v3.0 – Documentation Technique

Rayane HASSANI
Bastien LEDOUX
Léo MARTIN

Février 2026

Table des matières

1. Présentation du produit

EasySave v3.0 est l'évolution majeure du logiciel de sauvegarde ProSoft. Elle introduit l'exécution parallèle des travaux, le contrôle temps réel individuel et global (Play / Pause / Stop), la gestion des fichiers prioritaires, la limitation de bande passante, une pause automatique en cas de détection du logiciel métier, CryptoSoft en mono-instance et la centralisation des logs via Docker.

Informations de version

Attribut	Valeur
Version	3.0
Date de sortie	Mars 2026
Framework	.NET 10.0
Interface	Application WPF (Windows Presentation Foundation)
Prédécesseur	EasySave v2.0 (WPF) / v1.1 (Console)

Nouvelles fonctionnalités v3.0

- Exécution parallèle de tous les travaux de sauvegarde (abandon du mode séquentiel)
- Contrôle individuel par travail : Play, Pause (après le fichier en cours), Stop immédiat
- Contrôle global : Play, Pause, Stop sur l'ensemble des travaux
- Pause automatique si logiciel métier détecté — reprise automatique à l'arrêt
- Gestion des fichiers prioritaires : aucun fichier non prioritaire ne démarre si une extension prioritaire est en attente
- Limitation bande passante : un seul fichier volumineux (> n Ko) transféré à la fois (n paramétrable)
- CryptoSoft mono-instance : Mutex système + SemaphoreSlim(1) pour éviter les conflits
- Centralisation des logs via Docker : local, Docker, ou les deux simultanément
- Ajout identifiant machine/utilisateur dans les logs centralisés
- Suivi de progression temps réel par travail dans l'UI WPF

2. Configuration requise

Configuration minimale

Composant	Exigence
Système d'exploitation	Windows 10 / Windows 11 / Windows Server 2019+
Framework	.NET 10.0 Runtime
RAM	512 Mo minimum (1 Go recommandé pour la parallelisation)
Espace disque	100 Mo pour l'installation
Processeur	Compatible x64, multi-cœur recommandé
Résolution	1280×720 minimum recommandée

Prérequis supplémentaires

- CryptoSoft.exe présent dans le même dossier qu'EasySave.WPF.exe
- Docker Desktop (optionnel) pour la centralisation des logs
- Port 5000 ouvert si le service Docker de logs est activé

3. Installation

Étapes d'installation

1. S'assurer que .NET 10.0 Runtime est installé
2. Extraire le package EasySave à l'emplacement souhaité
3. Vérifier que CryptoSoft.exe est présent dans le même dossier
4. (Optionnel) Installer Docker Desktop pour la centralisation des logs
5. Exécuter EasySave.WPF.exe pour démarrer l'interface graphique

Démarrage du service Docker de logs (optionnel)

```
cd EasySave/LogServer
docker-compose up -d
```

4. Emplacement des fichiers

Fichiers de l'application

Fichier	Emplacement	Description
EasySave.WPF.exe	Dossier d'installation	Exécutable principal (interface graphique)
EasySave.Core.dll	Dossier d'installation	Bibliothèque métier partagée (parallélisme, contrôle)
EasyLog.dll	Dossier d'installation	Bibliothèque de journalisation (JSON/XML + Docker)
CryptoSoft.exe	Dossier d'installation	Outil de chiffrement XOR — mono-instance (Mutex)
docker-compose.yml	LogServer/	Configuration du service de centralisation des logs
Dockerfile	LogServer/	Image Docker du service de logs

Fichiers de données

Tous les fichiers de données restent dans le dossier AppData de l'utilisateur :

```
%APPDATA%\EasySave\
```

Fichier	Emplacement complet	Description
config.json	%APPDATA%\EasySave\config.json	Configuration générale (jobs, chiffrement, logiciel métier, format log, seuil, extensions prioritaires, mode centralisation)
state.json	%APPDATA%\EasySave\state.json	Fichier d'état temps réel (multi-jobs simultanés)
YYYY-MM-DD.json/.xml	%APPDATA%\EasySave\Logs\	Logs journaliers (format JSON ou XML)

5. Architecture

5.1 Patron de conception

La version 3.0 conserve l'architecture MVVM introduite en v1.0 et le projet EasySave.Core partagé de la v2.0. Les évolutions majeures sont l'introduction du parallélisme via Task/CancellationToken/ManualResetEventSlim, un PriorityCoordinator pour synchroniser les règles inter-jobs, et un SemaphoreSlim global pour la limitation de bande passante.

5.2 Structure du projet (évolutions v3.0)

```
.
├── CryptoSoft/              (Outil de chiffrement externe)
│   ├── FileManager.cs
│   └── Program.cs
├── EasyLog/                 (Composant de gestion des logs)
│   ├── Models/              (Entrées, Formats, Modes de log)
│   └── Services/            (LogService et Stratégies JSON/XML)
├── EasySave.Console/        (Interface utilisateur en ligne de commande)
│   ├── Views/               (CmdView.cs)
│   └── Program.cs
├── EasySave.Core/           (Cœur métier de l'application)
│   ├── Models/              (Config, Jobs, Enums, État de progression)
│   ├── Services/            (Backup, Chiffrement, Langues, Monitoring)
│   ├── Strategies/          (Sauvegarde Complète vs Différentielle)
│   ├── Utils/               (FileUtils.cs)
│   └── ViewModels/          (WpfViewModel.cs pour le lien UI)
├── EasySave.LogServer/      (Serveur de logs décentralisé - API)
│   ├── Controllers/         (LogController.cs)
│   ├── Dockerfile
│   └── Program.cs
├── EasySave.Tests/          (Tests unitaires et d'intégration)
│   └── ... (Tests de config, jobs, chiffrement, logs, etc.)
├── EasySave.WPF/            (Interface graphique Windows)
│   ├── Controls/            (Composants UI : Dashboard, JobCard, Progress)
│   ├── Styles/              (Thèmes Mode Nuit et Caramel Profond)
│   └── Views/               (MainView, WelcomeView, App.xaml)
├── CentralizedLogs/         (Répertoire de stockage des fichiers logs)
├── EasySave.slnx            (Solution globale du projet)
├── docker-compose.yml       (Orchestration du serveur de logs)
└── README.md                (Documentation du projet)
```

5.3 Principes architecturaux clés v3.0

Parallélisme

Chaque travail de sauvegarde s'exécute dans une Task dédiée. Le WpfViewModel lance toutes les tâches simultanément via Task.WhenAll(). Chaque ServiceBackupExecution possède son propre CancellationTokenSource (Stop) et ManualResetEventSlim (Pause/Resume).

Contrôle Play / Pause / Stop

Chaque job dispose d'un triplet de primitives de synchronisation dans le WpfViewModel. La pause est effective uniquement après la fin du transfert du fichier en cours (vérification dans la boucle de copie). Le stop est immédiat via CancellationToken.

Gestion des fichiers prioritaires (PriorityCoordinator)

Le PriorityCoordinator est un singleton partagé entre tous les jobs. Il maintient un compteur du nombre de fichiers à extension prioritaire en attente sur l'ensemble des travaux. Avant de copier un fichier non prioritaire, chaque stratégie appelle WaitIfPriorityPendingAsync() qui bloque jusqu'à ce que ce compteur soit à zéro.

Limitation de bande passante (BandwidthLimiter)

Un SemaphoreSlim(1) global est utilisé pour les fichiers dont la taille dépasse le seuil paramétrable (n Ko). Seul un fichier volumineux peut être transféré à la fois. Les fichiers sous le seuil ne sont pas affectés (sous réserve des règles de priorité).

CryptoSoft mono-instance

CryptoSoft est protégé par un Mutex système nommé (Global\CryptoSoft_ProSoft_V3) qui empêche toute deuxième instance de s'exécuter simultanément, même depuis plusieurs processus ou utilisateurs. En v3.0, le SemaphoreSlim d'EncryptionService est réduit à 1 (au lieu de 2 en v2.0) pour être cohérent avec la contrainte mono-instance.

5.4 Responsabilités des composants principaux

Composant	Projet	Responsabilité
WpfViewModel	EasySave.Core	Orchestre le parallélisme : lance N Tasks, expose les commandes Play/Pause/Stop par job et globales
ServiceBackupExecution	EasySave.Core	Exécute un travail de manière async, vérifie pause/stop/métier à chaque fichier
PriorityCoordinator	EasySave.Core	Singleton — synchronise les règles d'extensions prioritaires entre tous les jobs
BandwidthLimiter	EasySave.Core	Singleton — SemaphoreSlim(1) pour les fichiers volumineux
BusinessSoftwareMonitor	EasySave.Core	Singleton — polling 2s, pause auto tous les jobs dès détection, reprise automatique
EncryptionService	EasySave.Core	SemaphoreSlim(1) — garantit qu'une seule instance CryptoSoft tourne à la fois
LogService	EasyLog	Singleton — écriture locale + envoi optionnel au service Docker selon LogCentralizationMode
DockerLogSender	EasyLog	Client HTTP envoyant les entrées de log au service Docker en temps réel
LogAggregatorService	LogServer	Service Docker REST — reçoit les logs de N machines et écrit un fichier journalier unique
FullBackupStrategy	EasySave.Core	MODIFIÉ — appels WaitIfPriorityPendingAsync() et BandwidthLimiter avant chaque copie
DifferentialBackupStrategy	EasySave.Core	MODIFIÉ — idem FullBackupStrategy

6. Fonctionnalités

6.1 Exécution parallèle des travaux

En v3.0 le mode séquentiel est abandonné. Tous les travaux sélectionnés démarrent simultanément dans des tâches asynchrones indépendantes. La progression de chaque job est visible en temps réel dans l'interface.

6.2 Contrôle individuel et global (Play / Pause / Stop)

Chaque carte de travail dans l'onglet Jobs expose trois boutons :

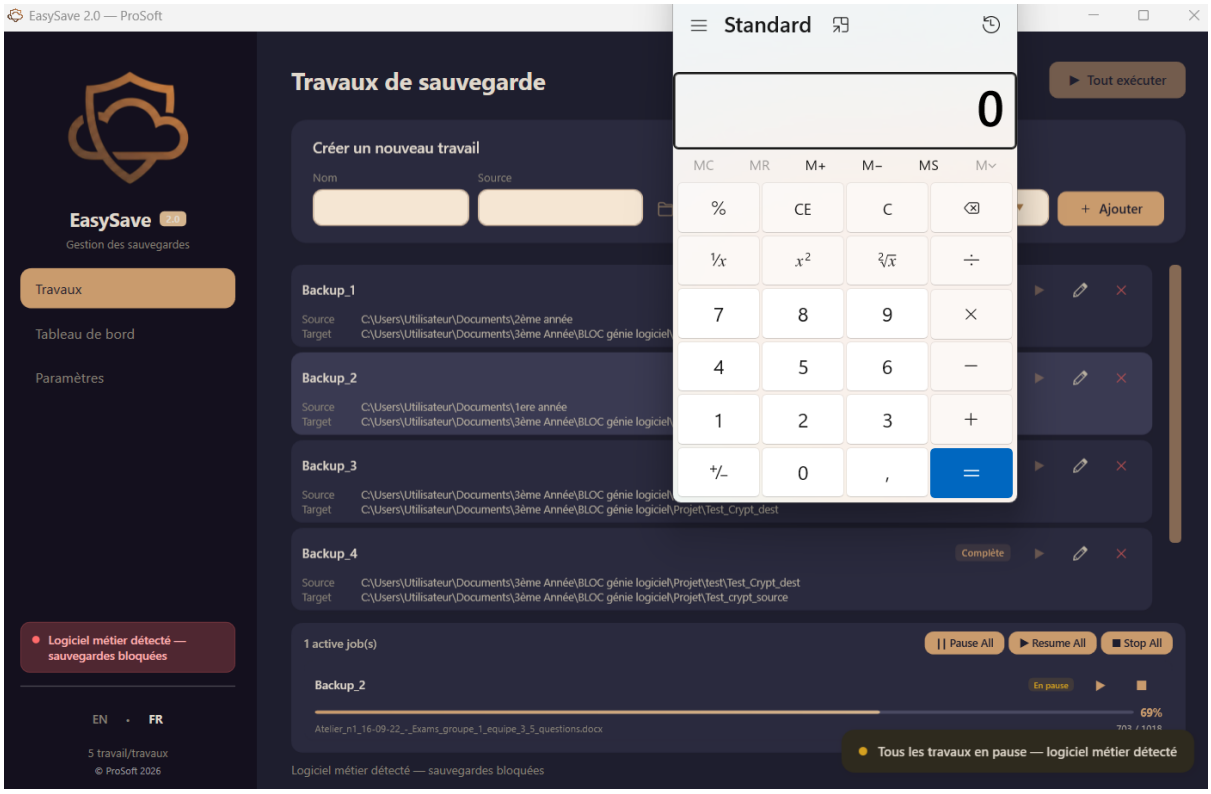
- Play (▶) : démarre ou reprend le travail après une pause
- Pause (⏸) : met en pause après la fin du transfert du fichier en cours
- Stop (■) : arrête immédiatement le travail et supprime la tâche

Des boutons globaux en haut de l'onglet permettent d'appliquer ces actions à tous les travaux simultanément.

6.3 Pause automatique si logiciel métier détecté

En v3.0, la détection du logiciel métier ne bloque plus uniquement le lancement : elle met automatiquement en pause TOUS les travaux en cours. Ils reprennent automatiquement dès que le logiciel métier est fermé.

Version	Comportement logiciel métier
v2.0	Bloque le lancement d'un nouveau travail. L'arrêt est loggué.
v3.0	Met en pause TOUS les travaux actifs. Reprise automatique à la fermeture du logiciel métier.

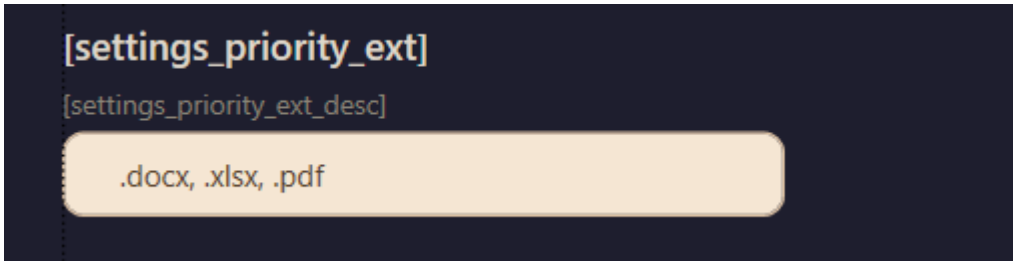


6.4 Gestion des fichiers prioritaires

L'utilisateur définit une liste d'extensions prioritaires dans les paramètres généraux (ex : .docx, .xlsx). Aucun fichier non prioritaire ne peut être transféré tant qu'il reste des fichiers à extension prioritaire en attente sur au moins un travail actif.

Mécanisme

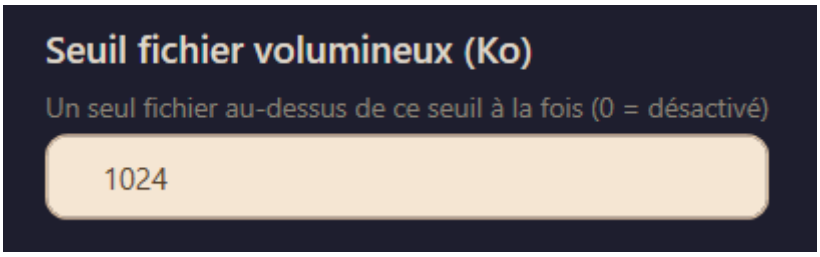
- Avant chaque copie de fichier non prioritaire : appel à PriorityCoordinator.WaitIfPriorityPendingAsync()
- Le PriorityCoordinator maintient un compteur atomique des fichiers prioritaires restants sur l'ensemble des jobs
- Quand ce compteur tombe à zéro : tous les WaitIfPriorityPendingAsync() en attente sont libérés



6.5 Limitation de bande passante

Pour éviter de saturer la bande passante, un seul fichier dépassant le seuil paramétrable (n Ko) peut être transféré à la fois. Les fichiers dont la taille est inférieure à ce seuil ne sont pas affectés par cette restriction (sous réserve des règles de priorité).

Taille du fichier	Comportement
< seuil (n Ko)	Transfert immédiat (sauf si fichier prioritaire en attente)
≥ seuil (n Ko)	Attend la libération du BandwidthLimiter (SemaphoreSlim(1))



6.6 CryptoSoft mono-instance

En v3.0, CryptoSoft.exe est protégé par un Mutex système global (Global\CryptoSoftMutex). Une seule instance peut s'exécuter à la fois, même si plusieurs travaux tentent de chiffrer simultanément. Les appels suivants attendent que le Mutex soit libéré.

Fonctionnement

- Au démarrage de CryptoSoft.exe : tentative d'acquisition du Mutex nommé
- Si le Mutex est déjà pris : attente (timeout confiugrable)
- EncryptionService utilise SemaphoreSlim(1) côté EasySave pour éviter d'envoyer deux appels simultanés à CryptoSoft
- Toute tentative simultanée est mise en file d'attente, non rejetée

6.7 Centralisation des logs via Docker

Si activée, cette fonctionnalité permet d'agréger les logs de toutes les machines client vers un seul fichier journalier sur un serveur Docker. Trois modes sont disponibles :

Mode (LogCentralizationMode)	Description
Local	Logs écrits uniquement sur le PC local (%APPDATA%\EasySave\Logs\)
Centralized	Logs envoyés uniquement au serveur Docker (aucun fichier local)
Both	Logs écrits localement ET envoyés au serveur Docker simultanément

Identification machine / utilisateur

Chaque entrée de log envoyée au serveur Docker inclut deux nouveaux champs :

- MachineId : nom de la machine (Environment.MachineName)

- UserId : nom de l'utilisateur Windows courant (Environment.UserName)

Architecture du service Docker

- Le service LogAggregatorService expose une API REST (POST /logs)
- Il reçoit les entrées de log en JSON depuis tous les clients
- Il écrit dans un fichier journalier unique (YYYY-MM-DD.json / .xml) indépendamment du nombre de machines
- Le service est configuré via docker-compose.yml (port, volume de persistance)


```
public string DockerUrl { get; set; } = "http://localhost:8080/api/logs";
```

7. Interface en ligne de commande (CLI)

La syntaxe CLI est identique aux versions précédentes. En v3.0, les travaux spécifiés en CLI sont exécutés en parallèle (et non plus de façon séquentielle).

Syntaxe	Exemple	Résultat
Simple	EasySave.WPF.exe 1	Exécute le travail 1
Plage	EasySave.WPF.exe 1-3	Exécute les travaux 1, 2 et 3 en parallèle
Liste	EasySave.WPF.exe 1;3	Exécute les travaux 1 et 3 en parallèle

8. Fichiers de configuration

8.1 config.json

Emplacement : %APPDATA%\EasySave\config.json

La v3.0 étend AppConfig avec trois nouveaux champs :

```
{
  "EncryptionKey": "Prosoft123",
  "EncryptionExtensions": [
    ".txt"
  ],
  "BusinessSoftwareName": "Calculatorapp",
  "LogFormat": 0,
  "LogMode": 2,
  "DockerUrl": "http://localhost:8080/api/logs",
  "LargeFileThresholdKo": 1024,
  "PriorityExtensions": [
    ".pdf"
  ],
  "Jobs": [
    {
      "Name": "Backup_1",
      "SourceDirectory": "C:\\Users\\Utilisateur\\Documents\\2\u00E8me ann\u00E9e",
      "TargetDirectory": "C:\\Users\\Utilisateur\\Documents\\3\u00E8me Ann\u00E9e\\BLOC g\u00E9nie logiciel\\Projet\\test\\tests",
      "Type": 1
    },
    {
      "Name": "Backup_2",
      "SourceDirectory": "C:\\Users\\Utilisateur\\Documents\\1ere ann\u00E9e",
      "TargetDirectory": "C:\\Users\\Utilisateur\\Documents\\3\u00E8me Ann\u00E9e\\BLOC g\u00E9nie logiciel\\Projet\\test\\tests2",
      "Type": 0
    },
```

Nouveaux champs v3.0

Champ	Type	Description
LargeFileSizeThresholdKo	int	Seuil en Ko au-dessus duquel un fichier est considéré 'volumineux' (défaut : 1024)
PriorityExtensions	string[]	Extensions de fichiers prioritaires (ex : .docx, .xlsx)
LogMode	string	0 = Local | 1 = Centralized | 2 = Both
DockerLogServerUrl	string	URL du service Docker de centralisation (http://localhost:8080/api/logs)

9. Fichiers de logs

9.1 Emplacement et nommage

Identique à la v2.0. Les fichiers logs locaux sont dans %APPDATA%\EasySave\Logs\.

9.2 Nouveaux champs v3.0 (logs centralisés)

Les entrées de log envoyées au serveur Docker incluent deux champs supplémentaires pour identifier la source :

Champ (NOUVEAU v3.0)	Description
MachineId	Nom de la machine source (Environment.MachineName)
UserId	Nom de l'utilisateur Windows courant (Environment.UserName)

10. Fichier d'état

Emplacement : %APPDATA%\EasySave\state.json

En v3.0, le fichier d'état contient simultanément l'état de TOUS les travaux en cours (tableau JSON).

10.1 Structure (multi-jobs)


```
{
  "Timestamp": "2026-02-24 11:16:03",
  "JobName": "Backup_3",
  "SourcePath": "C:\\Users\\Utilisateur\\Documents\\3\u00E8me Ann\u00E9e\\BLOC g\u00E9nie logiciel\\Projet\\test\\Test_crypt_source\\Bonjour CryptoSoft !.txt",
  "TargetPath": "C:\\Users\\Utilisateur\\Documents\\3\u00E8me Ann\u00E9e\\BLOC g\u00E9nie logiciel\\Projet\\Test_Crypt_dest\\Bonjour CryptoSoft !.txt",
  "FileSize": 20,
  "TransferTimeMs": 19,
  "EncryptionTimeMs": 3,
  "EventType": null,
  "EventDetails": null,
  "Username": "Utilisateur",
  "MachineName": "LAPTOP-1A34U3VH"
}

{
  "Timestamp": "2026-02-24 11:16:03",
  "JobName": "Backup_2",
  "SourcePath": "C:\\Users\\Utilisateur\\Documents\\1ere ann\u00E9e\\Bonjour CryptoSoft !.txt",
  "TargetPath": "C:\\Users\\Utilisateur\\Documents\\3\u00E8me Ann\u00E9e\\BLOC g\u00E9nie logiciel\\Projet\\test\\tests2\\Bonjour CryptoSoft !.txt",
  "FileSize": 20,
  "TransferTimeMs": 14,
  "EncryptionTimeMs": 1,
  "EventType": null,
  "EventDetails": null,
  "Username": "Utilisateur",
  "MachineName": "LAPTOP-1A34U3VH"
}
```


10.2 Valeurs d'état

Valeur	Description
Inactive	Travail non démarré
Active	Travail en cours d'exécution
Paused	Travail mis en pause (par l'utilisateur ou par détection logiciel métier)
Completed	Travail terminé avec succès
Stopped	Travail arrêté par l'utilisateur (bouton Stop)
Error	Le travail a rencontré une erreur

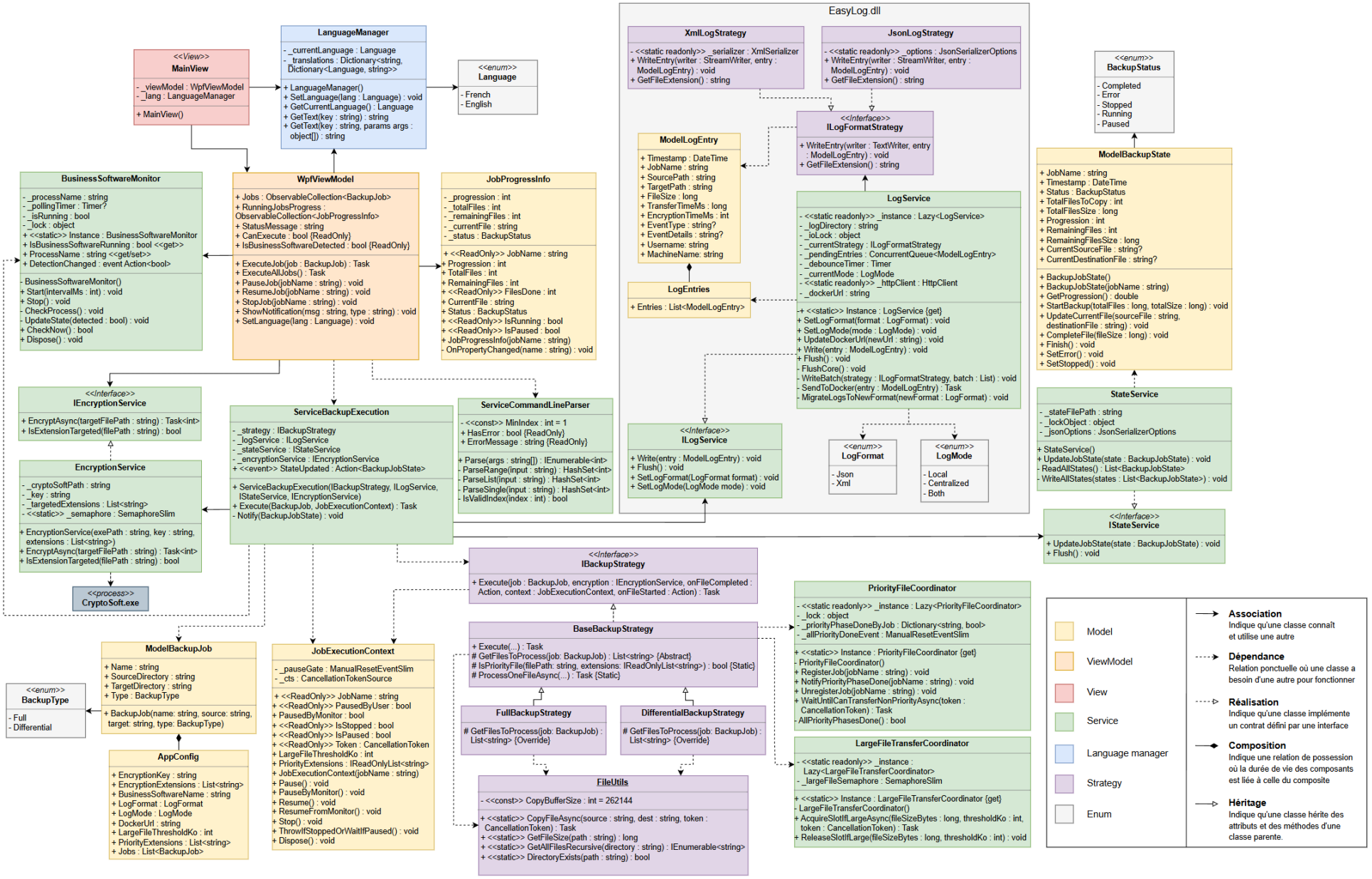
```
namespace EasySave.Core.Models.Enums
{
    // ===== BACKUP STATUS =====
    46 références
    public enum BackupStatus
    {
        Inactive,
        Active,
        Completed,
        Error,
        Stopped,
        Running,
        Paused
    }
}
```

11. Dépannage

Problème / Message	Cause probable	Solution
Jobs en pause imprévue	Logiciel métier détecté	Fermer le logiciel métier ou modifier le paramètre dans Paramètres > Général
Fichiers volumineux jamais transférés	BandwidthLimiter bloqué par un autre job	Vérifier qu'aucun autre job n'est bloqué (Stop puis relancer)
Fichiers non prioritaires jamais transférés	PriorityCoordinator en attente infinie	Vérifier que les fichiers prioritaires existent bien dans la source
Chiffrement lent / en attente	CryptoSoft mono-instance saturé	Normal : les chiffrements sont sérialisés. Réduire le nombre d'extensions ciblées.
Logs absents sur Docker	Service Docker arrêté ou URL incorrecte	Vérifier docker-compose (docker ps), vérifier DockerLogServerUrl dans Paramètres
config.json corrompu	Modification manuelle invalide	Supprimer %APPDATA%\EasySave\config.json et redémarrer
CryptoSoft introuvable (-99)	CryptoSoft.exe absent du dossier	Placer CryptoSoft.exe à côté de EasySave.WPF.exe

12. Diagrammes UML

12.1 Diagramme de classes — évolutions v3.0


EasyLog.dll
MainView
LanguageManager
Language
WpfViewModel
BusinessSoftwareMonitor
JobProgressInfo
XmlLogStrategy
JsonLogStrategy
ILogFormatStrategy
ModelLogEntry
LogEntries
LogService
ILogService
LogFormat
LogMode
BackupStatus
ModelBackupState
StateService
IStateService
IEncryptionService
EncryptionService
CryptoSoft.exe
ServiceBackupExecution
ServiceCommandLineParser
IBackupStrategy
BaseBackupStrategy
FullBackupStrategy
DifferentialBackupStrategy
PriorityFileCoordinator
LargeFileTransferCoordinator
FileUtils
ModelBackupJob
BackupType
AppConfig
JobExecutionContext
Model
ViewModel
View
Service
Language manager
Strategy
Enum
Association
Indique qu'une classe connaît et utilise une autre
Dépendance
Relation ponctuelle où une classe a besoin d'une autre pour fonctionner
Réalisation
Indique qu'une classe implémente un contrat défini par une interface
Composition
Indique une relation de possession où la durée de vie des composants est liée à celle du composite
Héritage
Indique qu'une classe hérite des attributs et des méthodes d'une classe parente.

1. Architecture MVVM : L'architecture repose sur un découplage strict entre l'interface WPF (Vue) et la logique métier via le WpfViewModel. Ce dernier agit comme le point d'entrée central : il traite les commandes utilisateur (ICommand), gère une collection de BackupJob et utilise le ServiceCommandLineParser au démarrage pour l'automatisation par scripts. Grâce au Data Binding et à INotifyPropertyChanged, toute modification dans BackupJobState est répercutée instantanément sur l'interface.

2. Moteur d'Exécution (Pattern Strategy) : La logique de sauvegarde est abstraite par l'interface IBackupStrategy.

- BaseBackupStrategy : Classe abstraite contenant le moteur de transfert commun, la gestion des priorités et le contrôle des pauses.
- Full & Differential strategies : Implémentations concrètes spécialisées dans le filtrage des fichiers (tout copier ou seulement les modifications). Cette structure permet d'étendre facilement le logiciel à de nouveaux modes de sauvegarde (Cloud, Zip) sans modifier le code existant.

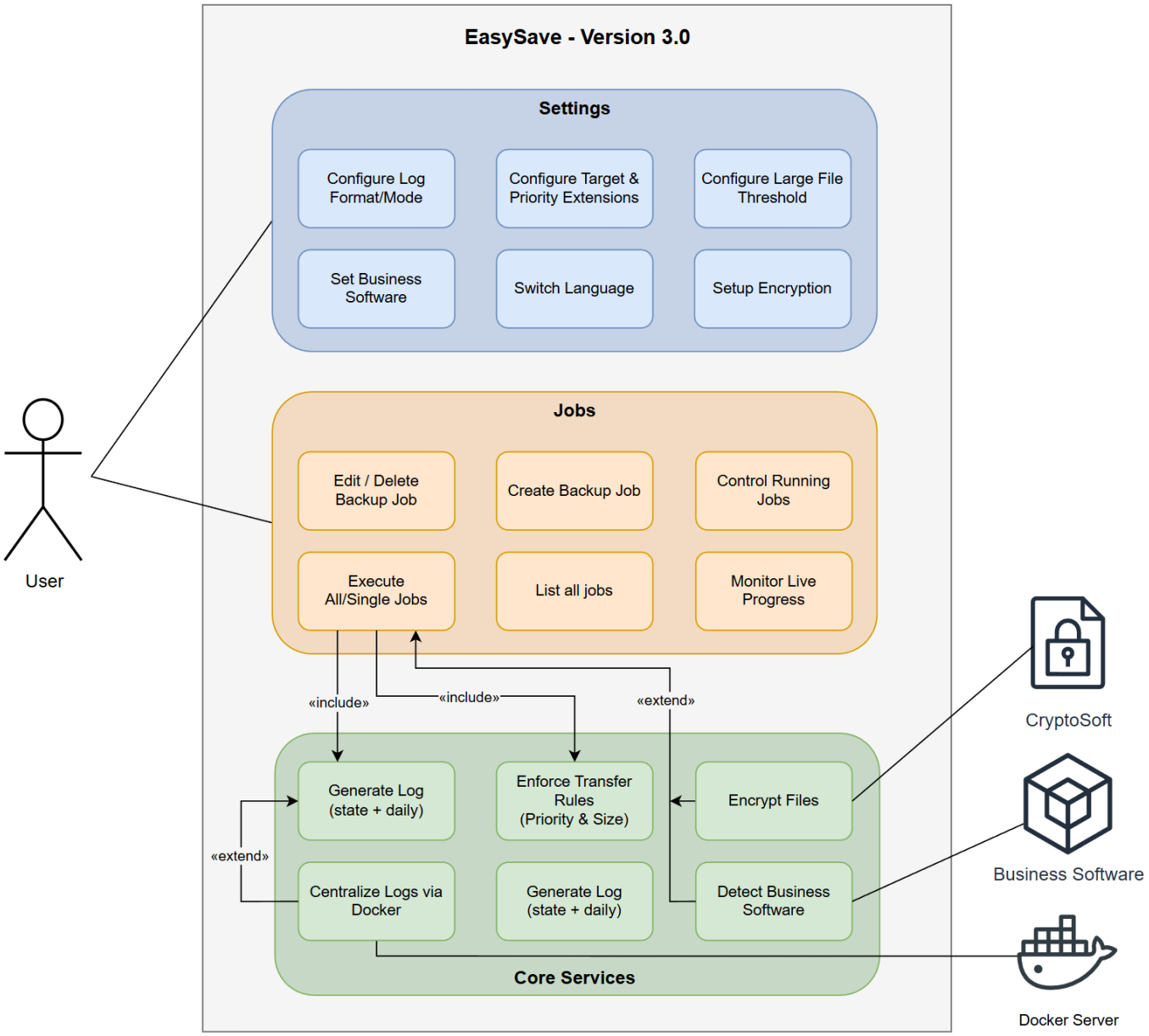
3. Coordination et Supervision (Singletons) : Trois services transverses sous forme de Singletons assurent l'intégrité du système en mode parallèle :

- LargeFileTransferCoordinator : Gère le sémaphore limitant le transfert simultané de fichiers volumineux.
- PriorityFileCoordinator : Synchronise les jobs pour garantir que les fichiers prioritaires passent avant les fichiers normaux sur l'ensemble du parc.
- BusinessSoftwareMonitor : Surveille les processus système et notifie le ViewModel pour suspendre les stratégies actives.

4. Services Externes et Persistance

- EncryptionService : Encapsule l'interaction avec CryptoSoft.exe en respectant la politique de mono-instance.
- LogService : Service unique responsable de l'écriture des fichiers (JSON/XML) et de la transmission réseau vers le serveur Docker via des requêtes HTTP.
- FileUtils : Bibliothèque statique utilitaire pour les opérations d'entrée/sortie physiques sur le disque.

12.2 Diagramme de cas d'utilisation



1. Paramétrage et Configuration (Settings) L'utilisateur dispose d'un contrôle total sur l'environnement d'exécution via un bloc de paramètres dédiés :

- Règles métier : Définition des extensions prioritaires, du seuil de taille pour les fichiers volumineux (Threshold) et du logiciel métier à surveiller.
- Sécurité et Langue : Configuration de la clé de chiffrement pour CryptoSoft et choix de la langue de l'interface.
- Logs : Choix du format de sortie (JSON/XML) pour la traçabilité locale.

2. Gestion et Contrôle des Travaux (Jobs) L'interface permet de gérer le cycle de vie des travaux de sauvegarde :

- CRUD : Création, édition et suppression des travaux de sauvegarde.
- Pilotage en temps réel : L'utilisateur peut lancer, mettre en pause ou arrêter un ou plusieurs travaux simultanément.
- Supervision : Un retour visuel permanent permet de monitorer la progression en direct pour chaque job actif.

3. Automatisation des Services Cœurs (Core Services) Le lancement d'une sauvegarde (Execute) déclenche une chaîne d'actions automatiques incluant plusieurs services critiques :

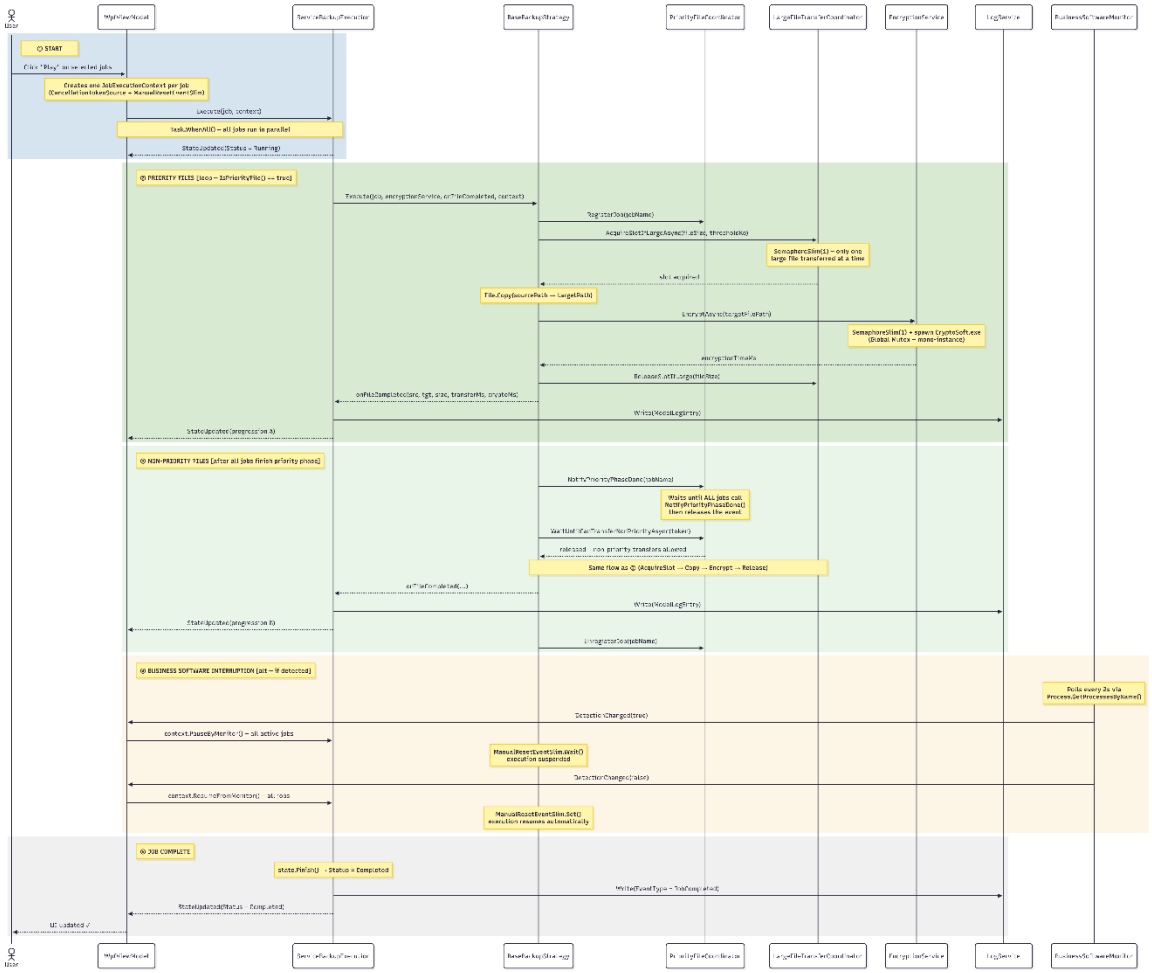
- Application des règles (include) : Chaque exécution inclut systématiquement la génération de logs (State & Daily) et l'application des règles de transfert (priorités et limitation des fichiers volumineux).
- Chiffrement (include) : Si l'extension du fichier le demande, le système fait appel à CryptoSoft pour sécuriser les données.
- Centralisation (extend) : Selon la configuration, le service peut étendre la journalisation vers le Serveur Docker pour une supervision déportée.

4. Interactions Externes Le système EasySave communique avec trois entités externes pour garantir son fonctionnement industriel :

- CryptoSoft : Pour l'exécution du chiffrement XOR.
- Logiciel Métier : Pour la mise en pause automatique via le module de détection.
- Serveur Docker : Pour la réception et le stockage centralisé des logs via le réseau.

12.3 Diagramme de séquence

Le diagramme ci-dessous représente le flux d'exécution d'un travail de sauvegarde en v3.0, de l'action utilisateur jusqu'à la fin du job.



1. **Start** — L'utilisateur lance les jobs via *WpfViewModel.RunAllJobsAsync*().
   Un *JobExecutionContext* est créé par job (contenant un *CancellationTokenSource* pour le Stop et un *ManualResetEventSlim* pour le Pause/Resume). Tous les jobs démarrent en parallèle via *Task.WhenAll*().

2. **Fichiers prioritaires** — BaseBackupStrategy traite en premier les fichiers dont l'extension figure dans PriorityExtensions. Avant chaque copie,

   LargeFileTransferCoordinator impose via un SemaphoreSlim(1) qu'un seul fichier volumineux (> LargeFileThresholdKo) transite à la fois sur l'ensemble des jobs.

   Si le fichier doit être chiffré, EncryptionService sérialise l'appel à CryptoSoft.exe (Mutex global mono-instance). Chaque fichier traité déclenche un Write(ModelLogEntry) sur LogService et un StateUpdated(progression%) vers le ViewModel.

3. **Fichiers non prioritaires** — Une fois sa phase prioritaire terminée, chaque job appelle PriorityFileCoordinator.NotifyPriorityPhaseDone(), puis attend sur WaitUntilCanTransferNonPriorityAsync().

   Le coordinateur (Singleton) ne libère cette attente que lorsque tous les jobs ont notifié leur fin de phase prioritaire, garantissant qu'aucun fichier normal ne passe devant un fichier prioritaire en attente sur un autre job.

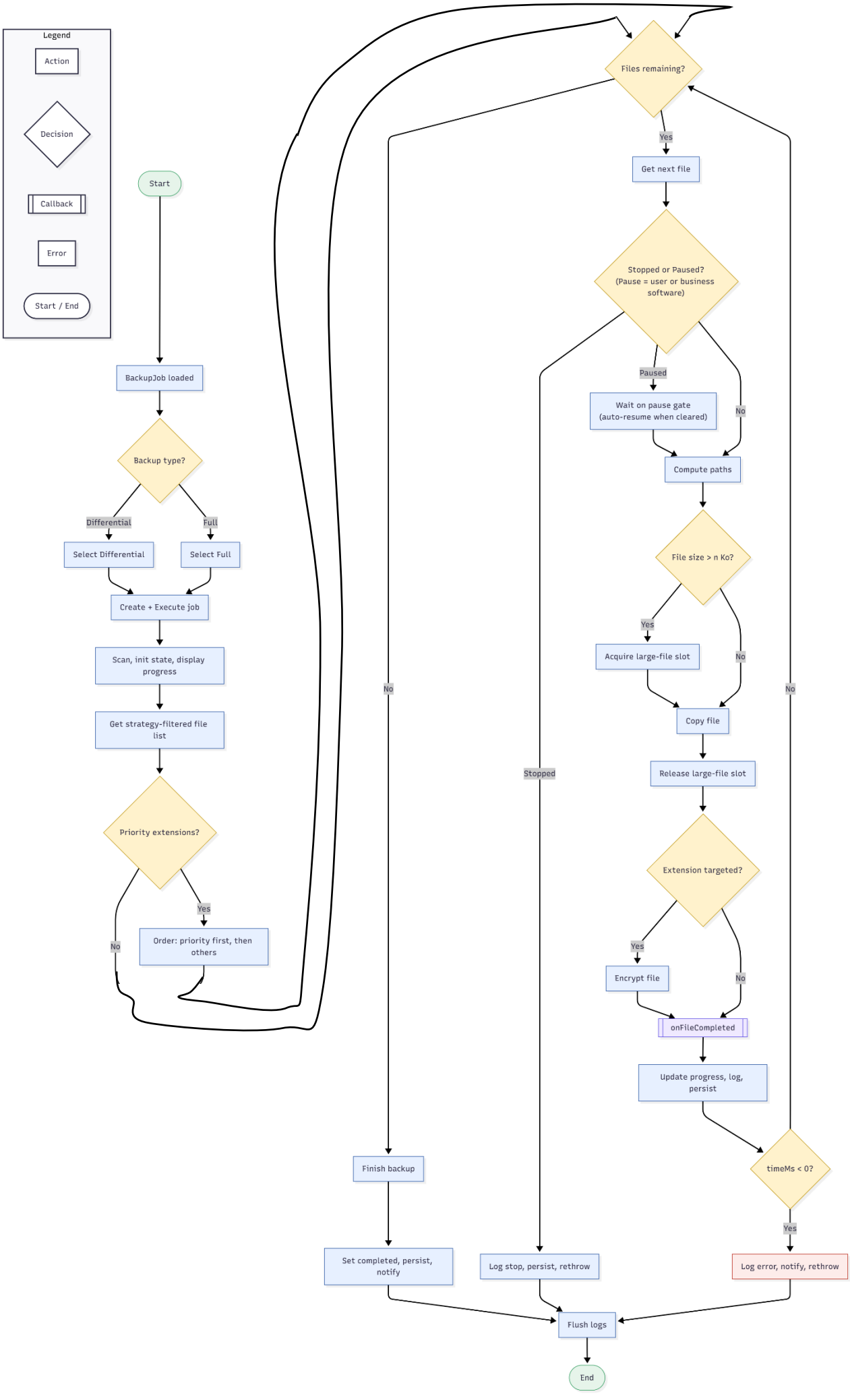
4. **Interruption logiciel métier** — BusinessSoftwareMonitor (Singleton) poll toutes les 2s.

   - À la détection, il émet DetectionChanged(true) : WpfViewModel appelle context.PauseByMonitor() sur chaque job actif, bloquant la stratégie sur ManualResetEventSlim.Wait().

   - À la fermeture du logiciel métier, la reprise est automatique via ManualResetEventSlim.Set().

5. **Fin** — state.Finish() passe le statut à Completed. LogService enregistre l'entrée finale et StateUpdated(Completed) met à jour l'interface

12.4 Diagramme d'activité


Legend
Action
Decision
Callback
Error
Start / End
Start
BackupJob loaded
Backup type?
Differential
Full
Select Differential
Select Full
Create + Execute job
Scan, init state, display progress
Get strategy-filtered file list
Priority extensions?
Yes
No
Order: priority first, then others
Files remaining?
Yes
Get next file
Stopped or Paused? (Pause = user or business software)
Paused
Wait on pause gate (auto-resume when cleared)
No
Compute paths
File size > n Ko?
Yes
Acquire large-file slot
No
Copy file
Release large-file slot
Extension targeted?
Yes
Encrypt file
No
onFileCompleted
Update progress, log, persist
timeMs < 0?
Yes
Log error, notify, rethrow
No
Finish backup
Set completed, persist, notify
Stopped
Log stop, persist, rethrow
Flush logs
End

Déroulement d'une sauvegarde — version 3.0

Lorsqu'une sauvegarde est déclenchée, ServiceBackupExecution commence par lire le répertoire source afin de calculer le nombre total de fichiers et leur taille cumulée. Ces informations initialisent l'état du travail (BackupJobState), qui est persisté et diffusée dans l'interface via l'événement StateUpdated. La stratégie de sauvegarde est choisie en amont (complète ou différentielle) et injectée dans le service : c'est elle qui ordonne la liste des fichiers réellement à transférer.

Avant de commencer le transfert des fichiers non prioritaires, la stratégie interroge le PriorityFileCoordinator (singleton partagé entre tous les travaux en cours). Tant qu'au moins un travail actif n'a pas terminé ses fichiers prioritaires, les autres travaux se mettent en attente sur un verrou (ManualResetEventSlim). Ce mécanisme garantit que les fichiers avec extensions prioritaires dans les paramètres sont toujours traitées en premier, tous travaux confondus.

Pour chaque fichier à traiter, le contexte d'exécution (JobExecutionContext) est consulté afin de détecter une demande de pause ou d'arrêt. Une pause déclenchée manuellement par l'utilisateur ou automatiquement par la détection d'un logiciel métier via le BusinessSoftwareMonitor, suspend le thread du travail jusqu'à ce que le verrou soit rouvert. Un arrêt lève une OperationCanceledException qui interrompt le travail et inscrit l'événement dans le log.

Si le fichier dépasse le seuil de taille configuré (n Ko), le LargeFileTransferCoordinator impose l'acquisition d'un slot exclusif avant le transfert : cela empêche que deux fichiers volumineux soient copiés simultanément sur le réseau. Dès le transfert terminé, le slot est relâché pour les autres travaux.

Une fois le fichier copié, si son extension est dans la liste des extensions à chiffrer, EncryptionService appelle CryptoSoft — rendu mono-instance par un mutex — et retourne le temps de chiffrement. Le callback onFileCompleted est ensuite appelé : il met à jour l'état de progression, écrit une entrée dans le fichier log journalier (via EasyLog.dll) avec horodatage, chemins source/cible, taille, temps de transfert et temps de chiffrement, puis notifie l'interface. En cas d'erreur (temps de transfert négatif), le travail passe en erreur et l'exception est propagée. Dans tous les cas, succès, arrêt ou erreur, le bloc finally garantit un flush du log et de l'état, assurant qu'aucune donnée n'est perdue.

13. Glossaire

Terme	Définition
Travail de sauvegarde	Tâche configurée avec source, cible et type
Sauvegarde parallèle	Exécution simultanée de plusieurs travaux dans des threads différents
CancellationToken	Primitive .NET permettant l'annulation coopérative d'une tâche asynchrone (Stop)
ManualResetEventSlim	Primitive .NET permettant la mise en pause / reprise d'une tâche (Pause/Play)
PriorityCoordinator	Singleton synchronisant les règles d'extensions prioritaires entre tous les jobs
BandwidthLimiter	SemaphoreSlim(1) global limitant le transfert simultané de fichiers volumineux
Fichier volumineux	Fichier dont la taille dépasse le seuil paramétrable (n Ko)
Extension prioritaire	Extension déclarée par l'utilisateur ; les fichiers correspondants sont transférés en priorité
Mono-instance	Contrainte garantissant qu'un seul processus CryptoSoft s'exécute à la fois (Mutex système)
LogCentralizationMode	Enum : Local | Docker | LocalAndDocker
DockerLogSender	Client HTTP envoyant les logs au service d'agrégation Docker
LogAggregatorService	Service Docker recevant les logs de N machines et écrivant un fichier journalier unique
MachineId / UserId	Identifiants ajoutés aux logs centralisés pour différencier les sources
AppData	Dossier de données d'application utilisateur Windows (%APPDATA%)
WPF	Windows Presentation Foundation — framework UI de Microsoft
MVVM	Patron de conception Model-View-ViewModel