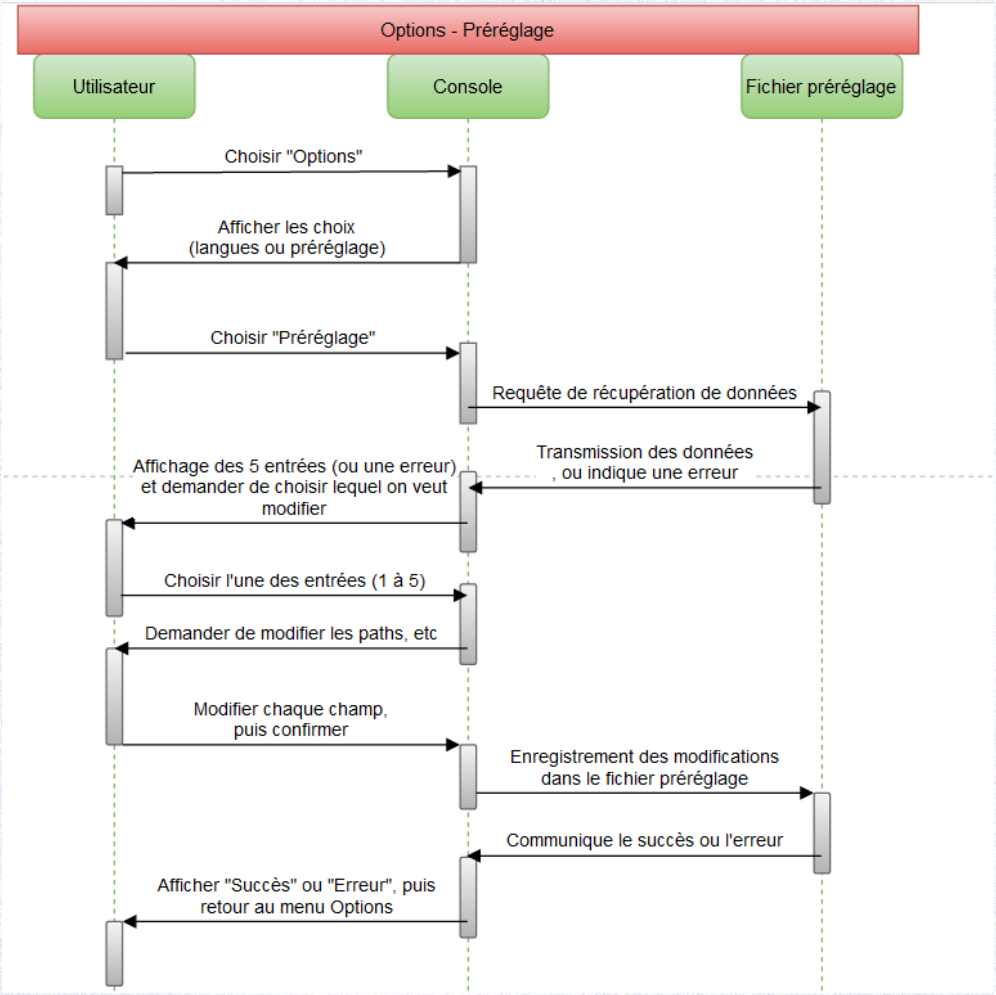
Pour changer la langue, on entre tout d'abord dans le menu « Options », puis l'utilisateur choisit « Langue », et la console affiche les langues disponibles (Français ou Anglais). Le programme récupère ensuite les traductions sur un fichier JSON, puis on revient au menu principal avec les textes traduits.



b) *Préréglage*

L'application permet de sauvegarder 5 configurations de transfert de fichier. Chaque configuration contient un nom, un dossier source, et un dossier destination.

Pour effectuer ces préréglages, on entre dans le menu options, puis on choisit « Preset », puis le programme affiche les préréglages actuels qu'il a récupéré depuis un fichier JSON. On choisit l'une des entrées à modifier, puis on entre les nouveaux path source, destination et nom. Le programme modifie le fichier JSON avec nos paramètres, puis, s'il n'y a pas de problème, la console revient sur le menu « Options ».



Documentation technique

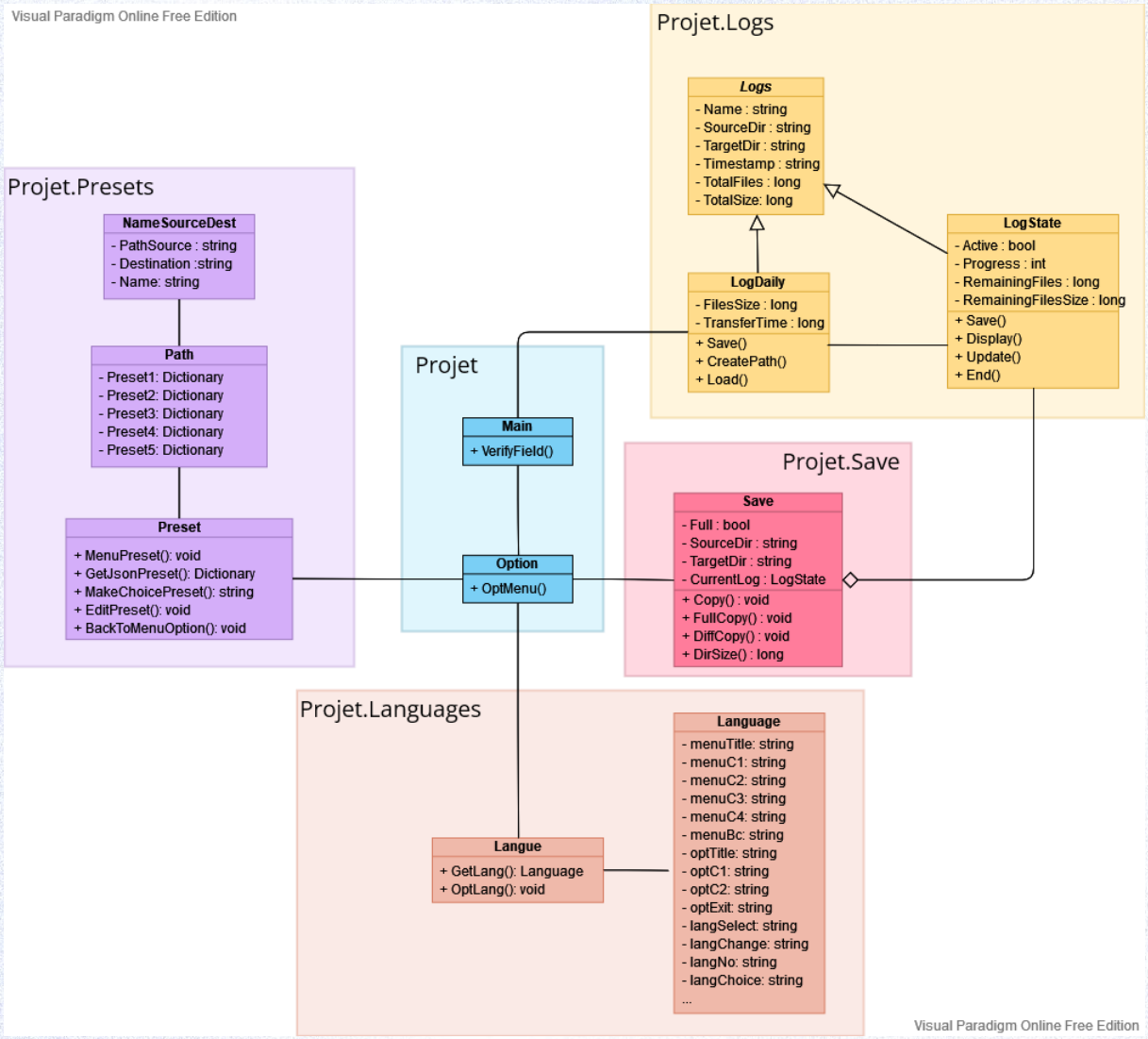
Cette partie comporte le détail technique au niveau du code.

Organisation du code

Pour l'application, on utilise la programmation orienté objet, donc en utilisant des classes.

Chaque classe est compris dans un espace de nom différent (aussi appelé Namespace). Nous avons 5 Namespace : « Presets », « Logs », « Save », « Languages », et Projet. Les 4 premiers sont tous dans le namespace Projet, et leurs noms indiquent directement leurs fonctions.

Les différentes classes, ainsi que leurs attributs et méthodes sont indiqués, et nous verrons tout cela en détail plus bas.



5. Classe Main

La classe Main est la classe qui est lancé lorsqu'on démarre l'application. Elle permet d'afficher le menu de façon propre et de rediriger l'utilisateur vers les autres classes. Elle possède 4 méthodes :

- **Menu()** : Type « void »

Permet d'appeler les autres fonctions de la classe. Pas de paramètre.

- **MakeChoice(dictLang)** : Type « string ».

Permet d'afficher les choix possibles (copier, options, afficher les logs, quitter). Prend en paramètre la langue.

- **Title()** : Type « void ».

Permet d'afficher le titre de l'application formaté de manière qu'il soit beau.

- **GetMainChoice(dictLang)** : Type « string ».

Permet d'acquérir les choix de l'utilisateur pour le menu.

- **RedirFromMain(choice, dictLang)** : Type « void ».

Permet de se rediriger sur les autres classes en fonction du choix de l'utilisateur.

6. Classe Option

La classe Option a pour but de laisser l'utilisateur choisir et de le rediriger sur les classes Langue ou Preset. Elle possède une seule méthode.

- **OptMenu(dictLang)** : Type « void ».

Permet d'afficher le menu des options, et de laisser l'utilisateur choisir entre « Langue », « Preset » et « Retour ».

Namespace Presets

7. Classe Preset

La classe Preset permet à l'utilisateur de modifier les paths préréglés dans un fichier Json. Cette classe s'appuie sur les classes Path et NameSourceDest pour déclarer les variables correspondante au fichier Json. Cette classe comprend 5 méthodes :

- **MenuPreset()** : Type « void ».

Permet d'appeler les différentes méthodes de la classe.

- **GetJsonPreset()** : Type « Dictionary<string, NameSourceDest>».

Permet d'acquérir les informations du fichier JSON appelé preset.json.

- **MakeChoicePreset(preset)**: Type « string ».

Permet à l'utilisateur de choisir quel preset il souhaite modifier.

- **EditPreset(choice, preset)**: Type « void ».

Permet de modifier le preset qui a été choisi, puis d'enregistrer les données dans le fichier Json.

- **BackToMenuOption()**: Type « void ».

Permet de revenir au menu des options lorsque la modification a été réalisé avec succès.

8. Classe Path

La classe Path permet de déclarer les 5 presets qui sont présent dans le fichier preset.json. Chaque preset possède 3 éléments : le nom, le path source, et le path destination. Nous utilisons donc le type Dictionary pour ces variables. Cette classe ne possède pas de méthode, mais 5 attributs : Preset1, Preset2, Preset3, Preset4, et Preset5, tous de type « *Dictionary<string, NameSourceDest>* ».

9. Classe NameSourceDest

La classe NameSourceDest permet de déclarer les éléments de chaque preset qui sont présent plusieurs fois dans le fichier preset.json. L'utilisation du Dictionary permet d'éviter la redondance de ces variables. Cette classe ne possède pas de méthode, mais possède 3 attributs : Name, PathSource, et PathDestination, tous de type « *string* ».

Namespace Languages

10. Classe Langue

La classe Langue permet à l'utilisateur de changer la langue. Pour le moment, seulement le français et l'anglais est pris en charge. La classe permet donc de lire des fichiers Json où se trouvent tous les textes dans toutes les langues. Chaque texte affichés dans la console sera donc une variable. Cette classe possède 2 méthodes :

- **GetLang()** : Type « Language ».

Permet d'instancier la langue, grâce à la lecture du fichier config.json. En effet, une fois le json désérialisé, on récupère la variable lang représentant le paramètre de langage du programme. On instancie alors cet objet grâce au package NuGet Json.NET et le renvoie au sein du programme.

- **GetFiles(dictLang)** : Type « Language ».

Permet de modifier la langue et réactualiser directement le programme dans les options. On dresse la liste pour toutes les langues (l'utilisateur/entreprise pouvant alors ajouter des traductions au sein du programme) présentes dans le dossier « ./data/lang ». L'utilisateur saisit alors l'indicatif de la langue et le programme réactualise la langue directement au sein du programme et retourne aux options.

11. Classe Language

La classe Language sert au multi-langage de notre programme, et les extrait du JSON. La classe, instanciée via GetLang(), va récupérer tous les textes venant du document et évitera de stocker toutes les langues au niveau du programme et ainsi économiser de la RAM.

Namespace Save

12. Classe Save

La classe Save permet de copier un dossier d'une source vers une destination. La classe va tout d'abord demander quel type de copie on veut faire (complète ou différentielle), puis d'enregistrer le tout dans des fichiers logs. La classe Save possède 3 méthodes :

- **Copy()** : Type « void »

Cette méthode prédéfinit les champs constants d'un travail de sauvegarde, tel que le type de sauvegarde ou les chemins sources et cibles.

Elle permet également de vérifier l'existence du dossier source, créer le dossier cible s'il est inexistant.

Enfin, cette méthode crée les DirectoryInfo source et cibles puis appelle ProcessCopy()

- **ProcessCopy(DirectoryInfo source, DirectoryInfo target)** : Type « void »

Cette méthode récursive permet de copier tous les fichiers du dossier source individuellement vers le dossier cible, en reproduisant l'arborescence présente dans le dossier source.

Pendant le procédé, cette méthode va faire appel à un objet LogDaily ainsi qu'un objet LogState afin d'envoyer les informations relatives à la copie en direct (nombre de fichiers restants, temps de traitement, etc…)

Voici la fréquence d'appel des logs :

- Écriture de State à chaque fichier
- Écriture de State à chaque dossier
- Mise à jour des champs de State à chaque fichier
- Rafraîchissement de l'affichage de State à chaque dossier

- **DirSize(DirectoryInfo directory)**: Type « long », statique

Récursive, cette méthode retourne la taille d'un dossier en additionnant à un « long » la taille en octets de chaque fichier trouvé dans ce dernier.

Namespace Logs

13. Classe LogBase

La classe sert de base à aux classes LogState et LogDaily, héritant tous les deux de LogBase.

Cette classe contient tous les champs communs aux deux classes citées précédemment, ainsi que 3 méthodes

- **CreatePath(string path)** : Type « void »

Cette méthode permet de créer l'arborescence de dossiers dans laquelle les logs seront enregistrés.

Cette arborescence est :

Type(Daily ou State)/Année/Mois/Jour

La date est définie par le champ DateTimeStamp, initialisé lors de l'instanciation des classes filles.

- **GetDay()** : Type « string »

Retourne la chaîne de caractères suivante à partir de DateTimeStamp, utilisée pour créer les dossiers de logs.

/ANNEE/MOIS/JOUR

- **GetMinute()** : Type « string »

Retourne la chaîne de caractères suivante à partir de DateTimeStamp, utilisée pour créer les fichiers de logs.

/HEUREhMINUTE.JSON

14. Classe LogState

La classe LogState permet de gérer les logs d'états, ils représentent l'avancement d'un travail de sauvegarde. Cette classe possède 3 méthode :

- **Save()** : Type « void »

Enregistre les propriétés de la classe dans un JSON, à chaque fois que cette méthode est appelée, les données précédentes d'un travail de sauvegarde sont remplacées par les nouvelles.

- **Display()** : Type « void »

Affiche les champs de LogState, les champs ne changeant pas au cours d'un même travail de sauvegarde sont affichés une seule fois, tandis que les champs variants sont affichés sur une même ligne, mise à jour à chaque appel de la méthode.

- **Update(long remainingFilesSize)** : Type « void »

Met à jour les champs de la méthode en fonction des paramètres envoyés par l'objet Save.

Calcule également le nombre de fichiers restants, leur taille globale ainsi que le pourcentage de progression en fonction de la taille restante.

- **End(paramètres)** : Type « XX »

Définit le champ « Active » sur false, sauvegarde une dernière fois le log puis affiche un message de réussite.

15. Classe LogDaily

La classe LogDaily permet de gérer les logs quotidiens, ils contiennent l'ensemble des informations liées aux fichiers traités lors d'un travail de sauvegarde. Cette classe possède 3 méthodes :

- **Update(long fileSize, double transferTime, string sourceDir, string targetDir)** : Type « void »

Cette méthode met à jour les champs de la LogDaily en fonction des informations envoyées par l'objet « Save » puis appelle la méthode Save

- **Save()** : Type « void »

Enregistre les propriétés de la classe dans un JSON, à chaque fois que cette méthode est appelée, les nouvelles données s'ajoutent aux anciennes au sein d'un même travail de sauvegarde.

- **Load()** : Type « void »

Cette méthode permet d'afficher les logs du dossier correspondant au jour actuel en affichant chaque log de chaque fichier présent dans ce dernier.

La transition entre deux fichiers est marquée par un texte rouge, la transition entre deux entrées est marquée par un texte cyan.

Il reste une amélioration possible, en offrant la possibilité à l'utilisateur de choisir le jour à afficher.

Conclusion

L’application permet donc de copier un fichier de différentes manière (complet ou différentiel) tout en enregistrant les informations dans des fichiers de logs, en prenant en charge les changements de langue (Français ou Anglais), ainsi que la sauvegarde de configuration préconfigurés (Path source et path destination déjà enregistré).